

LN: A Flexible Algorithmic Framework for Layered Queueing Network Analysis

GIULIANO CASALE, YICHENG GAO, ZIFENG NIU, and LULAI ZHU, Department of Computing, Imperial College London, UK

Layered queueing networks (LQNs) are an extension of ordinary queueing networks useful to model simultaneous resource possession and stochastic call graphs in distributed systems. Existing computational algorithms for LQNs have primarily focused on mean-value analysis. However, other solution paradigms, such as normalizing constant analysis and mean-field approximation, can improve the computation of LQN mean and transient performance metrics, state probabilities, and response time distributions. Motivated by this observation, we propose the first LQN meta-solver, called LN, that allows for the dynamic selection of the performance analysis paradigm to be iteratively applied to the submodels arising from layer decomposition. We report experiments where this added flexibility helps us to reduce the LQN solution errors. We also demonstrate that the meta-solver approach eases the integration of LQNs with other formalisms, such as caching models, enabling the analysis of more general classes of layered stochastic networks. Additionally, to support the accurate evaluation of the LQN submodels, we develop novel algorithms for homogeneous queueing networks consisting of an infinite server node and a set of identical queueing stations. In particular, we propose an exact method of moment algorithms, integration techniques for normalizing constants, and a fast non-iterative mean-value analysis technique.

CCS Concepts: • **General and reference** → **Performance**; • **Computing methodologies** → **Modeling and simulation;**

Additional Key Words and Phrases: Layered queueing networks, meta-solver, computational algorithms, performance measures, multi-formalism

ACM Reference format:
Giuliano Casale, Yicheng Gao, Zifeng Niu, and Lulai Zhu. 2024. LN: A Flexible Algorithmic Framework for Layered Queueing Network Analysis. *ACM Trans. Model. Comput. Simul.* 34, 3, Article 17 (July 2024), 26 pages. https://doi.org/10.1145/3633457

1 INTRODUCTION

Layering is a widespread feature of computer and communication networks, which can be analyzed using stochastic models such as **layered queueing networks (LQNs)** [25, 54]. LQNs build upon extended queueing network models to enable the analysis of simultaneous resource possession and stochastic call graphs in distributed systems. The underpinning stochastic models

The development of the LINE and LN has been partially funded by the European Commission grants FP7-318484 (MODAClouds), H2020-644869 (DICE), H2020-825040 (RADON), and by the EPSRC grant EP/M009211/1 (OptiMAM). The work of Yicheng Gao has been supported by an Imperial College - Chinese Scholarship Council (IC-CSC) studentship.
Authors' address: G. Casale, Y. Gao, Z. Niu, and L. Zhu, Department of Computing, Imperial College London, 180 Queen's Gate, SW7 2AZ, London UK; e-mails: {g.casale, y.gao20, zifeng.niu19, lulai.zhu15}@imperial.ac.uk.



This work is licensed under a Creative Commons Attribution International 4.0 License.

© 2024 Copyright held by the owner/author(s).
1049-3301/2024/07-ART17 $15.00
https://doi.org/10.1145/3633457

are typically analyzed by decomposition methods, which map the original LQN into an ensemble of interacting submodels, each being an extended mixed queueing network featuring open and closed job classes [24]. The submodels can then be solved iteratively, updating their input parameters across the iterations until the performance metric estimates converge.

Many performance evaluation paradigms exist for evaluating the mixed queueing networks associated with each LQN submodel, ranging from **stochastic simulation (SSA)**, **continuous-time Markov chain analysis (CTMC)**, **normalizing constant evaluation (NC), matrix analytic methods (MAM), mean-field approximations (Fluid)**, and **mean-value analysis (MVA)** [14]. However, the current scientific literature on LQNs emphasizes only a few, mainly MVA methods [25] and, more recently, mean-field approximations [41, 50]. We argue that the availability of multiple solution paradigms motivates the development of a richer approach to solving LQNs, namely, a meta-solver framework that can dynamically select the most appropriate algorithm to analyze each submodel arising from LQN decomposition. In particular, we show numerical evidence that this added flexibility helps in reducing the maximum LQN solution error compared to using a single submodel evaluation paradigm.

To build upon this observation, we present LN, an open-source meta-solver for LQN analysis. We define a layered network meta-solver as a framework that builds upon a collection of independent solvers to analyze each network layer in isolation. A key goal of a meta-solver is to maintain abstraction in the interaction with these solvers. This enables greater generality, extensibility, and fewer dependence on solver-specific concepts and methods than in existing LQN solvers. LN builds its meta-solver capability on top of a suite of ordinary queueing network solvers implemented in the LINE software package, which offers several tens of exact and approximate solution algorithms for extended queueing networks [15]. In LN, the submodels passed to LINE solvers are generated using loose layering [24, Section3.2] and thus consist of a single infinite-server node and m replicated queueing stations, i.e., stations having identical parameterization in every class of jobs. We refer to such models as *homogeneous* queueing networks. For such models, we develop specialized methods that are more accurate, or more efficient, than existing solution techniques for ordinary queueing networks, providing a case for systematically adopting in LN the loose layering decomposition style.

Our main contributions to the theory of homogeneous queueing networks are threefold. First, we propose a method of moments algorithm for solving *exactly* homogeneous models in linear time as the total population size grows. This is first developed for single server queueing stations. An extension to the multiserver case is also introduced, which can quickly solve large multiclass models with significant speedups compared to MVA. For example, we report on the ability of these algorithms to solve exactly multiclass models with thousands of jobs in milliseconds and without requiring exact algebra implementations, which is a crucial gain compared to the existing method of moment algorithms [12, 13].

Next, we develop Gaussian quadrature methods for approximating the normalizing constants of homogeneous models. By controlling the order of the quadrature, these methods can trade accuracy for speed while remaining efficient in models with large population sizes. Moreover, integral expressions are more straightforward to use than recurrence relations on some models, such as those with fractional populations.

Last, we introduce a non-iterative MVA-based approximation for homogeneous models, which allows for a fast model solution in return for slightly degraded accuracy in the submodel evaluation. Although not as accurate as iterative methods, this new technique is significantly faster and offers much greater accuracy than comparable non-iterative approximations, such as asymptotic bounds [23, 34], which are sometimes used to initialize LQN submodels.

The contribution is organized as follows: Sections 2 and 3 describe related work and background, respectively. Section 4 gives motivation for meta-solution approaches in LQNs. Section 5 introduces novel solution algorithms for homogeneous queueing networks. Experiments and case studies are presented in Section 6. Finally, Section 7 is dedicated to the conclusions. Proofs are given in the Appendix. Additional material on the software architecture of the LN tool is given in the online supplement. Compared to a preliminary version of this work [16], this article formalizes the meta-solver approach and demonstrates its benefits using random experiments. The article also develops the novel non-iterative approximation in Section 5.4 and extends in Section 5.2 the method of moment approach to multiserver stations.

2 RELATED WORK

Several state-of-the-art solvers specific to LQNs have been proposed over the years, including for example LQNS [25] and DiffLQN [50, 51]. LQNS is an established solver with many applications to real-world software engineering case studies. At heart, the tool applies to the LQN layers approximate mean-value analysis for extended queueing network models [25]. The toolchain also offers LQN analysis based on generalized stochastic Petri nets via its petrirsrvn tool, which relies on the GreatSPN [2] tool, and via discrete-event simulation, through the lqsim tool. We point to Reference [25] for a comparative analysis of LQNS with earlier tools.

DiffLQN is instead a solver that is based on the mean-field approximation theory developed in the context of PEPA models to scalably analyze LQNs [50, 51]. The mean-field fluid paradigm is particularly suited to the solution of large models, as it becomes asymptotically exact in layers with multiserver FCFS stations once the number of jobs and servers grows large in a fixed ratio. Subsequent work on mean-field approximations has further generalized the fluid solutions to **processor sharing (PS)** stations, class switching, random environments, response time percentiles [41], differentiated service weights [57], multiclass FCFS approximations [15], and mixed models [47].

The LINE tool is the main software baseline for this work [15]. The package offers exact, approximate, asymptotic, and simulation-based analysis of open, closed, and mixed multiclass queueing network models. Some solution methods in LINE are also dependent on external solvers that include JMT [5], LQNS [25], BuTools [29], KPC-Toolbox [19], and Q-MAM [6]. Similar tools in the literature include Octave queueing [35] and PDQ [28].

LN is built around the experience of these solvers, integrating many of the approximate MVA and Fluid methods used in tools such as LQNS and DiffLQN. In addition, LN enables the analysis of LQNs using the other solution paradigms (e.g., CTMC, NC, MAM) that have seldom been considered in the LQN literature. Moreover, the tool generalizes the LQN formalism to include models of cache replacement policies, which are unavailable in other solvers.

3 BACKGROUND AND PRELIMINARIES

3.1 Extended Queueing Networks

In this article, LQNs are decomposed into collections of ordinary (i.e., non-layered) queueing network models, each representing a specific subset of client-server interactions within the layered network. In these submodels, jobs are probabilistically routed across a set of nodes, primarily queueing stations, where they receive service, possibly subject to contention by other jobs. Each job belongs to a *class*, i.e., a type that defines its service, routing, and arrival characteristics at every node. The set of reachable classes for a job defines its *chain*. Typically, routing among the stations is probabilistic, and a job can *switch* class upon departing from a station, before reaching the next one, by traveling through specialized class-switching nodes.

Compared to standard product-form queueing networks [4], the submodels we consider here can feature extensions commonly required in applications. In particular, the LN solver uses the class of extended queueing networks considered in Reference [15], which includes product-form queueing networks [4] and their extensions to model priorities, load-dependence, class-switching, and instantaneous transitions. A difficult problem is to obtain in such models station and system performance measures such as average queue lengths, utilization, response times, throughputs, and arrival rates. Several analytical and simulation-based methods have been proposed over the years to analyze extended queueing network models of this kind. We briefly overview in the next subsections the MVA, NC, and Fluid paradigms.

The tool also leverages the caching extensions developed in Reference [15]. We point to References [8, Sections 8–10], [15], and [26] for an introduction and discussion of these features, several of which are standard in the performance evaluation. Throughout, we focus on models without caching and priorities, emphasizing instead analysis methods for networks with PS, FCFS, or infinite-server stations, which are the most common in LQN applications. We briefly illustrate in Section 6.2 the support in our solver for LQNs that feature caching.

3.1.1 Mean Value Analysis. The MVA paradigm is rooted in the arrival theorem, which offers a recurrence relation for mean response times in product-form queueing networks [44]. The subsequent development of the Linearizer [22] has led to many accurate approximation schemes for both product-form and extended queueing network models. An overview of such methods, commonly referred to as approximate MVA algorithms, is available in Reference [8, Section10]. Such methods are often fixed-point iterative approximations, which typically converge in milliseconds even on large-scale models. Due to this efficiency, LQN theory has been largely based on the approximate MVA theory [25].

3.1.2 Normalizing Constant Methods. Probabilistic analyses typically require a different approach. The evaluation of marginal and joint distributions for the CTMC underpinning the queueing network normally requires to obtain the normalizing constant that ensures that state probabilities sum to one. A large number of **normalizing constant (NC)** methods, starting from the classic convolution algorithm [11], are available. They include both recursive and asymptotic approximations; we point to Reference [14] for an overview. To our knowledge, NC methods have not been considered in past works on LQN analysis.

3.1.3 Mean-field Approximation. We refer to the Fluid paradigm as the analysis style for queueing networks consisting of techniques derived from Kurtz's mean-field approximation theory; see Reference [9] for a tutorial. Queueing network applications of these methods can scalably approximate large models, offering estimates for both transient and steady-state metrics [41, 47, 50]. However, such methods can face numerical difficulties in the presence of stiff ordinary equations as well as incur some inaccuracies in models with low parallelism levels. Recent works such as References [27, 42] are progressively reducing the severity of the last problem.

