Proceedings of the 2020 Winter Simulation Conference
K.-H. Bae, B. Feng, S. Kim, S. Lazarova-Molnar, Z. Zheng, T. Roeder, and R. Thiesing, eds.

INTEGRATED PERFORMANCE EVALUATION OF EXTENDED QUEUEING NETWORK MODELS WITH LINE

Giuliano Casale

Department of Computing
Imperial College London
SW7 2AZ, London, UK

ABSTRACT

Despite the large literature on queueing theory and its applications, tool support to analyze these models is mostly focused on discrete-event simulation and mean-value analysis (MVA). This circumstance diminishes the applicability of other types of advanced queueing analysis methods to practical engineering problems, for example analytical methods to extract probability measures useful in learning and inference. In this tool paper, we present LINE 2.0, an integrated software package to specify and analyze extended queueing network models. This new version of the tool is underpinned by an object-oriented language to declare a fairly broad class of extended queueing networks. These abstractions have been used to integrate in a coherent setting over 40 different simulation-based and analytical solution methods, facilitating their use in applications.

1 INTRODUCTION

Queueing network models have been extensively used in performance analysis and optimization of software applications, computer networks, call centers, logistics, and business processes, among others. Despite a rich literature, limited work has however been carried out towards offering modelling and simulation tools that integrate different analysis paradigms in a coherent way.

LINE (http://line-solver.sf.net/) is a MATLAB package to specify and analyze extended queueing network models. With its version 2.0, the tool now offers an object-oriented language to declare and analyze a fairly general class of queueing network models featuring class-switching, layering, state-dependent routing, non-exponential service, caching, and random environments. LINE exposes over 40 algorithms capturing general paradigms to analyze queueing networks, such as global balance equations, discrete-event simulation, mean-value analysis, matrix analytic methods, normalizing constant analysis, and mean-field/fluid approximations (Casale 2019). Thanks to the royalty-free MATLAB compiler runtime, LINE can also be used without a MATLAB license, for example as an external solver for the Palladio Simulator for software engineers (Pérez and Casale 2013; Heinrich *et al.* 2018).

The need for an integrated queueing modelling approach stems from the observation that modeling practitioners often opt for discrete-event simulation due to its generality and accuracy. For example, nearly all community questions over the last 15 years in the help forum of Java Modelling Tools (JMT) (Bertoli et al. 2006), a popular suite for queueing analysis, are centered around simulation models, despite the tool also offers analytical approximation techniques. Indeed, when it comes to replacing simulation with analytical approximations, for example to accelerate search-based decision making, it is difficult to find systematically accurate solution methods for general queueing networks. We argue that this calls for more emphasis within the modeling community on defining *integrated* frameworks that can consolidate research efforts on advanced stochastic methods in order to offer a range of alternative techniques for analysis within the same environment of simulation methods. Running algorithms within an integrated environment offers the possibility to programmatically check assumptions, recommend to the user the available analysis methods for a certain model, and inexpensively trial different algorithms against simulation to assess their accuracy, thus facilitating the adoption of advanced stochastic techniques in applied work.

The latest release of LINE 2.0 aligns to the above vision by evolving the original tool from a focus on mean-field/fluid approximations (Pérez and Casale 2017) into a general framework implementing a collection of solution paradigms. The package is also easy to extend and to interface through model-to-model transformations with external tools. For example, JMT (Bertoli et al. 2006) and LQNS (G. Franks et al. 2009) are already integrated within LINE. While our earlier work on LINE has developed the underpinning fluid analysis theory (Pérez and Casale 2017), this paper is the first one to present LINE 2.0 holistically as an integrated software tool. Compared to the 1.0 version, LINE 2.0 mainly adds an object-oriented modeling language and six new solvers (CTMC, JMT, MVA, MAM, NC, SSA) featuring tens of solutions methods. A short abstract documenting a preliminary demonstration of the tool has appeared in (Casale 2019).

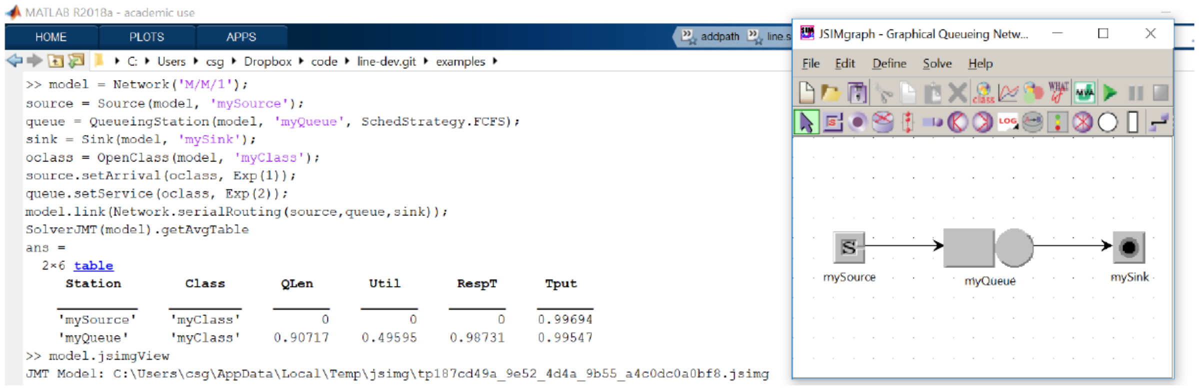
Other open source tools exist to analyze product-form and extended queueing networks with analytical techniques. JMVA (Bertoli et al. 2006), within the JMT suite, offers several mean-value and normalizing constant algorithms for product-form networks. The methods offered in LINE are a strict superset of those available in JMVA, adding methods for class-switching, priorities, state-dependent routing, among others. Octave Queueing (Marzolla 2014) offers multiple algorithms for regular and load-dependent product-form models, including exact and bounding techniques. A distinctive feature in Octave Queueing is the ability to deal with networks with blocking. As in the case of JMVA, Octave Queueing is not yet extended to a number of non-product-form features that are supported in LINE. PDQ (Gunther 2011) is another popular tool for product-form queueing network analysis, however it does not deal with extended queueing networks. A distinctive feature of this tool is the ability to compute the optimal load for a queueing network. This may be achieved using LINE by combining its solvers with native MATLAB optimization functions. LQNS (G. Franks et al. 2009) is a very popular tool for layered queueing networks. The tool provides advanced support for extended queueing networks based on approximate approximate mean-value analysis and a simulation tool (LQSIM). Compared to LQNS, LINE also supports layered networks. Although offering a less advanced implementation of mean-value analysis methods than LQNS, LINE features different analysis techniques, such as fluid approximations, probabilistic methods, and hybrid solutions where only a subset of the layers are simulated, e.g., to replay service or arrival times from a measured trace.

The rest of the paper is organized as follows. Section 2 presents the modelling language abstractions supported in LINE 2.0, followed in Section 3 by a definition of the core principles of its software architecture and solvers. Details about the algorithms developed in LINE for probabilistic analysis are illustrated in Section 4. Section 5 illustrates the tool on numerical examples. Finally, Section 6 gives conclusions. If not otherwise specified, we shall refer in the following to LINE 2.0 simply as LINE.

2 MODELLING LANGUAGE

LINE 2.0 features an object-oriented modelling language to declare a fairly general family of stochastic networks. Figure 1 illustrates the language syntax to define a M/M/1 queue and a visualization obtained via model-to-model transformation to JMT. Due to limited space, we point the reader to standard queueing theory definitions in (Bolch et al. 2006) and to the LINE manual for details on the language syntax, and focus the discussion in this section instead on features and design choices.

From a modeling standpoint, LINE emphasizes in particular the analysis of queueing networks with *class switching*, as opposed to standard multiclass models. Class-switching is important to model *workflows*, which often feature re-entrant lines where a job visits twice the same node spending time according to a different service distribution at each passage. Such behavior can be naturally modelled as a job changing class. The class may be equivalently seen as carrying the job state. Even though class-switching models are well-understood theoretically when it comes to their mean-value analysis (Reiser and Lavenberg 1980), there is a shortage of results for their probabilistic analysis, which are instead developed in LINE as we discuss in Section 4.


MATLAB R2018a - academic use
HOME
PLOTS
APPS
addpath
C: ▸ Users ▸ csg ▸ Dropbox ▸ code ▸ line-dev.git ▸ examples ▸
JSIMgraph - Graphical Queueing Netw...
File Edit Define Solve Help
mySource
myQueue
mySink

```
>> model = Network('M/M/1');
source = Source(model, 'mySource');
queue = QueueingStation(model, 'myQueue', SchedStrategy.FCFS);
sink = Sink(model, 'mySink');
oclass = OpenClass(model, 'myClass');
source.setArrival(oclass, Exp(1));
queue.setService(oclass, Exp(2));
model.link(Network.serialRouting(source,queue,sink));
SolverJMT(model).getAvgTable
ans =
  2×6 table
     Station        Class        QLen       Util      RespT      Tput
   ___________    _________    _______    _______    _______    _______

   'mySource'     'myClass'          0          0          0    0.99694
   'myQueue'      'myClass'    0.90717    0.49595    0.98731    0.99547
>> model.jsimgView
JMT Model: C:\Users\csg\AppData\Local\Temp\jsimg\tp187cd49a_9e52_4d4a_9b55_a4c0dc0a0bf8.jsimg
```

Figure 1: M/M/1 simulation in LINE and visualization via export to JMT

LINE models consist of networks of *nodes* visited by jobs. Upon visits nodes jobs may incur a response time, in which case the node is called a *station*. While most nodes in LINE belong to standard queueing theory, such as multi-server *queues*, infinite-server *delays*, or arrival/departure points to exchange jobs with the external world (*source/sink*), some node types supported in the package are non-standard, such as *caches* that are stateful nodes to perform class-switching, *logger* nodes that record traversal in log files during simulation, or *routers* that are topological branch points used only for dispatching purposes. Some of these custom node types are available also in other tools such as JMT.

Queueing network models in LINE may be analyzed in isolation, or grouped into collections, called *ensembles*, that specify a more complex stochastic system, such as a layered queueing network (LQN) or a system operating in a random environment. The analysis of ensembles has been already available in LINE 1.0 and iteratively combines the solutions of the individual sub-models, aggregating results using for LQNs the Method of Layers (MoL) with loose layering (Rolia and Sevcik 1995; Franks 1999) and for random environments the blending method (Casale et al. 2014). In what follows, we focus on describing the available techniques in LINE to analyze a single queueing network, pointing the reader to the LINE manual for more details about ensembles.