3.2 Layered Queueing Networks

Layered queueing networks (LQNs) are a rich class of extended queueing network models that have found application in software performance engineering, distributed system analysis, and modeling of stochastic networks, among other fields [3, 25, 49]. Compared to ordinary queueing networks, the distinguishing feature of LQNs is the ability to model stochastic workflows and simultaneous resource possession. Due to their long history, a number of variants and specializations of these models have appeared in the literature, e.g., networks with simultaneous resource possession [30], semi-open queueing networks [31], and stochastic rendezvous networks [54]. We

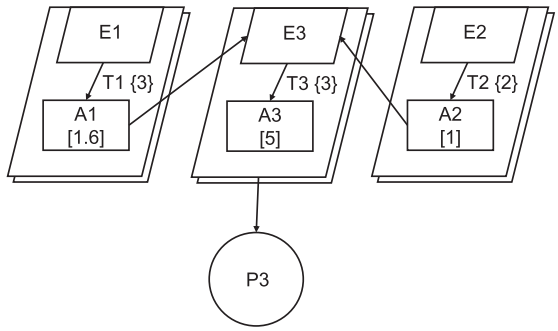


Fig. 1. Example of an LQN model.

here only introduce the essential background concepts on the LQN formalism and point the reader to Reference [53] for more details.

In an LQN, jobs issued by clients pay visits to a network of queueing resources following a workflow modeled as a **directed acyclic graph (DAG)**. The DAG includes *calls* made by the job to *entries* exposed by *tasks* running on *host processors*. For example, an LQN model of a software system may use calls to describe service invocations, entries to describe service endpoints, tasks to describe software servers, and host processes to describe hardware resources. Informally, these concepts bear some resemblance to the standard queueing theoretic notions of routing (calls), classes (entries), and stations (tasks and host processors), as we discuss in more detail in Section 3.3.2, yet they enable the description of a more complex dynamics on a layered network topology.

Upon invocation of an entry e through a call, a DAG-based workflow of one or more *activities* (i.e., service phases) is executed until completion of the call. This workflow is called the *activity graph* bound to the entry e. Within this workflow, an activity may issue a synchronous call to an entry of another task while keeping a server in its parent task blocked, leading to simultaneous resource possession. Asynchronous calls and forwarding calls are also possible; we point to Reference [53] for additional details.

Figure 1 shows a small-scale example that summarizes the above definitions for the basic elements of LQN models: *tasks*, *host processors*, *entries*, and *activities*. Tasks are depicted as stacked parallelograms. A task runs on a single processor (e.g., P3), represented by a circle. Specific services provided by a task are called entries and drawn as smaller parallelograms at the top of the task. Each rectangle denotes a particular activity performed during the execution of an entry. The number between square brackets specifies the service demand of the activity (e.g., 1.6 for activity A1). Jobs in LQNs are generated by a special task termed the reference task, e.g., tasks T1 and T2 in Figure 1, which model two classes of 3 and 2 users, respectively, both calling entry E3 exposed by task T3. The number between curly brackets represents the server multiplicity of the task, which is the maximum execution parallelism for a task, e.g., T3 has a pool of 3 servers that can process calls to its entry E3. The processors of T1 and T2 are not drawn for conciseness. Larger models are typical in applications; we point to References [25, 45, 46] and the bibliography in Reference [1] for further examples.

3.3 Decomposition-based Analysis Techniques

Extensive prior work in the area has shown that an LQN can be accurately solved by iteratively evaluating a collection of ordinary mixed queueing networks, obtained using decomposition, until reaching a fixed-point iterative solution [25]. Each decomposed submodel conceptually represents a layer of the system under study [46]. The collection of submodels forms an interactive ensemble,

in the sense that the output parameters of a model for a lower layer, such as the response times at the queueing stations, become an input parameter of the upper layer models, typically service times at the queueing stations. We now briefly discuss the strategy to divide a given LQN into multiple layers and basic definitions concerning the obtained submodels that have been adopted in the proposed solver.

3.3.1 Layering Strategy. Layering strategies are methods for decomposing an LQN model into an ensemble of ordinary queueing networks. Prior work has noted a relatively weak sensitivity of the LQN solution accuracy based on the layering style [24, Section3.2]. In the proposed evaluation solver, we adopt loose layering [24, Section3.2], which ensures that each layer includes a single queueing station, representing either a task or host processor, coupled with an infinite-server node to model the inter-request times of clients. The station may be replicated, with stations having identical demands.

For example, under loose layering decomposition, the model in Figure 1 yields four submodels: T1→ P1, T2→P2, T3→P3, and (T1,T2)→T3, where → indicates a client-server relationship. Note that each of these four models describes a single server in the LQN (P1, P2, P3, T4), thus corresponding submodels do not exist for the reference tasks (T1, T2), which are pure clients issuing calls to the system.

The rationale for choosing loose layering as the default strategy in our solver, unlike the default choice in existing tools such as LQNS or DiffLQN, while some queueing network algorithms such as MVA display linear complexity as a function of the number of stations m within a submodel, other paradigms rely on computations that have polynomial or exponential complexity with respect to m. While the asymptotic complexity of methods such as MVA is nearly unaffected[1] if one solves m homogeneous models with a single type of station, rather than a monolithic model comprising m different queueing stations, methods of the latter kind are computationally more efficient in analyzing smaller models. For example, a CTMC solver may be fairly scalable in models with a single queue, but quickly incurs state space explosion with multiple stations. A similar issue is displayed by Monte Carlo sampling methods in the NC approach, which approximate a multidimensional integral in m dimensions and thus are quite sensitive to the growth of this parameter [52]. A drawback of loose layering is that certain features, such as heterogeneous load balancing or fork/join sections, are somewhat inconvenient to model, as the participating stations may be scattered across different submodels, requiring specific solutions to cope with this difficulty.[2]

3.3.2 Submodel Generation. We now outline the mapping between LQN abstractions and the above models. Under loose layering, we abstract a layer l, which models a specific server in the LQN, as consisting of an extended queueing network with m identical stations, modeling the server and its replicas, and a single infinite-server station, modeling the client think times. Thus, m denotes the replication factor of the server. We shall refer to such a queueing network as a *homogeneous* model. In more detail, the homogeneous models we consider are mixed queueing networks with m identical c-server queues and a delay (i.e., infinite-server) node. Such networks can be mapped using standard techniques into closed queueing networks after suitable correction of the service demands of the queueing stations [8]. Jobs in class r have service demand D_r at the

[1]This applies to exact MVA methods. For approximate MVA methods, linear complexity holds at each iteration, but the number of iterations can be affected by m, although typically rather weakly.

[2]For example, to address situations of this kind, if the response time of a station in layer l' needs to be evaluated within layer $l \leq l'$, then it is possible to add a surrogate infinite server to layer l to represent such a station, iteratively synchronizing its think time with the response time observed in l'. Adding such a new infinite server station to layer l does not significantly change the submodel structure, since these nodes can normally be aggregated with little approximation error.

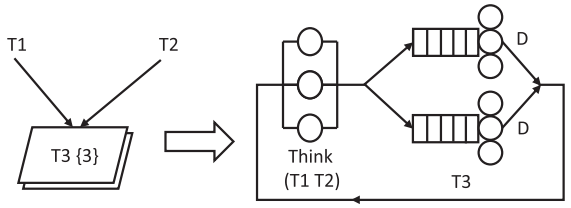


Fig. 2. A layer of the LQN model with replication $m = 2$ and multiplicity $c = 3$.

multiserver station and think time Z_r at the infinite-server station. We allow service distributions to be either phase type or degenerate distributions, in particular the *immediate* distribution, which models zero processing times. The handling of the latter typically relies in our solver on the stochastic complement technique that allows hiding selected transitions in a Markov chain [37]; we point to Reference [12] for more details.

The number of servers in the queueing stations is equal to the multiplicity c of the server in that layer. Figure 2 gives the model of a layer for the LQN where T3 acts as a server. Clients issuing synchronous calls to layer l are represented as jobs initialized in a *reference class* at the delay. In the example of Figure 2, these may be classes T1 and T2. Subsequently, jobs cycle between the delay and the queues, switching their class to represent the specific tasks, entries, and activities that the clients visit (or invoke) during execution. For example, according to Figure 1, in the submodel for server T2, a job starting in reference class T1 will first switch to class E1, then to class A1, spending time in these classes at the delay station of the submodel. Then, the job will be routed to the server into a class labeled A1→E3 to distinguish it from similar calls to E3 issued by jobs starting in the other reference class T3, which is also present in the submodel. Calls issues from the latter would be in a class labeled A2→E3. Last, the job will complete at the station in class A1→E3 and return into class T1 at the delay.

Parameters such as think times and service demands are iteratively updated as per the method of layers [46]. For example, if certain sections of the client activity graph require access to another layer $l' \neq l$, then the corresponding execution times are modeled as think times in the submodel for l that already incorporate the queueing contention in layer l'. Moreover, the probability of a client executing a particular entry is set proportional to the last throughput of this entry. We then update routing probabilities at each iteration, because not all solution paradigms are visit-based.

Clients that send asynchronous calls to layer l are instead represented as open Poisson arrival streams. The coexistence of open and closed classes, therefore, gives rise to mixed models, which can be reduced to closed ones by demand scaling [8]. This allows us to analyze mixed models directly using solution paradigms for closed queueing networks.

Service classes are mapped to a set of R chains, obtained by computing the strongly connected components of the routing matrix. Each chain j represents a client task to the layer and has an associated number of jobs N_j that are initialized at the infinite-server station, starting in the chain reference class. Note that this does not lose information, as it is possible to exactly recover the per-class performance metrics from the per-chain ones [8, 55].

For ease of presentation, since a multi-chain model can always be reduced to a corresponding multiclass model with R classes, one per chain, we shall use the terms "chain" and "class" interchangeably.

3.3.3 Performance Metrics. Performance metric computation is solver-dependent. We illustrate their definitions here taking the NC approach as a reference. We assume for simplicity that

scheduling leads to product-form models and single server stations ($c = 1$). Moreover, we give definitions for cases where the station is a queueing resource, but similar definitions apply when this is instead an infinite server node. Approximations to handle other cases, such as multiclass FCFS scheduling, are discussed later in the article.

Let $N = (N_1, \ldots, N_R)$ be the population vector for a layer, $|N| = \sum_{r=1}^{R} N_r$, and recall that $G(m, N)$ is the normalizing constant of the state probabilities for the associated product-form model, which consists of m identical queueing stations and an infinite-server node. Denote by 1_r a row vector of all zeros with a one in the rth dimension. We may exploit the following relations for the mean class-r throughput $X_r(N)$ and for the mean class-r queue length $Q_r(N)$ at any of the identical queueing stations:

$$X_r(N) = \frac{G(m, N - 1_r)}{G(m, N)} \qquad Q_r(N) = D_r \frac{G(m + 1, N - 1_r)}{G(m, N)}.$$

The system throughput $X_r(N)$ is assumed to be computed at a reference station, i.e., the infinite server, for which we set the mean number of visits of class r to unity. Little's law may then be combined with the previous relations to obtain other metrics at the queueing station, such as mean response times, mean residence times, and resource utilization [23, 34]. Let V_r be the mean number of visits that a job in class r pays to the queueing station. The class-r residence time $R_r(N)$ is the mean time accumulated by class r in visits to the queue prior to completing at the reference station, i.e., $W_r(n) = Q_r/X_r$. The response time is then the average residence time per visit, i.e., $W_r(N) = R_r(N)/V_r$. Last, the utilization is the percentage of time the processor is busy, given by $U_r(N) = X_r(N)D_r$. In the case of c-server stations, we assume $U_r(N)$ to be always re-scaled by c to range in $[0, 1]$, except if the station is an infinite server; in which case, we assume that $U_r(N) = Q_r(N)$, since $c \to \infty$.