2.1 Nodes and routing

Nodes visited by jobs are explicitly classified within the solver into *stateless* and *stateful* nodes, depending on whether they have an associated state or not. Node classification may depend on the user parameterization, e.g., a *router* node may be stateful if instantiated with a round-robin routing policy, but stateless under a static probabilistic policy.

A rich set of processes, scheduling, and routing options are available in LINE to further qualify nodes. Service and arrival processes are assumed phase-type, but support exists to analyze in simulation non-Markovian distributions, a full list is available in the LINE manual. Moment-based and trace-based fitting are also offered, for example via moment-matching formulas for acyclic phase-type distributions (Bobbio, Horváth, and Telek 2005). In some solvers, it is also possible to analyze non-renewal arrivals by means of Markovian Arrival Processes (MAPs), which are also supported and fitted based on the KPC-Toolbox (Casale et al. 2008b).

Queueing stations can be instantiated with several *scheduling policies*, ranging from Processor Sharing (PS), Infinite Server (INF), and First-Come First-Served (FCFS) scheduling, to more advanced policies such as Discriminatory Processor Sharing (DPS), Generalized Processor Sharing (GPS), Shortest Job First (SJF), Shortest Expected Processing Time (SEPT), or Head-Of-Line priority (HOL).

Routing policies are typically based on a Markovian routing with class-switching, but support exists in the tool also for state-dependent routing such as Round-Robin or Join-the-Shortest-Queue (JSQ). Lastly, nodes are also allowed to feature a finite buffer and multiple servers.

2.2 Chains and classes

One of the most delicate aspects in class-switching models is to decide on the precise definition of classes and chains. In LINE, models are composed by nodes visited by jobs belonging to one of R job classes. Classes may be assigned to a priority group. Class-switching entails the existence of some positive probabilities $p_{i,j}^{r,s}(t)$, at time t since initialization, that a job joins node j in class r after last completing service at node i as a class s job. The set of reachable classes for a job is termed its *chain*. In LINE, we assume that a network has $C \leq R$ chains and that a class can belong to a single chain. Even though class-switching in LINE may arise as a result of state-dependent behavior, typical models involve user-defined constant class-switching probabilities $p_{i,j}^{r,s} \equiv p_{i,j}^{r,s}(t)$, $\forall t$, from which chains can be readily computed (Bolch et al. 2006, Section 7.3.6.1).

In standard terminology, a closed class defines a workload with a constant population, however this definition becomes inconsistent under class-switching given that the total population in a closed class may not be conserved. As such, LINE defines closed classes as those composed by jobs that cannot be routed out from a source node or to a sink node. Job conservation is in general applicable only to *chains* consisting of closed classes only.

Closed chains may also include transient classes, where jobs are initialized but that they may eventually leave to cycle within a subset of recurrent classes. Classes may have zero arrival rates or empty initial populations at start, as their population may still vary over time due to the class-switching mechanism.

2.2.1 *Computing chain properties*

For a network with state-independent routing probabilities $p_{i,j}^{r,s}$, it is simple to compute a binary matrix $B = [b_{r,s}]$, where $b_{r,s} = 1$ if there exist an (i, j) pair for which $p_{i,j}^{r,s} > 0$, or otherwise $b_{r,s} = 0$. The matrix $G = B + B^T$ can then be seen as an incidence matrix for an undirected graph that describes communicating classes, and from which we can identify chains by finding its connected components.

This analysis is however impeded by state-dependent routing, which complicates the analysis of the chains. For state-dependent policies, LINE requires every state-dependent routing policy to statically supply as part of the implementation a binary matrix, with entries consisting of pairs *(input class, target class)*, which declares the allowed class switchings across all possible routing states. Such matrix is sufficient to compute the matrix B also in the case of state-dependent routing.

2.2.2 *Performance metrics*

Performance metrics such as queue-lengths, response time, throughputs, and utilizations can be analyzed in transient or in steady-state, either in terms of probability distributions or their moments. However, the presence of chains affects the definition of some of these metrics as follows.

For example, for a job that moves through classes $A \rightarrow B \rightarrow C$, it is difficult to decide if the system (i.e., end-to-end) response time should be split across the classes or for instance attributed only just to the entry class A or the exit class C. In LINE, to avoid such ambiguities, the tool restricts the computation of system metrics to *chains*, while class-level breakdowns are available only for metrics at nodes. That is, the response time of a class-switching sequence $A \rightarrow B \rightarrow C$ is offered only as a metric for the chain $\{A, B, C\}$. Since closed chains conserve the job population, this definition also ensures that the metric satisfies the Little's Law (Bolch et al. 2006).

Note that for closed classes LINE declares a completion upon arrival of the job to a *reference station* associated to the chain. There is also the possibility to decide which classes within the chain can complete upon arrival to this station. This generalizes the standard theory by allowing a job to possibly perform a

Table 1: Solvers offered in LINE

Solver	*Description*	*Ensembles*
CTMC	Direct numerical analysis of Markov process	No
FLUID	Fluid/mean-field approximation	No
JMT	Ad-hoc discrete event simulation	No
MAM	Matrix analytic methods	No
MVA	Mean-value analysis	No
NC	Normalizing constant analysis	No
SSA	Stochastic simulation of Markov process	No
LN	Analysis of layered models	Yes
LQNS	Analysis of layered models with LQNS	Yes
ENV	Analysis of random environment	Yes

number of self-loops at its reference station prior to completing the visit, without having to count each loop as a separate completion.

2.3 State modeling

A conceptual problem in the context of state modeling is how to ease the definition of the initial state in a transient analysis, given the fact that some node types have fairly complex state vectors, as is the case for example of multiclass multiserver phase-type FCFS queues. To address this issue, LINE allow users to supply the initial state as an *aggregate* state, counting the jobs of different classes at the nodes, but not for example specifying their order in the buffer, which is still however required to uniquely identify the state of FCFS stations. To fill-in the missing information, the tool will then automatically generate all the possible initial states that match the aggregate counts supplied by the user and assign a *prior probability* to each of them. To cope with state space explosion, information may be supplied by the user on the number of jobs in service for each class, or about their current phase, or by upper bounding the number of possible initial states. LINE will then solve models for all possible initial states, returning to the user a set of performance indexes already weighted according to the prior probabilities of the initial states.

3 SOFTWARE ARCHITECTURE

3.1 Design for extensibility

One major design decision in LINE has been to facilitate extensions of the tool by third parties, either with native MATLAB code or through interfacing with external tools. The software architecture addresses this requirement with model-to-model (M2M) transformations. In particular, LINE models can be directly exported to the input formats of tools such as JMT (jsimg, jsimw, and jmva formats), LQNS (lqnx and srvn formats), and in the PMIF XML format. Such transformation are bi-directional, as LINE also allows to import models from such formats into MATLAB, and then to generate scripts that will automatically recreate those models in the LINE native language.

In addition to ease the interfacing of external tools with M2M transformations, one challenge has been to facilitate the integration of third party algorithms so that contributors do not need to learn the LINE internals to add new algorithms. To do so, a method called *getStruct()* is available for each model that generates a *static* data structure with summary information about the network, which is trivial to parse by third party algorithms. During this process, auxiliary quantities such as visit ratios and chain analysis are calculated, as discussed later in Section 4. This data structure is fairly detailed, as it includes over 40 fields, but it is generated only once and then efficiently updated using ad-hoc library functions. Such functions can help in reducing the cost of solving a sequence of models in optimization-based search.

3.2 LINE Solvers

Solvers in LINE do not identify a single algorithm, rather they represent a family of solution methods aimed at computing performance indexes. The solvers available in version 2.0 are shown in Table 1. Compared to earlier versions, all solvers are novel except the FLUID solver, which still has undergone major improvements with the handling of multiclass FCFS scheduling. We now briefly discuss the above solvers, emphasizing in particular their distinctive features.

3.2.1 *CTMC solver*

This solver focuses on generation and direct numerical solution of the Markov process underlying the extended queueing network. Each node is internally associated to a *Markovian agent*, in the process algebraic sense of papers such as (Harrison 2003), which can perform one of A active actions. An action can fire, if enabled, according to phase-type timers, after synchronizing with a passive action of the same type enabled in the same or other agents. Infinite state spaces are handled by user-defined truncation.

For each reachable state, the rates of the enabled synchronizations are used to define the infinitesimal generator Q of the Markov process. Instantaneous transitions are modeled by a small holding time ε, but such transitions are ultimately hidden from the final infinitesimal generator using the *stochastic complementation* method (Meyer 1989). The resulting stochastic complement Q^* is also an infinitesimal generator, which can be numerically analyzed either for steady-state metrics, possibly via mapping to GPUs using MATLAB's native support, or for transient values, which are obtained by integration of the Kolmogorov forward equations using MATLAB's `ode23` and `ode15s` (stiff) solvers. The CTMC solver also offers an implementation of the uniformization method (Bolch et al. 2006).

The CTMC solver can also mark individual transition rates in Q^* so to return a marked Markovian Arrival Process (MMAP) (Horvath and Telek 2019) to characterize particular events. Such a MMAP is defined by a phase-type distribution with generator D_0, and non-negative matrices D_a, $a = 1,\dots,A$, so that $Q^* = \sum_{i=0}^{A} D_i$. Actions are marked according to a global ontology of events that can occur at the supported nodes (e.g., arrivals, departures, phase changes, ...) is also maintained.

For example, this MMAP may be easily used to numerically evaluate the equilibrium distribution at arrival or departure instants using library functions shipped with LINE. For example, the equilibrium distribution seem upon occurrence of action a is given by the distribution $\pi_a \geq 0$ given by the system of linear equations

$$\pi_a = \pi_a(-D_0^a)^{-1} D_a \qquad ||\pi_a||_1 = 1 \tag{1}$$

where $D_0^a = D_0 + \sum_{\substack{i=1,\dots,A \\ i \neq a}} D_a$. From the distribution π_a embedded at the times where action a occurs, it is then possible to calculate moments and probabilistic performance metrics for particular stations or classes.