Before discussing the novel algorithms proposed to compute these metrics, we remark that specific simplifications arise in evaluating normalizing constants for homogeneous models due to the structure of the product-form solutions. At first, if either $Z_r = 0 \wedge D_r = 0$ or $N_r = 0$ holds for a class r, then this class can be removed from the model, as it does not contribute to the normalizing constant. Define $\mathcal{R}_D$ as the set of remaining classes for which $D_r = 0$. We note that the contribution of such classes to $G(m, N)$ is given exactly by a factor $\prod_{s \in \mathcal{R}_D} Z_s^{N_s}/N_s!$, which can be efficiently computed. Hence, every model can be reduced without loss of generality to one where all classes have $D_r > 0$, which we will assume throughout.

3.3.4 Fixed Point LQN Solution. Solutions of interactive models in an ensemble are reconciled through fixed-point iterations until performance metrics across the layers are consistent within a numerical tolerance. To this end, after topological sorting, the LQNS solver adopts an elevator algorithm whereby the graph that describes the client-server relationships is traversed top down and bottom up in alternation, thus cyclically inverting the order in which the layers are analyzed [24]. This algorithm is also used in our LN solver to iterate over all the models within an ensemble.

Convergence testing in LN proceeds as follows: At each iteration, and for each submodel l, the largest change in per-class mean queue-length between iterations i and $i + 1$ is recorded as $\Delta_l^{i+1} = \max_r(Q_r^{(i)} - Q_r^{(i+1)})/N_r$, where $Q^{(i)}$ denotes the mean queue-length of class r obtained at iteration i. If there are L submodels, then LN stops when $\Delta^{i+1} = \sum_{l=1}^{L} \Delta_l^{i+1}$ falls below a given tolerance. If this is not achieved within a maximum number of iterations, then LN assumes that input parameters are oscillating across iterations and applies a moving window is applied to the residence time and through estimates at each layer to ease convergence. If this is not reached within a maximum number of iterations, then the algorithm returns the current results.

Table 1. Solver Selection Statistics for 20,000 Randomly Generated Homogeneous Models with FCFS and PS Stations

Solver	*Server multiplicity*			*Class populations*			*Total percentage*
	Low	*Mid*	*High*	*Low*	*Mid*	*High*	
MVA	26.625	1.895	1.210	20.500	6.735	2.495	29.730
NC	35.655	22.830	11.580	45.190	18.465	6.410	70.065
Fluid	0.125	0.055	0.025	0.200	0.005	0.000	0.205

4 LQN META-SOLUTION

4.1 Motivating Example

To compare different solution methods for solving closed queueing network models with varying characteristics, we investigate a set of 20,000 random models. Each model consists of a delay and a queue, with the scheduling strategy for each queue being randomly chosen as either FCFS or PS. The service time distribution for each job class is chosen uniformly at random to be either exponential, hyper-exponential, or Erlang. Note that the latter distributions admit product-form solution under PS but not under FCFS except for the case where exponential service times are identical [4], which occurs in our random models only in single-class networks. The models are designed to handle low, medium, and high loads, with the total job population ranging in [1,10] (low), [11,30] (medium), and [31,40] (high), respectively.

Each model is solved using the MVA, NC, and Fluid solvers available in LINE[3] 2.0.27. To determine the optimal solution for each model, we measure the performance of each method using a combined metric that considers both accuracy and execution time, computed through a weighted geometric mean of the two factors with coefficients 0.75 and 0.25, respectively. Specifically, the accuracy metric is the arithmetic mean of the absolute relative errors on the system response time and the absolute relative error on the system throughput, computed relatively to discrete-event simulation values. We use for the simulation the JSIMgraph tool, part of the JMT suite [5]. The solution method that exhibits the maximal accuracy metric is selected. The resulting distribution of the selected solution paradigm across all 20,000 models is illustrated in Table 1.

The results shown in Table 1 reveal that the majority of models in LINE are best solved by the NC and MVA methods, with Fluid being the best solver only in approximately 0.2% of models. Specifically, Table 1 indicates that the NC and MVA methods both exhibit comparable performance for small-scale models, with a difference of 9.03%. However, for medium-scale and large-scale models, the NC solver performs better than the MVA method. However, Table 1 demonstrates that the NC method outperforms the MVA method for models serving either low, medium, or high workloads, with an advantage of 24.69%, 11.73%, and 3.91%, respectively. The Fluid model is fairly accurate, but it is worse than MVA and NC in solution speed. This is because integrating the ODEs is fairly expensive, especially when the differential equations are stiff. For example, consider a cyclic closed queueing network model including a delay station with unit think times and two FCFS nodes. There are two classes of jobs with 10 jobs each. Class 1 has mean demand 1.5 at both queues, while class 2 has demand 3.0 at both stations. Averaging execution times over 100 repetitions, MVA and NC complete in about 16 ms, while the Fluid solver takes about 42 ms and displays similar, but slightly lower, accuracy than MVA and NC, due to the fact that the model has single server stations and therefore does not perfectly align to the assumptions of Kurtz's theorem [41]. Nevertheless, the Fluid solver remains a useful tool for transient analysis and response time distribution evaluation, as illustrated later in Section 6.2.

[3]Version available at: https://github.com/imperial-qore/line-solver

Table 2. LN Automated Paradigm Selection Heuristics in LINE 2.0.27

Paradigm	*Selected if:*	*Main Methods*
NC	• Single class models • Multiclass models with single server queues • Small population models (including multiservers)	• CoMoM, Section 5.1 • Multiserver CoMoM, Section 5.2 • Gauss-Legendre, Section 5.3 • RD heuristic [18]
MVA	• Multiclass models with multiserver queues • High or medium population models • Non-product-form models	• QD-AMVA [20] • Linearizer [21] • Non-iterative approximation, Section 5.4

It should be noted that the above conclusions hold for the specific implementations in the considered version of LINE. However, they illustrate the fact that no solver dominates the other, since each between MVA and NC can handle best a large fraction of models. This gives a case for developing automated methods for selecting the best algorithms to run within meta-solver approaches to leverage individual paradigm strengths.

In summary, the NC and MVA methods are generally better suited for solving most models, with the choice between the two methods, depending on the expected load as well as the scale of the model. The Fluid method may be suitable for a small percentage of models, yet as we show later in Section 6, it still offers good average performance with low maximal error.

4.2 LN Meta-solver

Stemming from the previous motivating example, the LINE **Layered Network solver (LN)** is a concrete implementation of a meta-solver approach to LQN analysis. All the solution paradigms available in LINE, including analytical methods (NC, MVA, MAM, and Fluid), discrete-event simulation (JMT, SSA), and CTMC analysis, can thus be leveraged to evaluate LQNs. Given an input LQN model, LN first produces an ensemble of submodels according to the principles discussed in Section 3. An initial assignment of performance metrics is made to each submodel assuming no queueing, after calculating essential quantities such as the visit ratios of client entries to tasks. Upper bounds to the multiplicity levels of the servers are also applied at this stage at infinite server stations to ensure that multiplicity coefficients are all finite. Then, a set of solvers is prepared for the analysis of the generated submodels.

After the initialization phase, LN starts to iteratively solve the ensemble of submodels, updating the model parameters after each iteration. The order in which the submodels are processed is arbitrary, and a submodel may be skipped at a certain iteration if its parameter changes are negligible. In particular, the current implementation adopts the same ordering as in the LQN elevator algorithm [24]. Each time the ensemble of submodels is analyzed and modified, a global update of the solver parameters ensues. The iterative solution procedure stops and returns the obtained results after detecting convergence of the performance metrics within a threshold or after exceeding an iteration limit.

Compared to existing solvers, such as LQNS, the main differences are the selection of the solver to be applied to each submodel and the range of algorithms within each solver. The process of selecting a solver for a submodel is either chosen manually by the end-user or automated via heuristics. Table 2 illustrates the main heuristic selection criteria used in LN shipped with LINE version 2.0.27. Given that non-product-form models are difficult to characterize exactly, it is challenging to come up with universal rules to decide when a method is better than another; thus, the heuristic rules in the table summarize our empirical observations from the execution of the solvers on random instances. The conditions for selection are applied top-down; for example, a small population model will always be processed by the NC solver, also when it features multiclass multiserver stations.

Each solver offers a selection of several algorithms; a selection of the main methods is highlighted in the table. At present, LINE offers about 70 different algorithms across its solvers, each applicable to a specific family of submodels. An inner selection function is applied within each solver to choose which algorithm to run based on the characteristics of the model. Several algorithms are introduced in this article in the sections listed in the table. The logic for selecting algorithms within each solver is hidden to the LN meta-solver, which interacts with the individual solvers only via abstract interfaces; we point to the online appendix for more details on the specification of such interfaces. The selection rule of MVA prioritizes QD-AMVA on models where some classes have fractional populations with less than one job; otherwise, the Linearizer algorithm is chosen. Instead, the NC solver uses the Gauss-Legendre presented later in Section 5.3 as default and the Logistic Expansion proposed in Reference [14] for very large models. The CoMoM and multiserver CoMoM exact methods we propose later in Sections 5.1 and 5.2 can also be selected by the user for additional accuracy, and the latter is the default for multiclass homogenous models with a multiserver queue.

In LINE MVA solver, Linearizer is applied by default to models with homogeneous single server queues. QD-AMVA is implemented similarly to Reference [20] and used by default to handle homogeneous multiserver PS queues, which are treated in this approximation as stations with load-dependent rates. QD-AMVA is also preferred to Linearizer to approximate models with very low population, including those with less than 1.0 job in some classes. This is because the QD-AMVA queue-length interpolation is class-independent, therefore avoiding some inaccuracies that arise in this case when using Linearizer. Linearizer is instead used to approximate multiserver FCFS stations based on either Seidmann's approximation or the Rolia-Franks approximation [25, 46, 48].

In LINE NC solver, the default methods are the algorithms presented in the next section, in particular, the Gauss-Legendre integration is used by default unless the model has multiserver stations, in which case, multiserver CoMoM is preferred. In addition to these, the RD method is used to handle the corrections required to handle load-dependent stations [18].

5 ALGORITHMS FOR HOMOGENEOUS QUEUEING NETWORKS

From now on, we focus on LQN decomposition based on loose layering and present the algorithmic innovation developed to increase accuracy and performance of the solution algorithms for individual LQN layers.

5.1 CoMoM Algorithm for Homogeneous Models with Single Server Queues

Recall that mixed queueing network models can be mapped with suitable transformation to a model consisting only of closed classes [8, Section 8.2.3]. A closed product-form network solution approach that demonstrates a specific advantage of relying on the NC paradigm instead of mean-value analysis is the **Class-oriented Method of Moments (CoMoM)** algorithm proposed in Reference [12] for exact analysis and several approximations derived from integral forms for the normalizing constant $G(\boldsymbol{m}, \boldsymbol{N})$ that enable probabilistic analyses, e.g., asymptotic expansions [14, 36]. We now describe the extensions developed for the two families of algorithms.