3.2.2 *JMT solver*

As discussed before, LINE allows to export and import JMT simulation models. The JMT solver carries out these transformations, after automatically retrieving JMT from its remote repository. The solver also allows M2M transformation to the JMVA application within JMT, which implements a number of advanced methods for product-form analysis.

3.2.3 *SSA solver*

The stochastic simulation analysis solver reuses a large part of the CTMC solver code in order to simulate Markov processes for which the state space is intractably large. State space generation is performed on-the-fly by recording the state of each Markovian agent during the simulation. To increase performance, each state in SSA is hashed. As the simulation evolves, the state vector size may also change, e.g., due to the increasing length of a FCFS buffer, and this is handled seamlessly by the SSA simulator, which also returns the full state trajectory.

Table 2: Some normalizing constant estimation methods available in the NC solver

Method	*Description*	*Ref.*	*Notes*
ca	Exact convolution algorithm	(Buzen 1973), (Reiser and Kobayashi 1975)	Exact
comom	Class-oriented method of moments	(Casale 2009)	Exact
mom	Method of moments	(Casale 2006)	Exact
mva	Mean value analysis	(Reiser and Lavenberg 1980)	Exact
le	Logistic asymptotic expansion	(Casale 2017)	Approximate
pana	Panacea asymptotic expansion	(McKenna and Mitra 1984),(Robertazzi 2000)	Approximate
mmint	McKenna-Mitra integral form	(McKenna and Mitra 1984)	Approximate
mci	Monte carlo integration	(Ross, Tsang, and Wang 1994)	Randomized
imci	Improved Monte carlo integration	(Wang, Casale, and Sutton 2016)	Randomized
ls	Logistic sampling	(Casale 2017)	Randomized

A fixed number of samples is obtained prior to termination of the simulation and independent replications are used to parallelize the simulation on multiple CPU cores. Sample paths can also be easily drawn using both the SSA and CTMC solvers, either in terms of the original state, an aggregate state (as discussed earlier for state initialization), or in terms of marginal states for particular nodes.

Empirically, we have found it useful to have two simulation environments within LINE, namely SSA and JMT, as this has helped in the mutual debugging of the two tools. Another lesson we have learned is that, for performance reasons, it is natural in ad-hoc simulators like JMT to generate random samples from Markovian distributions without scheduling on the simulator calendar each individual event corresponding to phase-type transitions internal to the distribution, but rather just looking at its time to absorption. However, this approach ultimately means that is is not possible to fully reconstruct the trajectory of the stochastic system, since phase transitions are hidden from the simulation logs. Thus, there exist in practice a trade-off between the speed of ad-hoc simulation tools such as JMT, and the granularity of the sample path information available in Markov process simulation tools such as SSA.

3.2.4 *FLUID solver*

The FLUID solver implements the mean-field approximation presented in (Pérez and Casale 2017) and its extensions to discriminatory processor sharing in (Zhu et al. 2020). Response time distributions and passage times can also be efficiently extracted, which are both critical to the support in LINE for random environment analysis using an implementation of the blending method (Casale et al. 2014).

The analysis of multiclass multi-server FCFS queueing stations is obtained by first replacing these nodes into PS stations with the same mean, and then by iteratively updating the mean service times at these nodes using the asymptotic decay rate of the diffusion approximation proposed in (Kobayashi 1974). As this method is also leveraged in the NC solver for probabilistic analysis, we point the reader to Section 4 for details of the approximation method.

3.2.5 *NC solver*

The NC solver focuses on analyzing probability distributions using exact and approximate product-form expressions for the network state probabilities. Upon demand, moments of performance metrics may also be estimated, although this is not the primary use case for this solver. To estimate the normalizing constant G for the resulting product-form, which is the quantity assuring that the distribution sums to one, LINE implements a range of exact and approximate methods to choose the best trade-off between complexity and approximation accuracy, as shown in Table 2. Because the techniques in the table require models to have a single constant-rate server, at multi-server stations this solver uses Seidmann's approximation (Seidmann et al. 1987), which replaces them where needed with a tandem sub-network consisting of a single-server station and an infinite server node. As mentioned, a decay rate approximation is used to model multiclass FCFS nodes, as we discuss later in Section 4.

An exact method in Table 2 that deserves further explanation is the option to use mean-value analysis (mva method) to obtain a normalizing constant $G(\vec{N})$, for a closed model with population vector $\vec{N} = (N_1, \dots, N_R)$. Since mean-value analysis exactly determines the class-r throughputs $X_r(\vec{N}')$ for all population vectors $\vec{N}' \leq \vec{N}$ and classes r (Reiser and Lavenberg 1980), normalizing constants are recursively computed as a by-product of exact mean-value analysis using the recurrence relation

$$G(\vec{N}) = \frac{1}{X_r(\vec{N})} G(\vec{N} - 1_r) \tag{2}$$

where 1_r is a vector of zeros but a one in the r-th position (Reiser and Kobayashi 1975). The termination condition is $G(\vec{0}) = 1$, with $\vec{0} = (0, \dots, 0)$. Numerical issues that affect normalizing constants are readily avoided by taking the logarithms of both sides of the recurrence relation. The recurrence relation also holds exactly for models with multi-server nodes, which are handled using MVALD-MX (Bruell et al. 1984).