We now propose a method of moments algorithm that exactly solves homogeneous models in linear time as the total population size grows, assuming models with single server queueing stations. As opposed to existing methods such as MoM [13] and CoMoM [12] that can *theoretically* achieve linear complexity, the proposed technique is the first one that realizes this in concrete implementations by avoiding the use of exact arithmetic, which introduces overheads up to about log-linear in the total population size [12, 13]. Moreover, it does so without solving systems of linear equations usually appearing in methods of moment algorithms. The result allows the efficient solution of submodels arising from loose layering in milliseconds. Also, it serves as a

basis for efficient approximations for more complex networks with multiserver stations [18, 48] and for non-product-form models. We also discuss here an extension for the exact analysis of homogeneous models of multiserver nodes.

For a model with N jobs belonging to a fixed number of R classes, CoMoM implementations require approximately log-quadratic time and log-linear space in $|N|$ to obtain an *exact* solution, thus being more scalable than the exact MVA algorithms. The latter has a time and space complexity of $O(|N|^R)$. Moreover, contrary to other moment-based methods, CoMoM can avoid degeneracies when the model consists of one or more replicated queueing stations, as in the case of loose layering. An equivalent result is not currently available for MVA. Among the complications, it is worth noting that MVA expressions such as the celebrated arrival theorem are bi-linear in their defining terms, mean queue lengths, and mean throughputs, yielding systems of non-linear equations that are not as tractable as CoMoM linear matrix recurrence relation.

We enhance CoMoM by developing explicit solutions to its system of linear equations for homogeneous models. Such solutions apply to models with an arbitrary number of classes R. For a vector v with d dimensions, let $|v| = \sum_{i=1}^{d} v_i$ and define $\text{diag}(v)$ as a diagonal matrix with the elements of v placed on the main diagonal. We now give an exact result.

Theorem 5.1. *Consider a product-form queueing network model with R classes, having m identical single server queueing stations with service demand $D_r > 0$ in class r and an infinite-server node with think time Z_r in class r. Define the collection of normalizing constants*

$$g(m, N) = \begin{bmatrix} G(m, N) & G(m, N - 1_1) & \cdots & G(m, N - 1_{R-1}) \end{bmatrix}$$

and the basis

$$\Lambda(N) = \begin{bmatrix} g(m + 1, N) & g(m, N) \end{bmatrix}^T.$$

Then, the following matrix recurrence relation holds:

$$\Lambda(N) = (\boldsymbol{F}_{1,R} + N_R^{-1} \boldsymbol{F}_{2,R}) \Lambda(N - 1_R) \tag{1}$$

in which

$$\boldsymbol{F}_{1,R} = \begin{bmatrix} D_R \boldsymbol{E}_{1,1} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix} \quad \boldsymbol{F}_{2,R} = \begin{bmatrix} m D_R \boldsymbol{S} & Z_R \boldsymbol{S} \\ m D_R \boldsymbol{I} & Z_R \boldsymbol{I} \end{bmatrix}$$

$$\boldsymbol{S} = -m^{-1} \begin{bmatrix} -|\tilde{N}| - m & \tilde{Z}^T \\ -\text{diag}(\tilde{D})^{-1} \tilde{N} & \text{diag}(\tilde{D})^{-1} \text{diag}(\tilde{Z}) \end{bmatrix},$$

where $\boldsymbol{E}_{1,1}$ is of order R with a single nonzero entry in position $(1, 1)$, $\widetilde{N} = (N_1, \ldots, N_{R-1})^T$, $\widetilde{D} = (D_1, \ldots, D_{R-1})^T$, $\widetilde{Z} = (Z_1, \ldots, Z_{R-1})^T$, $\boldsymbol{I}$ is the identity matrix of order R, and $\boldsymbol{0}$ is the zero matrix of order R.

A proof of the theorem is in Appendix A. Note in particular that the knowledge of $\Lambda(N)$ and $\Lambda(N - 1_R)$ is sufficient to determine the expression of all the mean performance metrics introduced in Section 3.3.3. Termination conditions for (1) are obtained noting that $G(\cdot, 0) = 1$ and that, whenever any element of N is negative, $G(\cdot, N) = 0$.

In terms of numerical stability, to prevent floating-point range exceptions, the proposed solution can either be computed using exact or multi-precision arithmetic. In practice, we have observed that scaling at each step the vector $\Lambda(N)$ so $|\Lambda(N)| = 1$, and removing the effect of such scaling only in the final result is sufficient to sanitize numerical problems in practical uses, except for negligible numerical fluctuations. This makes the theoretical and implementation complexity identical and, empirically, much faster than using exact arithmetic.

Last, we remark that the result in this section assumes integer populations, since for fractional values an exact CTMC solution is undefined. In this case, we solve the models with integer

Table 3. Relative Error and Runtime upon Computing $\log G(m, N)$ Exactly

Classes	*Total jobs*	*Method*	*Runtime [s]*
8	40	Convolution	0.0033
8	40	CoMoM (original)	0.0047
8	40	CoMoM (enhanced)	0.0014
8	400	Convolution	1.4201
8	400	CoMoM (original)	1.1433
8	400	CoMoM (enhanced)	0.0016
8	4,000	Convolution	Memory exhausted
8	4,000	CoMoM (original)	Timeout
8	4,000	CoMoM (enhanced)	0.0017
8	10^6	Convolution	Memory exhausted
8	10^6	CoMoM (original)	Timeout
8	10^6	CoMoM (enhanced)	0.2591

populations $\lceil N \rceil$ and $\lfloor N \rfloor$ and linearly interpolate the mean performance metrics. For example, the mean class-r queue-length would be estimated as $Q_r(N) = Q_r(\lfloor N \rfloor) + \eta(Q_r(\lceil N \rceil) - Q_r(\lfloor N \rfloor))$, with $\eta = N - \lfloor N \rfloor$. Similar expressions can be used for the other metrics.

5.1.1 Numerical Example. To illustrate the performance of the enhanced CoMoM algorithm, we consider a challenging model with $R = 8$ classes, where $m = 1$, $Z_r = r$ and $D_r = 10^{-r}$, $r = 1, \ldots, R$. Jobs are split equally across the classes. The exponential spacing of the demands and the large population make the analysis numerically challenging. We set a timeout of 10 seconds to solve a model. The original CoMoM leverages exact arithmetic in Java, whereas the enhanced method implements in MATLAB the recursion we have proposed in Theorem 5.1 using standard floating-point arithmetic. Table 3 shows numerical results, which corroborate the high scalability of the enhanced CoMoM for homogeneous models. Results are obtained on an AMD Ryzen 7 2700X processor with 64 GB of RAM. Note that at population $|N| = 4,000$ convolution becomes unviable due to excessive memory requirements, but, since the normalizing constant reaches order 10^{-602}, it would have exceeded the floating-point range during execution. Scaling methods for Convolution have been proposed in Reference [33], however, it is not difficult in our experience to generate examples of large models where this technique still cannot prevent floating-point range exceptions. Instead, the enhanced CoMoM can also solve the largest model with 10^6 jobs, agreeing within the first 11 digits of $\log G(m, N)$ with the results of the **logistic expansion (LE)** proposed in Reference [14], which is asymptotically exact. We have also observed in all cases that at least the first six digits of the mean per-class throughputs computed by the enhanced CoMoM were identical to the ones obtained by the AQL approximate MVA method [56]. As no other exact method can reach this model scale, it is difficult to rigorously verify exactness, yet the result suggests no, or at least negligible, presence of error accumulation.

5.2 CoMoM Algorithm for Homogeneous Models with Multiserver Queues

Using normalizing constants instead of MVA simplifies the calculation of probabilistic measures on each layer, as illustrated in this section. Thanks to loose layering, specialized results can be derived to allow simple computation of marginal probabilities in a layer, which enables the exact analysis of load-dependent service behavior, which encompasses multiserver stations as a special case [8, 18]. We focus here in particular on computing using the CoMoM approach the marginal probability $\pi_N(m, n)$ that n jobs are queueing or receiving services at any of the m identical queueing stations.

This is also equal to the probability that $\pi_N(m, |N| - n)$ jobs are waiting at the infinite server station.

Computing $\pi_N(m, n)$ is generally a difficult problem, since with R classes there is a combinatorially large number of job mixes that result in the same total job population n at the m queueing nodes. In this case, our evaluation solver leverages a novel result, developed in the next theorem, which obtains marginal probabilities in $O(|N|^2)$ time and $O(|N|)$ space when applied to homogeneous models. For $m = 1$, this improves over MVA methods that require instead $O(|N|^R)$ time and space, while matching the complexity of CoMoM extension to marginal probabilities [12, Section-VII], but without requiring the solution of a system of linear equations as in the original CoMoM method. Other key novelties are that, unlike the original method, the expression below applies also to homogeneous models with $m > 1$ and with load-dependent service.

Theorem 5.2. *Consider a product-form queueing network model with R classes, having m identical c-server queueing stations with service demand $D_r > 0$ in class r, and an infinite-server node with think time Z_r in class r. Let $\pi_N(m, k)$ be the marginal probability that the m queueing stations have k resident jobs in total, $k = 0, \ldots, |N|$. Define the following basis of unnormalized probabilities:*

$$\Pi(N) = G(m, N) \left[\pi_N(m, |N|), \ldots, \pi_N(m, 0)\right]^T$$

with $\Pi(0) = (0, \ldots, 0, 1)^T$. Then, $G(m, N) = |\Pi(N)|$ and the following exact recurrence relation holds:

$$\Pi(N) = N_R^{-1} \boldsymbol{T}_R \Pi(N - 1_R), \tag{2}$$

in which

$$\boldsymbol{T}_R = \begin{bmatrix} Z_R & |N|\dfrac{D_R}{\mu(|N|)} & 0 & \cdots & 0 \\ 0 & Z_R & (|N|-1)\dfrac{D_R}{\mu(|N|-1)} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & Z_R & \dfrac{D_R}{\mu(1)} \\ 0 & 0 & 0 & 0 & Z_R \end{bmatrix}$$

and where the load-dependent rate $\mu(n)$ can be computed as the throughput of a single-class product-form queueing network consisting of m identical c-server queueing stations, each having unit service demand.

Proof of this exact result is given in Appendix D. Note that the service rates $\mu(n)$ can be precomputed offline in $O(|N|^2)$ time, for example, using the load-dependent MVA algorithm [10], as their values are independent of R, D_r, and Z_r, $1 \le r \le R$. Note that this preliminary step has the same asymptotic complexity as the proposed algorithm for growing values of $|N|$.

5.2.1 Numerical Results. We have verified with numerical examples, comparing with the load-dependent MVALD-MX [10], that the formulas in Theorem 5.2 match the exact solution while avoiding the exponential time and space requirements of traditional methods as the number of classes grows. Applying the same scaling as the one used for the multiserver CoMoM, we are able also to solve large models. For example, a model with $R = 8$ classes, $|N| = 40$ jobs, $m = 2$ identical PS queues with $c = 2$ servers, similar to the first row of Table 4, is solved exactly by MVALD-MX in 238 seconds, occupying about 76 MB of memory, while the recurrence relation in Theorem 5.2 completes in about 1.5 milliseconds with negligible memory consumption.