3.2.6 *MVA solver*

The solver is primarily based on the Bard-Schweitzer approximate mean value analysis (AMVA) algorithm, but also offers and implementation of the exact MVALD-MX algorithm (Bruell et al. 1984). Non-exponential service times are handled using a M/G/1-type approximations as in (Reiser 1979). Multi-server queues are dealt with using the multi-server method in (Rolia and Sevcik 1995) where heterogeneous FCFS scheduling follows the standard AMVA-FCFS approximation (Bolch et al. 2006, Section 10.2). Multi-server PS is dealt with using the QD-AMVA approximation (Casale et al.. 2015).

DPS queues are analyzed with a time-scale separation method, so that for an incoming job of class r and weight w_r, classes with weight $w_s \geq 5w_r$ are replaced by high-priority classes that are analyzed using the method of the shadow server (Bolch et al. 2006, Section 10.3.2). Conversely the remaining classes are treated by weighting the queue-length seen upon arrival in class $s \neq r$ by the correction factor w_s/w_r.

The MVA solver also includes textbook methods to analyze simple single-class models. These include, among others, formulas for $M/M/k$, $GI/M/1$, and $M/G/1$ queueing stations. Various approximations for the $GI/G/k$ queue are also featured, such as Kingman's formula and the Kramer-Langenbach-Belz formula (Bolch et al. 2006). For closed systems, a variety of asymptotic bounds are also implemented, such as the geometric bounds (Casale et al. 2008a).

3.2.7 *MAM solver*

The matrix analytic method (MAM) solver depends externally on the BU tools library for matrix-analytic methods (Horvath and Telek 2019), which enables to solve quasi-birth death (QBD) processes as well as multi-class $MMAP[K]/PH[K]/1$ queues. Similarly to other solvers, Seidmann's approximation (Seidmann et al. 1987) is used to analyze the multi-server case.

In order to build the multiclass input to the queueing system under study, MAM considers the open chains in the model. If a chain c consists of R_c classes, each having a phase type renewal arrival process (α_r, T_r), where α_r is an initial probability vector and T_r a subgenerator, then the stream is first converted into a Markovian arrival process (Bolch et al. 2006, Section 6.8.3) and subsequently a Kronecker sum of the defining matrix is used to obtain the superposed arrival stream. This superposition is carried out so to retain the markings of the arrival types and finally generate a $MMAP[K]$ arrival process.

The MAM solver also features a basic decomposition approach for open queueing networks, where the arrival process at each node is obtained by re-scaling the mean of the external arrival process in each chain, described as a $MMAP[K]$, and considering in isolation each node as a $MMAP[K]/PH[K]/1$ queue. Also in this case, Seidmann's approximation is used to introduce a correction for the multi-server case.

4 Probabilistic analysis with the NC solver

In this section, we detail some of the novel methods implemented in LINE for probabilistic analysis, in particular within the NC solver, which supports only Markovian queueing networks. We are concerned in this section with computing the joint probability distribution $\pi(\vec{n})$ where $\vec{n} = (\vec{n}_1, \dots, \vec{n}_M)$, with $\vec{n}_i = (n_{i,1}, \dots, n_{i,R})$ counting the number of jobs $n_{i,r}$ at each station $i = 1, \dots, M$, in class $r = 1, \dots, R$ at steady-state.

4.1 Aggregate model

In order to calculate properties of chains that are used in the computation, such as aggregate visit ratios and mean service times for a chain, the solver first eliminates *class switching* nodes, which are topological elements where a job can instantaneously change class, but not station. To do so, the solver uses the stochastic complementation method for discrete-time Markov chains (DTMC) (Meyer 1989) applied to the routing matrix $P = [p_{i,j}^{r,s}]$ interpreted as a DTMC. Assume without loss of generality that P may be partitioned as

$$P = \begin{bmatrix} P_{11} & P_{12} \\ P_{21} & P_{22} \end{bmatrix}$$

where P_{11} captures the routing table entries relatively to station index pairs (i, j). Then we can define the stochastic complement as the square matrix of order RM

$$P^* = P_{11} + P_{12}(I - P_{22})^{-1} P_{21} \tag{3}$$

This matrix gives the routing probabilities between the queueing stations only, corrected in an exact fashion for the net effect of the class-switching nodes. Using P^*, the tool then computes the mean visit ratios $v_{i,c}^*$ and mean service time $s_{i,c}^*$ of each chain $c = 1, \dots, C$ at each node i, using standard methods (Bolch et al. 2006, Section 7.3.6.1). This enables the definition of an aggregate model consisting of a network without class-switching and with C classes, each chain c having population N_c^* given by the sum of the number of jobs N_s in all classes s belonging to that chain.