Table 4. Relative Error and Runtime upon Approximating $\log G(m, N)$

Classes	*Total jobs*	*Method*	*Rel. error [%]*	*Runtime [s]*
8	40	MATLAB integral	0.0000	0.0006
8	40	Gauss-Legendre	0.0000	0.0004
8	40	Gauss-Laguerre	0.0000	0.0010
8	40	Logistic expansion	−0.1249	0.0012
8	400	MATLAB integral	0.0144	0.0005
8	400	Gauss-Legendre	−0.0001	0.0006
8	400	Gauss-Laguerre	−0.0001	0.0010
8	400	Logistic expansion	0.0033	0.0013
8	4, 000	MATLAB integral	Unstable	0.0008
8	4, 000	Gauss-Legendre	−0.0006	0.0021
8	4, 000	Gauss-Laguerre	−0.0006	0.0010
8	4, 000	Logistic expansion	0.0003	0.0013
8	10^6	MATLAB integral	Unstable	0.0008
8	10^6	Gauss-Legendre	−0.0592	0.0095
8	10^6	Gauss-Laguerre	0.2508	0.0011
8	10^6	Logistic expansion	0.0000	0.0013

5.3 Gaussian Quadrature Methods

As illustrated in the numerical examples, the CoMoM method for single server stations faces a slight increase in time requirements as the population grows. This also occurs as R increases, since the CoMoM basis has $2R$ elements. While tens or hundreds of milliseconds may be negligible for a single model, LQNs are solved iteratively and can feature many layers, hence solution times compound quickly. In large models, it can be useful to trade accuracy for speed.

A simple expression for the normalizing constant is given by the McKenna-Mitra integral form [36]. This is generally a multidimensional integral, with one dimension for each queueing station in the model. Thus, in a homogeneous model for a layer, one would expect a m-dimensional integral. We show, however, that inhomogeneous models this integral form takes the following simpler expression:

Theorem 5.3. *Under the same assumptions of Theorem 5.1, the normalizing constant of state probabilities for the queueing network model admits the following integral form:*

$$G(m, N) = \frac{1}{(m-1)! \prod_{r=1}^{R} N_r!} \int_{u=0}^{+\infty} e^{-u} u^{m-1} \prod_{r=1}^{R} (Z_r + D_r u)^{N_r} du. \tag{3}$$

A proof is given in Appendix C. The main difficulty associated with evaluating $G(m, N)$ directly is that Equation (3) is prone to numerical difficulties. This is because quadratures do not operate directly in the log domain and are therefore numerically sensitive to the magnitude of the factors under the integration sign, one being an exponentially decaying function (e^{-u}), the other being a polynomial of high order $|N| + m - 1$. A novel strategy developed in the NC solver to evaluate Equation (3) is to use Gaussian quadrature methods coupled with the log-sum-exp trick [7]. We have implemented both Gauss-Legendre and Gauss-Laguerre quadratures for Equation (3), finding them empirically better suited at evaluating normalizing constants than MATLAB's default integral method and overall the best evaluation methods unless job populations are asymptotically large. We point to Appendix B for a brief introduction to both Gauss-Legendre and Gauss-Laguerre quadratures.

Generally, Gauss-Laguerre quadrature enables increasingly precise evaluations of Equation (3) for growing values of its order K. However, it also faces numerical difficulties under a large population of jobs N, for which the quadrature weights and the integrand can display vastly different magnitudes. In such cases, we evaluate instead $\log G(m, N)$ in the quadrature summation by applying to the expression the log-sum-exp method using, e.g., with the implementation in Reference [7].

5.3.1 Numerical Example. We consider the same models considered for CoMoM and numerically evaluate the integral form (3). The integral method of MATLAB is run with an absolute tolerance of 10^{-12}. Node and weights for the Gaussian quadratures are pre-computed offline: Due to numerical instability, we can reach for the Gauss-Laguerre method up to order 300, while for Gauss-Legendre, we could pre-compute weights up to order 20,000 in the range $u \in [0, 10^6]$. We also include in the test the LE asymptotic expansion implemented in NC, which is a scalable method for models with few stations and many classes. The method applies a Laplace approximation to the simplex integral form for the normalizing constant in Reference [14]. Asymptotically, the LE results are tight to the exact solutions.

The results are given in Table 4. Errors are computed using the exact CoMoM results. Overall, we see that Gauss-Legendre is typically sufficient except in large asymptotic models where LE solutions are closer to optimal. The lower performance of Gauss-Laguerre is interpreted as being due to the restriction of using up to 300 nodes and weights in the interpolation due to numerical instability in their computation. Since Gauss-Legendre quadratures of order $K = 2n - 1$ are exact for polynomials of order up to n. Since the normalizing constant is the integral of a polynomial of order $n = |N| + m - 1$, it is possible to use the $K = 2|N| + 2m - 3$ order as a threshold for when the quadrature will cease to be exact and switch afterward to LE. For example, with Gaussian integration of order $n = 300$ and $m = 1$ the solution would switch to LE for $|N| \geq 6,000$. Note that, on top of this, approximation errors are incurred in Table 4 by the log-sum-exp trick used for numerical stabilization, which explains why small errors are incurred by Gauss-Legendre and Gauss-Laguerre also in cases where the quadrature should be exact. Another source of errors is that Gauss-Legendre requires a finite interval and has therefore been truncated to the range $u \in [0, 10^6]$, whereas the normalizing constant integral is defined in the range $u \in [0, \infty]$.

In summary, the numerical analysis reveals that a combination of Gauss-Legendre quadrature, for models with tens or hundreds of jobs, and LE, for larger models, provides an effective way to approximate homogeneous models.

Multiserver models. We remark that, contrary to CoMoM, the above technique is not immediate to extend to multiserver stations. An integral form for this case is available in Reference [38, Equation (A20)], yet it is a multiple integral and depends on a Bessel function with oscillatory behavior, which renders the numerical quadrature more difficult. Seidmann's approximation may then be preferred to apply Gauss-Legendre and Gauss-Laguerre quadratures to models with multiserver stations [48]. This simple method replaces a c-server station with demands D_r with a sub-network consisting of a single server queueing station with demands D_r/c and an infinite server station having demands $D_r(c-1)/c$. Under this transformation, as in the original system, a class-r job can traverse the two stations in D_r time overall when these are found both empty upon arrival. Moreover, the new infinite server station delay can be exactly aggregated within the pre-existing infinite server think time so the model retains overall the same number of stations. This approximation style may be also instantiated with the multiserver CoMoM method of Section 5.1.

5.4 Non-iterative MVA Approximation

Last, we leverage the ability of our solver to encompass multiple solution paradigms to develop a fast, non-iterative, MVA-based approximation scheme for multiclass models. As mentioned in

Section 4.2, non-iterative methods can be used, for example, to quickly initialize performance measures in submodels. We consider here single server stations and assume to approximate the multiserver case using Seidmann's approximation we have discussed before [48]. We also take product-form assumptions, but the expressions may be extended to the FCFS case using the standard approximate MVA FCFS approximation and following similar passages.

For each class r, the standard MVA expression for the throughput in that class is

$$X_r(N) = \frac{N_r}{Z_r + mD_r + \sum_{s=1}^{R} D_r(N_s - \delta_{rs} - X_s(N - 1_r)Z_s))}, \tag{4}$$

where $\delta_{sr} = 1$ if $s = r$ and 0 otherwise, and we have used that the total population of jobs in the queue is $\sum_{s=1}^{R}(N_s - \delta_{rs} - X_s(N-1_r)Z_s)$, since $X_s(N-1_r)Z_s$ represents by Little's law the number of jobs of class s residing at the delay node as seen at the instant of arrival at any of the queues in the homogeneous model.

To obtain a non-iterative formula, we use two approximations. First, we apply the interpolation [40, p. 649]

$$X_r(N - 1_r) \approx \frac{N_r - 1}{N_r} X_r(N). \tag{5}$$

Next, we estimate the contributions of classes $s \neq r$ in Equation (4) using twice the asymptotic upper bounds for the throughput [23, 34], obtaining

$$\sum_{\substack{s=1 \\ s\neq r}}^{R} D_r X_s(N-1_r)Z_s \approx B_r \quad B_r = \sum_{\substack{s=1 \\ s\neq r}}^{R} D_r \frac{N_s Z_s}{Z_s + D_s(m + |N| - 1 - \sum_{c=1}^{R} Z_c(N_c - \delta_{cr})/(Z_c + D_c(m + |N| - 2)))}. \tag{6}$$

Plugging Equations (5) and (6) into (4) and re-arranging terms, these two approximations together make Equation (4) a second-order univariate polynomial in the variable $X_r(N)$. This can therefore be solved explicitly, yielding the final expression

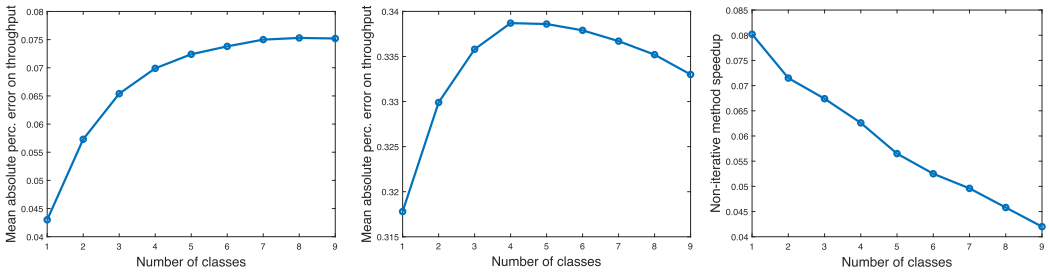
$$X_r(N) \approx \frac{Z_r - B_r + D_r|N| - \sqrt{B_r^2 + D_r^2|N|^2 + 2D_r|N|Z_r + Z_r^2 - 2B_rD_r|N| - 2B_rZ_r - 4N_rD_rZ_r}}{2D_rZ_r}. \tag{7}$$

The last expression can be computed in exactly R steps, one for each class r, thus with $O(1)$ complexity with respect to the total population size $|N|$.

The mean queue-length at any of the m replicas is then simply $Q_r(N) = m^{-1}(N_r - Z_rX_r(N))$, since the population not resident at the delay will be, on average, equally distributed among the m queueing stations. Using X_r and Q_r for all classes, it is immediate from Little's law to obtain response times and utilization values.

5.4.1 Numerical Example. To illustrate the accuracy and speed of the proposed method, we consider the approximation of random closed product-form models for a varying number of classes R and jobs N. We set $m = 1$ station, $N = \lceil 1 + \beta 50 \rceil$, where β is a vector of non-negative uniform random numbers summing to 1.0, the demands D_r are chosen uniformly at random with mean 1.0, and the think times are also randomized as $Z_r = 70 + 5R\log_{10} RU$, where U is a uniform random number in $[0, 1]$. With this parameterization, we generate 10,000 models for each value of $R \in [2, 10]$ and compare Equation (6) to the solution of the AQL approximate MVA algorithm, which is highly accurate [56].

Figure 3(a) illustrates the approximation error of (7) as the number of classes increases. The average utilization across all the models generated by the particular choice of Z_r is such that it remains fairly constant as R increases, falling in the narrow range $[0.72, 0.80]$. The error grows



(a) Non-iterative approximation (b) Asymptotic bound analysis (c) Runtime ratio compared to AQL

Fig. 3. Mean absolute percentage error for X_r on 10,000 for (a) the non-iterative approximation (7) compared to (b) an asymptotic upper bound [23]. Figure (c) shows the ratio of runtime of (7) compared to AQL [55], showing that the non-iterative approximation is much faster and the gap grows with more classes.

slightly from 4.35% for $R = 2$ classes to about 7.51% with $R = 10$ classes, confirming the good accuracy of the non-iterative approximation in (7) in light of its negligible computational cost. Figure 3(b) compares in the same experiment the performance of a standard asymptotic upper bound for the throughput [23], showing that the error is much larger than with (7), about 33%, on average.

Figure 3(c) shows the ratio between the execution time of AQL and the execution time of the proposed non-iterative method. As shown, the computational gains grow roughly linearly with the number of classes. In our test, (7), on average, took about 0.35ms to compute with $R = 10$ classes.