Let $\vec{N}^* = (N_1^*, \dots, N_C^*)$ be the population vector for the aggregated model. If the queueing network model satisfies the standard product-form assumptions, it is then possible to compute state probabilities for the original model using the exact closed-form expression

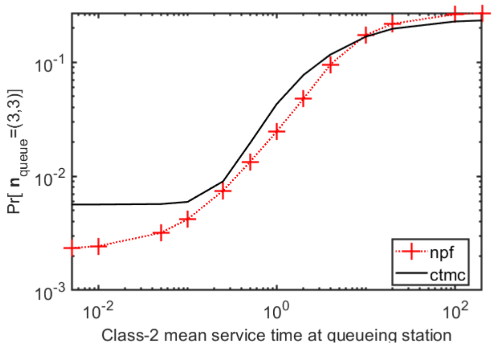
$$\pi(\vec{n}) = \frac{1}{G(\vec{N}^*)} \prod_{i=1}^{M} n_i! \prod_{r=1}^{R} \frac{(v_{i,r} s_{i,r})_{i,r}^{n_{i,r}}}{n_{i,r}! \prod_{k=1}^{c} \min(n_i, m_i)} \tag{4}$$

where m_i is the number of servers for queueing station i. A difference of (4) compared to the usual product-form expression is that $G(\vec{N}^*)$ is the normalizing constant for the aggregated model without class-switching (Zahorjan 1979), and with parameters $v_{i,c}^*$ and $s_{i,c}^*$, whereas the remaining terms depend on the classes in the original model. Another difference is that the range of the state vector $\vec{n}$ extends to all distribution of jobs across the stations allowed under the class-switching mechanism, which is far larger than for a multiclass model.

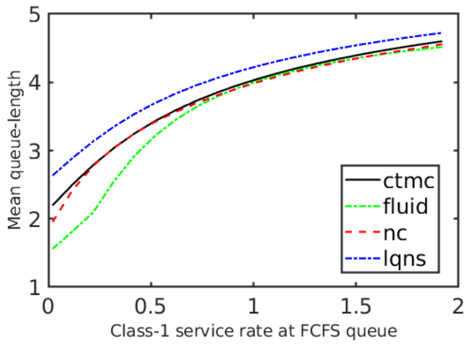
4.2 Non-product form queueing analysis

The analysis techniques described in the previous sections are leveraged in the NC solver to approximately analyze non-product form models with FCFS scheduling and heterogeneous mean service times. Albeit these models do not feature in general a product-form result, LINE approximates their solution by iteratively replacing in (4) the service times $s_{i,r}$ of each class $r = 1, \dots, R$ with new values $s_{i,r}[k]$ that capture the multiclass FCFS scheduling effects as explained below.

At the first iteration, the solver evaluates a product-form model where every FCFS node i is replaced by a PS station with service times $s_{i,r}$. At the following iterations, the service times at FCFS stations are



(a)



(b)

Figure 2: Accuracy of estimates for two non-product form model with multiclass FCFS.

updated using a convex combination of the initial values with the service time that would have produced the asymptotic decay rate of the diffusion approximation in (Kobayashi 1974). In applying the latter, we consider the station in isolation with an arrival stream treated as a *single-class* process. Define $X_{i,r}[k]$ as the throughput of class r jobs at station i at iteration k, and let $X_i[k] = \sum_{r=1}^{R} X_{i,r}[k]$. Station i is assumed to have m_i servers. Then at iteration $k+1$, we use the interpolation

$$s_{i,r}[k+1] = (1-a_i)s_{i,r} + a_i(b_i\eta_i + (1-b_i)\gamma_i)m_i/X_i[k] \tag{5}$$

where $a_i \in [0,1]$ and $b_i \in [0,1]$ are interpolation coefficients. Empirically, we have found that the assignment $a_i = b_i = \rho_i^4[k]$ can give fairly accurate estimates, where $\rho_i[k]$ denotes the utilization of station i at iteration k. The terms γ_i and η_i are decay rates to approximate multiserver G/G/1 nodes (Gautam 2012, Sec 7.2.2)

$$\gamma_i = (\rho_i[k]^{m_i} + \rho_i[k])/2 \tag{6}$$

and the heavy-load queue-length decay rate of a $M/G/1$ queue

$$\eta_i = \exp\left(-2(1-\rho_i[k])/(C_i^2[k] + \rho_i[k])\right) \tag{7}$$

where

$$C_i^2[k+1] = \sum_{r=1}^{R} X_{i,r}[k]c_{i,r}^2/X_i[k] \tag{8}$$

in which $c_{i,r}^2$ is the squared coefficient of variation of the service times at node i for class r. The initial conditions are $\rho_i[1] = 0$ and $C_i^2[1] = 1$, $\forall i$. We are not aware of the above interpolation having appeared before in the literature.

5 Numerical examples

Probabilistic analysis. In this example, we look at the quality of the heterogeneous FCFS approximation across the solvers we have presented in a simple model with exponential distributions. Figure 2(a) illustrates the approximation presented in Section 4 (npf curve) against exact numerical solution of the underlying Markov process (ctmc curve). The model has a cyclic topology and consists of an infinite service node and a FCFS queue. There are $R = 2$ classes of jobs, each with $N_r = 3$ jobs. The service time at the infinite server station is exponential with mean $Z_r = 2$ for both classes. Class 1 at the FCFS queue receives exponential service time at rate 1.0, whereas class 2 receives hyper-exponential service with squared coefficient-of-variation 9 and mean shown in the horizontal axis of Figure 2(a). With probability 0.5, jobs in class 1 can switch to class 2 upon departing either station. Conversely, class 2 jobs switches with probability 1 into class 1 only upon returning from the queue to the delay.