6 EXPERIMENTS AND CASE STUDIES

The developments presented in the previous sections focus on improving the solution of a single layer and meta-solver methods. In this section, we first assess the computational gains arising from the meta-solver approach on a set of random instances evaluated for mean performance metrics. Then, we illustrate further advantages of the meta-solver approach in supporting advanced analyses that are difficult to handle in the MVA paradigm. Last, we provide a case study to demonstrate the extensibility of the meta-solver approach, which allows us to embed with solvers for other stochastic formalisms in LQN analysis, as illustrated by means of integrating caching and queueing analyses.

6.1 Experiments on Random Models

To evaluate the performance of LN, we generate a set of 30 random LQNs featuring a varying number of tasks, processors, layers, and synchronous calls. The detailed parameters of these models are listed in Section A of the online supplement. In these models, we also assign random values to demands, multiplicities, and call means. Each LQN is solved with LN using the methods shown in Table 5.

A comparison of the performance of different solution methods in this experiment is then shown in Figure 4. The results highlight the mean, median, and maximum absolute error of the system throughput, as seen at the entry of the reference task that models the client workflow. The figure reveals that the heuristic proposed in Section 6 (*ln.auto*) outperforms all other LN solution methods, despite being a combination thereof. Specifically, *ln.auto* achieves a mean, median, and max error on the throughput seen by the client of 0.032, 0.022, and 0.171. We have observed that during the execution of *ln.auto* on the random LQNs, the heuristic chooses MVA about 9.6% times and NC the

Table 5. Solution Methods for Solving LQN Models

Solution method	*Description*
ln.mva	LN with approximate MVA in each layer.
ln.nc.comom	LN with the homogeneous CoMoM algorithm of Section 5.1 in each layer.
ln.nc.gleint	LN with the Gauss-Legendre integration of Section 5.3 in each layer.
ln.fluid	LN with the mean field approximation from Reference [41] in each layer.
ln.auto	LN with the automated solver selection heuristic of Section 4.2 in each layer.

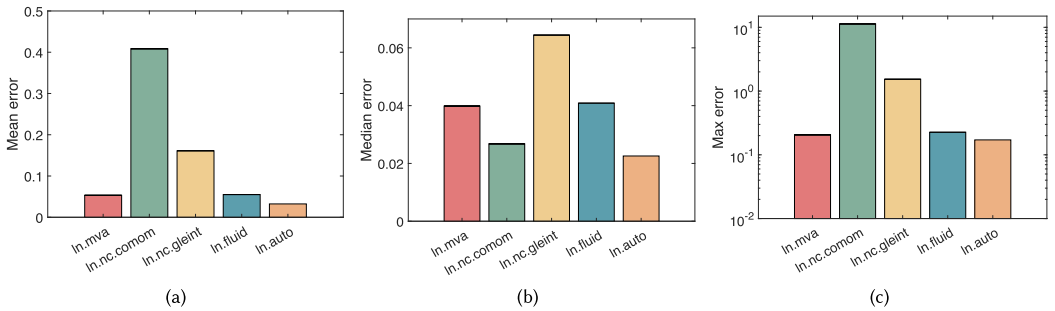


Fig. 4. Performance of different solution methods in analyzing LQN models: (a) mean error, (b) median error, (c) max error.

remaining times. Yet, it is interesting to note that, while CoMoM and the Gauss-Legendre methods often produce indistinguishable results, small deviations can still produce noticeable differences when looking at the iterative solution of an LQN. Yet, both methods appear to be useful, since Gauss-Legendre is better in mean while CoMoM is better in median.

The Fluid, NC, and MVA algorithms, albeit being in general very accurate, are less effective when used in isolation than when combined using the *ln.auto* method. This gives an indication of the gains resulting from using a meta-solver. We explain this due to accuracy variations in the solvers on submodels that require approximations falling outside product-form assumptions. The most common instance is an FCFS model with different (or non-exponential) service times, which is handled in the MVA solver using standard high-variability MVA approximations [43]. This case is instead handled differently in the NC and Fluid solvers leveraging the decay-rate interpolation presented in Reference [15]. The latter is a heuristic that replaces FCFS queues with PS queues where demands are identical across the classes. The demands values are assigned by means of a non-iterative heavy-load diffusion approximation of the original FCFS queueing system [8, Section10.1]. The resulting model is then tractable using product-form solution algorithms. In practice, models exist where each of the three solvers is more accurate than the others, leading to a benefit in using a meta-solver.

We also remark that the Fluid solver performs better in this experiment than in Table, 1 as the error metric here looks only at throughput accuracy without weighting in execution speed. Overall, these findings indicate that the proposed meta-solver heuristic of Section 4.2 can be useful for solving LQN models with heterogeneous submodels.

6.2 Case Study: Multi-solver-based Layer Analysis

Once the LN iteration has reached a fixed-point for the input parameters of the submodels, multiple paradigms can be applied to obtain the metrics of interest. We here focus on the latter case. We illustrate this feature in the example shown in Figure 1, which describes a scenario where two

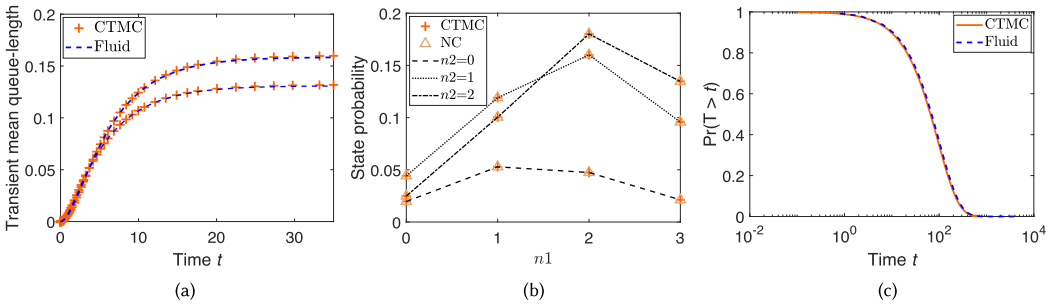


Fig. 5. Leveraging LINE meta-solver capabilities in analyzing LQN models.

job classes T1 and T2 require services from a server T3. Let S be a scale factor; in the model, we assume $3S$ and $2S$ jobs for *class*1 and *class*2, respectively, and T3 is an FCFS service station with multiplicity $c = 2S$.

We consider transient analysis, for which LINE provides multiple solver options. With the Fluid solver, one can compute transient metrics and passage time distributions in each layer. This solver performs mean-field approximations for PS nodes based on the theory presented in Reference [41]. As mentioned, FCFS stations are also treated as PS nodes with service demands corrected through a decay-rate interpolation [15]. Figure 5(a) shows the transient average queue length when the scale factor is set to $S = 1$ for the two job classes, representing T1 and T2 as clients, for the layer where T3 is modeled as a server. The plot shows a tight matching between the figures given by both CTMC and Fluid solvers. We can observe that the system reaches steady-state at around $t = 16$. In the figure, the Fluid solver looks very accurate for this non-saturated single queue scenario. Generally, trends of CTMC and Fluid tend to be close under different parameterizations, but the two curves may not overlap at different loads and with varying number of servers.

The NC and CTMC paradigms allow the computation of joint and marginal state probabilities in each layer. This makes it possible to obtain probabilistic measures, which may be useful in parameter inference and buffer overflow analysis. Figure 5(b) displays the joint steady-state probabilities calculated by NC and CTMC solvers for the whole state space at T3, given the service demand of A3 as 50 and again a scale factor of $S = 1$. The results from both solvers are almost the same, but the speed of NC solver is much faster. In this example, the calculation of each probability takes CTMC solver around 4 seconds, while the NC solver takes less than 50 milliseconds.

It is also notable that, despite in this example the queue is multiserver FCFS with heterogeneous service times, the $M/G/k$-diffusion approximation in NC allows us to compute the state probabilities very accurately, even if these do not admit a product-form solution. We have also attempted to scale up the same analysis to several tens of jobs, however, the CTMC becomes infeasible. In these scenarios, the simulation cost becomes orders of magnitude slower than NC.

Beyond joint probabilities, LN allows us to compute response time percentiles at each layer using the Fluid and CTMC solvers. In the model, the operations of T3 are executed by the processor P3; here, we use both solvers the obtain the response time distribution at the layer where T3 acts as a client to P3. Results for $S = 1$ are shown in Figure 5(c), demonstrating agreement of the solutions. With $S = 200$ the infinitesimal generator matrix has an order of tens of thousands and too large for the CTMC solver, which is unable to produce an answer. Conversely, the Fluid solver has a cost invariant with the scale S and can solve the same model in less than a second, both for the cases $S = 200$ and $S = 1$.

We have repeated the last analysis only on the layer where T3 acts as a server. Since T3 is a multiclass FCFS station, the state space grows much faster than before for increasing values of S.

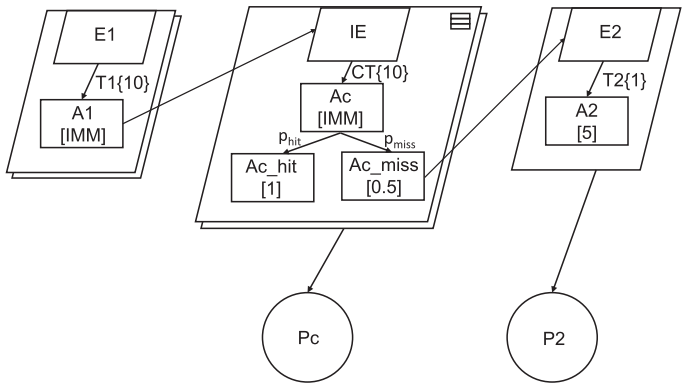


Fig. 6. Example of a multi-formalism model containing a caching layer (middle task).

For example, with $S = 50$ the CTMC is already intractable. Yet, the Fluid solver remains scalable, taking about 2 seconds to solve the model for $S = 50$. The increased time compared to the previous experiment is due to the additional iterations required by the $M/G/k$-diffusion approximation. On this layer, it is also possible to use the CoMoM algorithm for multiserver stations presented in Section 5.2, which is exact and produces visually identical results to the Fluid solver. We find that NC solver based on CoMoM is more scalable than the Fluid solver, taking about 1.4 seconds with $S = 50$, using the response time distribution analysis method proposed in Reference [38]. If one excludes the cost of computing the load-dependent scaling factors used in the technique proposed in Reference [38], then only 88 milliseconds are actually required by CoMoM to solve the model.

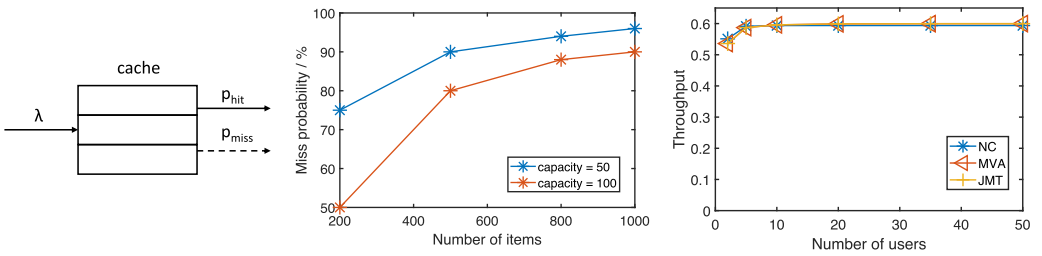
Recently, we have also demonstrated the possibility of using the queueing network solution **mixture density networks (MDNs)** for response time distribution analysis [39]. The MDN-based approach considerably increases the precision of computing percentiles for LQN response times compared to analytical approximations, which are notoriously difficult for multiclass networks.

6.3 Case Study: Multi-formalism Capabilities

LINE can analyze, analytically or via simulation, models with integrated queueing and caching formalisms. That is, the tool extends the LQN formalism, enabling the inclusion of *cache* nodes. When visiting a cache, a job reads an item according to some probability, resulting in either a cache hit or a cache miss. The subsequent processing activities can depend on whether the read outcome was a hit or miss. Cache reads also activate a replacement policy (e.g., random replacement, FIFO, LRU) to evict infrequently used items. The LN solver features the ability to define caches in an LQN using specialized tasks and entries named CacheTask and ItemEntry, which capture the data access requirements of jobs traversing the LQN. More details can be found in Reference [26].

LN can therefore also solve LQNs with caching, such as the three-layer model in Figure 6. The host processors P1, Pc, P2 adopt the processor-sharing scheduling policy. The number of users is represented by the multiplicity of task T1 and the number of jobs is represented by the multiplicity of task T2 and the cache task CT. A cache task is a novel LQN element introduced by LN to describe data item reads from a cache, with different activities occurring based on whether a cache hit or a cache miss occurred. This is modeled using state-dependent class-switching.

In the example under study, item access probabilities obey a discrete uniform distribution and the cache is configured with a **random replacement (RR)** strategy. Arbitrary access probability distributions may be configured in LN and replacement strategies such as FIFO or LRU are supported. Jobs requested from T1 retrieve the items in the cache task CT. If the required items are



(a) Cache upper submodel (b) Miss probability analysis (c) MVA and NC solutions

Fig. 7. Capabilities of LINE in analyzing multi-formalism LQN models.

cached, then jobs will be processed by the *hit* activity with a probability of p_{hit}. Otherwise, jobs will be transformed to the *miss* activity with a probability of p_{miss} and be further processed by task T2.

To solve an LQN model containing caching, LN first decomposes the entire model into a group of layers, as illustrated in Section 3.3. For the layer without cache nodes, the solutions are given in Section 5. However, for the layer that involves a cache node, LN additionally divides the layer into two submodels. In the upper submodel, the cache node is isolated in an open model with Poisson arrivals, as shown in Figure 7(a). In the lower submodel, the delay and the queueing station are contained in a closed queueing network with routing probabilities dynamically obtained from the p_{hit} and p_{miss} values obtained at the last iteration on the upper submodel. More details can be found in Reference [26].

For this model that combines both the queueing and caching stochastic formalisms, numerical results given by LN are shown in Figure 7. Figure 7(b) demonstrates the miss probabilities against different numbers of items, which decrease with the improvement of the cache capacity. Figure 7(c) compares the throughput accuracy for cache solved by MVA and NC solver, respectively. MVA analyzes caches by the **fixed-point iteration method (FPI)** proposed in Reference [17], whereas NC implements the normalizing constant asymptotics proposed in the same paper. The solvers are both compared against an equivalent model constructed with a JMT model using both queueing and Petri net formalisms, similar to the validation model used in Reference [26], but adapted to the example at hand. The results indicate high accuracy of both MVA and NC in capturing the cache layer throughput, which is generally a function of the cache hit ratio.

7 CONCLUSION

We have presented a novel meta-solver framework to evaluate LQN models, discussing a concrete implementation in the LN solver, part of the LINE software package. The article argues that it is beneficial to consider multiple queueing network solution paradigms at once, to leverage their distinctive strengths. We have illustrated the concept also by introducing new analysis methods for homogeneous queueing networks, such as those obtained through LQN loose layering, in particular, Gaussian integrals, a specialized and numerically stable version of the exact CoMoM algorithm [12], and a non-iterative mean-value analysis approximation. These new algorithms can efficiently analyze LQN layers in milliseconds or less. Case studies have shown, with a concrete tool implementation, the ability of the methodology to combine several formalisms and solution methods within LQNs. Future work may focus on developing solver-independent methods for LQN interlock correction and learning-based techniques to automate solver selection. The latter development could, in particular, avoid the adoption of heuristic rules, such as the ones listed in Table 1, which may otherwise require updating every time a new solution method is added to the solvers.

APPENDICES

A PROOF OF THEOREM 5.1

For a homogeneous model, the CoMoM recurrence relation may be written as $\boldsymbol{A}\Lambda(N) = \boldsymbol{B}\Lambda(N - 1_R)$, where

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}_{1,1} & \boldsymbol{A}_{1,2} \\ \boldsymbol{0} & \boldsymbol{A}_{2,2} \end{bmatrix} \quad \boldsymbol{B} = \begin{bmatrix} \boldsymbol{B}_{1,1} & \boldsymbol{0} \\ \boldsymbol{B}_{2,1} & \boldsymbol{B}_{2,2} \end{bmatrix}.$$

Let $\boldsymbol{0}_{I,J}$ indicate a block of zeros of size $I \times J$. Defining $\tilde{D} = (D_1, \ldots, D_R)^T$, we have

$$\boldsymbol{A}_{1,1} = \begin{bmatrix} 1 & -\tilde{D}^T \\ \boldsymbol{0}_{R-1,1} & -m\,\mathrm{diag}(\tilde{D}) \end{bmatrix} \quad \boldsymbol{A}_{1,2} = \begin{bmatrix} -1 & \boldsymbol{0}_{1,R-1} \\ \tilde{N} & -\mathrm{diag}(\tilde{Z}) \end{bmatrix} \quad \boldsymbol{A}_{2,2} = N_R\boldsymbol{I}$$

$$\boldsymbol{B}_{1,1} = D_R\boldsymbol{E}_{1,1} \quad \boldsymbol{B}_{2,1} = mD_R\boldsymbol{I} \quad \boldsymbol{B}_{2,2} = Z_R\boldsymbol{I}.$$

The inverse of the block upper triagonal matrix A is now computed as

$$\boldsymbol{A}^{-1} = \begin{bmatrix} \boldsymbol{A}_{1,1}^{-1} & \boldsymbol{S}\boldsymbol{A}_{2,2}^{-1} \\ \boldsymbol{0}_{R-1,1} & \boldsymbol{A}_{2,2}^{-1} \end{bmatrix},$$

with $\boldsymbol{S} = -\boldsymbol{A}_{1,1}^{-1}\boldsymbol{A}_{1,2}$. Observe first that

$$\boldsymbol{A}_{1,1}^{-1} = \begin{bmatrix} 1 & -m^{-1}e^T \\ \boldsymbol{0}_{R-1,1} & -m^{-1}\,\mathrm{diag}(\tilde{D})^{-1} \end{bmatrix} \quad \boldsymbol{A}_{2,2}^{-1} = N_R^{-1}\boldsymbol{I},$$

where $e^T = \tilde{D}^T\,\mathrm{diag}(\tilde{D})^{-1} = (1, \ldots, 1)$. Thus,

$$\boldsymbol{S} = -\begin{bmatrix} 1 & -m^{-1}e^T \\ \boldsymbol{0}_{R-1,1} & -m^{-1}\,\mathrm{diag}(\tilde{D})^{-1} \end{bmatrix} \begin{bmatrix} -1 & \boldsymbol{0}_{1,R-1} \\ \tilde{N} & -\mathrm{diag}(\tilde{Z}) \end{bmatrix} = -m^{-1}\begin{bmatrix} -|\tilde{N}| - m & \tilde{Z}^T \\ -\mathrm{diag}(\tilde{D})^{-1}\tilde{N} & \mathrm{diag}(\tilde{D})^{-1}\,\mathrm{diag}(\tilde{Z}) \end{bmatrix}.$$

Note that $\boldsymbol{A}_{2,2}$ is the only block that depends on N_R. We can therefore write

$$\boldsymbol{A}^{-1}\boldsymbol{B} = \boldsymbol{F}_{1,R} + N_R^{-1}\boldsymbol{F}_{2,R},$$

where

$$\boldsymbol{F}_{1,R} = \begin{bmatrix} \boldsymbol{A}_{1,1}^{-1} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} D_R\boldsymbol{E}_{1,1} & \boldsymbol{0} \\ mD_R\boldsymbol{I} & Z_R\boldsymbol{I} \end{bmatrix} = \begin{bmatrix} D_R\boldsymbol{E}_{1,1} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix},$$

$$\boldsymbol{F}_{2,R} = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{S} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} D_R\boldsymbol{E}_{1,1} & \boldsymbol{0} \\ mD_R\boldsymbol{I} & Z_R\boldsymbol{I} \end{bmatrix} = \begin{bmatrix} mD_R\boldsymbol{S} & Z_R\boldsymbol{S} \\ mD_R\boldsymbol{I} & Z_R\boldsymbol{I} \end{bmatrix}.$$

B GAUSSIAN QUADRATURES

A Gauss-Laguerre quadrature of order K evaluates exponentially weighted integrals using the approximation

$$\int_{x=0}^{\infty} e^{-x} f(x)dx \approx \sum_{k=1}^{K} w_k f(x_k), \tag{8}$$

where x_k denotes the kth root of the Laguerre polynomial $L_K(x) = \sum_{j=0}^{K} \binom{K}{i} \frac{(-1)^j}{j!} x^j$, and with weights $w_k = x_k \left((k+1)^2 \left[L_{k+1}(x_k)\right]^2\right)^{-1}$. Gauss-Legendre methods are similar but applicable to finite ranges $[a, b]$. Setting $a = 0$ and large b, they can also help evaluate the normalizing constant. Their main benefit is that nodes and weights do not incur the same floating-point range exceptions as observed instead for Gauss-Laguerre quadratures of large order. We point to Reference [32] for further details on Gauss-Legendre methods.

C PROOF OF THEOREM 5.3

For a homogeneous model with m identical single server stations, the McKenna-Mitra integral takes the form

$$G(m,N) = \frac{1}{\prod_{r=1}^{R} N_r!} \int_{u_1=0}^{+\infty} \cdots \int_{u_m=0}^{+\infty} e^{-(u_1+\ldots+u_m)} h(u_1+\ldots+u_m) du_1 \cdots du_m,$$

where $h(u) = \prod_{r=1}^{R} (Z_r + D_r u)^{N_r}$. We note that the multidimensional integral may be interpreted as computing $E[h(U_1 + \ldots + U_m)]$ for the i.i.d. exponential random variables $U_i \sim Exp(1)$. The result then readily follows after noting that $U_1 + \ldots + U_m$ is Erlang-m distributed with density $f(u) = \frac{1}{(m-1)!} u^{m-1} e^{-u}$.

D PROOF OF THEOREM 5.2

Initially, consider a load-independent model with service rate $\mu(n) = 1$. Let the entries of $\Pi(N)$ be indicated with $\widetilde{\pi}_N(n)$, $n = |N|, \ldots, 0$. A probabilistic population constraint holds for homogeneous models with $m = 1$ [12, Thm. 6]

$$N_R \widetilde{\pi}_N(n) = Z_R \widetilde{\pi}_{N-1_R}(n) + n D_R \widetilde{\pi}_{N-1_R}(n-1) \tag{9}$$

for all $n = 1, \ldots, |N|$ and where $\widetilde{\pi}_{N-1_R}(n-1) = 0$ if $n = 0$. With a load-dependent queueing station ($m = 1$), we generalize here the derivation in Reference [12] with similar passages to the following form:

$$N_R \widetilde{\pi}_N(n) = Z_R \widetilde{\pi}_{N-1_R}(n) + n \frac{D_R}{\mu(n)} \widetilde{\pi}_{N-1_R}(n-1), \tag{10}$$

where $\mu(n)$ is the load-dependent scaling at the queueing station. Note that the service rate $\mu(n)$ is the same, irrespective of the number of classes R and the current class r, due to the structure of the product-form solution. Then, leveraging the flow-equivalent server theorem [21], it can be computed as the throughput of an equivalent single-class model for the sub-network consisting only of the queueing stations. Organizing Equation (10) in matrix form, we get Equation (2).

REFERENCES

[1] [n.d.]. Annotated Bibliography of Papers Related to Layered Queueing. Retrieved from http://www.sce.carleton.ca/rads/lqns/papers/. Accessed 2 December 2023.

[2] Elvio G. Amparore, Gianfranco Balbo, Marco Beccuti, Susanna Donatelli, and Giuliana Franceschinis. 2016. *30 Years of GreatSPN*. Springer, 227–254.

[3] Angelos Aveklouris, Maria Vlasiou, and Bert Zwart. 2019. Bounds and limit theorems for a layered queueing model in electric vehicle charging. *Queue. Syst. Theor. Appl.* 93, 1-2 (2019), 83–137. DOI: https://doi.org/10.1007/s11134-019-09616-z

[4] Forest Baskett, K. Mani Chandy, Richard R. Muntz, and Fernando G. Palacios. 1975. Open, closed, and mixed networks of queues with different classes of customers. *J. ACM* 22, 2 (1975), 248–260.

[5] Marco Bertoli, Giuliano Casale, and Giuseppe Serazzi. 2007. The JMT simulator for performance evaluation of non-product-form queueing networks. In *ANSS*. IEEE, 3–10.

[6] Dario Bini, Beatrice Meini, Sergio Steffé, Juan F. Pérez, and Benny Van Houdt. 2012. SMCSolver and Q-MAM: Tools for matrix-analytic methods. *ACM SIGMETRICS Perform. Eval. Rev.* 39, 4 (2012), 46–46.

[7] Pierre Blanchard, Desmond J. Higham, and Nicholas J. Higham. 2021. Accurately computing the log-sum-exp and softmax functions. *IMA J. Numer. Anal.* 41, 4 (2021), 2311–2330.

[8] Gunter Bolch, Stefan Greiner, Hermann de Meer, and Kishor S. Trivedi. 2006. *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*. John Wiley & Sons.

[9] Luca Bortolussi, Jane Hillston, Diego Latella, and Mieke Massink. 2013. Continuous approximation of collective system behaviour: A tutorial. *Perform. Eval.* 70, 5 (2013), 317–349.

[10] Steven C. Bruell, Gianfranco Balbo, and P. V. Afshari. 1984. Mean value analysis of mixed, multiple class BCMP networks with load dependent service stations. *Perform. Eval.* 4, 4 (1984), 241–260.

[11] Jeffrey P. Buzen. 1973. Computational algorithms for closed queueing networks with exponential servers. *Commun. ACM* 16, 9 (1973), 527–531.

[12] Giuliano Casale. 2009. CoMoM: Efficient class-oriented evaluation of multiclass performance models. *IEEE Trans. Softw. Eng.* 35, 2 (2009), 162–177.
[13] Giuliano Casale. 2011. Exact analysis of performance models by the method of moments. *Perform. Eval.* 68, 6 (2011), 487–506.
[14] Giuliano Casale. 2017. Accelerating performance inference over closed systems by asymptotic methods. In *ACM SIGMETRICS*. ACM, 8:1–8:25.
[15] Giuliano Casale. 2020. Integrated performance evaluation of extended queueing network models with LINE. In *WSC*. IEEE, 2377–2388.
[16] Giuliano Casale, Yicheng Gao, Zifeng Niu, and Lulai Zhu. 2022. LN: A meta-solver for layered queueing network analysis. In *QEST (Lecture Notes in Computer Science,* Vol. 13479*)*, Erika Ábrahám and Marco Paolieri (Eds.). Springer, 232–254. DOI: https://doi.org/10.1007/978-3-031-16336-4_12
[17] Giuliano Casale and Nicolas Gast. 2021. Performance analysis methods for list-based caches with non-uniform access. *IEEE/ACM Trans. Netw.* 29, 2 (2021), 651–664.
[18] Giuliano Casale, Peter G. Harrison, and Wai Hong Ong. 2021. Facilitating load-dependent queueing analysis through factorization. *Perform. Eval.* 152 (2021), 102241.
[19] Giuliano Casale, Richard R. Muntz, and Giuseppe Serazzi. 2008. Geometric bounds: A noniterative analysis technique for closed queueing networks. *IEEE Trans. Comput.* 57, 6 (2008), 780–794.
[20] Giuliano Casale, Juan F. Pérez, and Weikun Wang. 2015. QD-AMVA: Evaluating systems with queue-dependent service requirements. *Perform. Eval.* 91 (2015), 80–98.
[21] K. Mani Chandy, Ulrich Herzog, and Lin Woo. 1975. Parametric analysis of queuing networks. *IBM J. Res. Devel.* 19, 1 (1975), 36–42.
[22] K. Mani Chandy and Doug Neuse. 1982. Linearizer: A heuristic algorithm for queueing network models of computing systems. *Commun. ACM* 25, 2 (1982), 126–134.
[23] Peter J. Denning and Jeffrey P. Buzen. 1978. The operational analysis of queueing network models. *ACM Comput. Surv.* 10, 3 (1978), 225–261.
[24] Gregory Franks. 2000. *Performance Analysis of Distributed Server Systems*. Ph.D. Dissertation. Carleton University.
[25] Greg Franks, Tariq Al-Omari, Murray Woodside, Olivia Das, and Salem Derisavi. 2009. Enhanced modeling and solution of layered queueing networks. *IEEE Trans. Softw. Eng.* 35, 2 (2009), 148–161.
[26] Yicheng Gao and Giuliano Casale. 2022. JCSP: Joint caching and service placement for edge computing systems. In *IEEE/ACM IWQoS*. IEEE.
[27] Nicolas Gast and Benny Van Houdt. 2017. A refined mean field approximation. *Proc. ACM Measur. Anal. Comput. Syst.* 1, 2 (2017), 1–28.
[28] Neil J. Gunther. 2011. *Analyzing Computer System Performance with Perl::PDQ*. Springer.
[29] Gábor Horváth and Miklós Telek. 2017. BuTools 2: A rich toolbox for Markovian performance evaluation. In *VALUETOOLS*. ICST, 137–142.
[30] Patricia A. Jacobson and Edward D. Lazowska. 1982. Analyzing queueing networks with simultaneous resource possession. *Commun. ACM* 25, 2 (1982), 142–151.
[31] Jing Jia and Sunderesh S. Heragu. 2009. Solving semi-open queuing networks. *Oper. Res.* 57, 2 (2009), 391–401.
[32] Fredrik Johansson and Marc Mezzarobba. 2018. Fast and rigorous arbitrary-precision computation of Gauss-Legendre quadrature nodes and weights. *SIAM J. Sci. Comput.* 40, 6 (2018), C726–C747.
[33] Simon S. Lam. 1982. Dynamic scaling and growth behavior of queueing network normalization constants. *J. ACM* 29, 2 (1982), 492–513.
[34] Edward D. Lazowska, John Zahorjan, Gordon S. Graham, and Kenneth C. Sevcik. 1984. *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*. Prentice Hall.
[35] Moreno Marzolla. 2014. The octave queueing package. In *QEST*. Springer, 174–177.
[36] James McKenna and Debasis Mitra. 1984. Asymptotic expansions and integral representations of moments of queue lengths in closed Markovian networks. *J. ACM* 31, 2 (1984), 346–360.
[37] Carl D. Meyer. 1989. Stochastic complementation, uncoupling Markov chains, and the theory of nearly reducible systems. *SIAM Rev.* 31, 2 (1989), 240–272.
[38] Debasis Mitra and James McKenna. 1986. Asymptotic expansions for closed Markovian networks with state-dependent service rates. *J. ACM* 33, 3 (1986), 568–592.
[39] Zifeng Niu and Giuliano Casale. 2021. A mixture density network approach to predicting response times in layered systems. In *MASCOTS*. IEEE, 1–8.
[40] Krishna R. Pattipati, Michael M. Kostreva, and John L. Teele. 1990. Approximate mean value analysis algorithms for queuing networks: Existence, uniqueness, and convergence results. *J. ACM* 37, 3 (1990), 643–673.
[41] Juan F. Pérez and Giuliano Casale. 2017. LINE: Evaluating software applications in unreliable environments. *IEEE Trans. Reliab.* 66, 3 (2017), 837–853.

[42] Francesca Randone, Luca Bortolussi, and Mirco Tribastone. 2022. Jump longer to jump less: Improving dynamic boundary projection with h-Scaling. In *Quantitative Evaluation of Systems*, Erika Ábrahám and Marco Paolieri (Eds.). Springer International Publishing, Cham, 150–170.
[43] Martin Reiser. 1979. A queueing network analysis of computer communication networks with window flow control. *IEEE Trans. Commun.* 27, 8 (1979), 1199–1209.
[44] Martin Reiser and Stephen S. Lavenberg. 1980. Mean-value analysis of closed multichain queuing networks. *J. ACM* 27, 2 (1980), 313–322.
[45] Jerry Rolia, Giuliano Casale, Diwakar Krishnamurthy, Stephen Dawson, and Stephan Kraft. 2009. Predictive modelling of SAP ERP applications: Challenges and solutions. In *VALUETOOLS*. 1–9.
[46] Jerome A. Rolia and Kenneth C. Sevcik. 1995. The method of layers. *IEEE Trans. Softw. Eng.* 21, 8 (1995), 689–700.
[47] Johan Ruuskanen, Tommi Berner, Karl-Erik Årzén, and Anton Cervin. 2021. Improving the mean-field fluid model of processor sharing queueing networks for dynamic performance models in cloud computing. *Perform. Eval.* 151 (2021), 102231.
[48] Abraham Seidmann, Paul J. Schweitzer, and Sarit Shalev-Oren. 1987. Computerized closed queueing network models of flexible manufacturing systems: A comparative evaluation. *Large Scale Syst.* 12 (1987), 91–107.
[49] Yasir Shoaib and Olivia Das. 2011. Web application performance modeling using layered queueing networks. *Electron. Notes Theor. Comput. Sci.* 275 (2011), 123–142. DOI: https://doi.org/10.1016/j.entcs.2011.09.009
[50] Mirco Tribastone. 2013. A fluid model for layered queueing networks. *IEEE Trans. Softw. Eng.* 39, 6 (2013), 744–756.
[51] Tabea Waizmann and Mirco Tribastone. 2016. DiffLQN: Differential equation analysis of layered queuing networks. In *ICPE*. ACM, 63–68.
[52] Weikun Wang and Giuliano Casale. 2013. Bayesian service demand estimation using Gibbs sampling. In *MASCOTS*. IEEE, 567–576.
[53] Murray Woodside. 2013. Tutorial Introduction to Layered Modeling of Software Performance. Retrieved from http://www.sce.carleton.ca/rads/lqns/lqn-documentation/tutorialh.pdf
[54] M. Woodside, John E. Neilson, Dorina C. Petriu, and Shikharesh Majumdar. 1995. The stochastic rendezvous network model for performance of synchronous client-server-like distributed software. *IEEE Trans. Comput.* 44, 1 (1995), 20–34.
[55] John Zahorjan. 1979. An exact solution method for the general class of closed separable queueing networks. *SIGMETRICS Perform. Eval. Rev.* 8, 3, 107–112.
[56] John Zahorjan, Derek L. Eager, and Hisham M. Sweillam. 1988. Accuracy, speed, and convergence of approximate mean value analysis. *Perform. Eval.* 8, 4 (1988), 255–270.
[57] Lulai Zhu, Giuliano Casale, and Iker Perez. 2020. Fluid approximation of closed queueing networks with discriminatory processor sharing. *Perform. Eval.* 139 (2020), 102094.

Received 7 March 2023; revised 17 August 2023; accepted 8 November 2023