Figure 2(a) plots the probability of the state where all jobs reside at the FCFS queue. This is obtained by the NC solver plugging the estimated $s_{i,r}$ in the standard probabilistic expressions for product-form closed queueing networks. In general, this is a very difficult approximation problem since the exact solution of the underpinning stochastic model is not theoretically understood, and thus approximations are heuristic in nature. Nevertheless, the results suggests that the approximation follows qualitatively the exact trend. Probabilistic metrics of this kind are not available with the MVA and FLUID solvers used in the previous example, however they are available as original research contributions within LINE.

Mean-value analysis. We consider the same topology as in the previous example, but without class-switching. We require all distributions at the queue i to be Erlang-2 and set the mean service time at the infinite service node to be $Z_r = 1$ in both classes, and the mean service time of class 2 at the FCFS queue to be 1. We vary the mean service time of class 1 at the FCFS queue between 1/50 and 2. The number of jobs in both classes is $N_r = 3$. The FCFS queue has now $m_i = 2$ servers.

Figure 2(b) illustrates the results, comparing the exact results, computed with the CTMC solver, with the FLUID and NC approximations, and the FCFS approximation available in LQNS based on MVA. The results suggest that NC is more accurate than the other methods for lower values of the service time, while for larger values the FLUID solver also becomes progressively more accurate. LQNS also features very good accuracy. In other examples, we have seen similar trends, but which method performs best depends on the queueing network under study. For example, if the number of servers in the considered model is varied to $m_i = 1$, then LQNS is the most accurate method, while for $m_i = 4$ the FLUID solver has the highest precision. The advantage of an integrated environment such as LINE is to ease the use of multiple approximation techniques for a given model.

6 Conclusion

In conclusion, we have presented LINE 2.0, a MATLAB-based software library to specify and solve extended queueing networks. Models are solved either with native or external solvers, computing metrics such as throughputs, utilizations, response times, queue-lengths, and state probabilities. We have also illustrated the numerical support available in LINE for analytical estimation of state probabilities in non-product form models. The approach leverages a novel use of the diffusion approximation to analyze multiclass closed networks. In future work, we aim at extending the solvers to a broader spectrum of matrix-analytic methods.

REFERENCES

Bertoli et al. 2006. “Java Modelling Tools: an Open Source Suite for Queueing Network Modelling and Workload Analysis”. In *Proc. of the 3rd Conf. on Quantitative Evaluation of Systems (QEST)*, 119–120: IEEE.

Bobbio, A., A. Horváth, and M. Telek. 2005. “Matching Three Moments with Minimal Acyclic Phase Type Distributions”. *Stochastic Models* 21(2-3):303–326.

Bolch et al. 2006. *Queueing Networks and Markov Chains*. Wiley.

Bruell et al. 1984. “Mean value analysis of mixed, multiple class BCMP networks with load dependent service stations”. *Performance Evaluation* 4:241–260.

Buzen, J. P. 1973. “Computational Algorithms for Closed Queueing Networks with Exponential Servers”. *Comm. of the ACM* 16(9):527–531.

Casale, G. 2006. “An Efficient Algorithm for the Exact Analysis of Multiclass Queueing Networks with Large Population Sizes”. In *Proc. of joint ACM SIGMETRICS/IFIP Performance*, 169–180: ACM Press.

Casale, G. 2009. “CoMoM: Efficient Class-Oriented Evaluation of Multiclass Performance Models”. *IEEE Trans. on Software Engineering* 35(2):162–177.

Casale, G. 2017. “Accelerating Performance Inference over Closed Systems by Asymptotic Methods”. In *Proc. of ACM SIGMETRICS*: ACM Press.

Casale, G. 2019. “Automated Multi-paradigm Analysis of Extended and Layered Queueing Models with LINE”. In *Proc. of ICPE*, 37–38.

Casale et al. 2008a. “Geometric Bounds: A Noniterative Analysis Technique for Closed Queueing Networks”. *IEEE Trans. Computers* 57(6):780–794.

Casale et al. 2008b. “KPC-Toolbox: Simple Yet Effective Trace Fitting Using Markovian Arrival Processes”. In *Proc. of the 5th Conf. on Quantitative Evaluation of Systems (QEST)*, 83–92.

Casale et al. 2014. “Blending randomness in closed queueing network models”. *Perform. Eval.* 82:15–38.

Casale et al.. 2015. “QD-AMVA: Evaluating Systems with Queue-Dependent Service Requirements”. In *Proc. of IFIP PERFORMANCE*.

Franks, G. 1999, December. *Performance Analysis of Distributed Server Systems*. Carleton University.

G. Franks et al. 2009. “Enhanced Modeling and Solution of Layered Queueing Networks”. *IEEE Trans. Softw. Eng.* 35(2):148–161.

Gautam, N. 2012. *Analysis of Queues*. CRC Press.

Gunther, N. J. 2011. *Analyzing Computer System Performance with Perl:PDQ (2. ed.)*. Springer.

Harrison, P. G. 2003. “Turning back time in Markovian process algebra”. *Theor. Comput. Sci* 290(3):1947–1986.

Heinrich *et al.* 2018. “The palladio-bench for modeling and simulating software architectures”. In *ICSE (Companion Volume)*, 37–40: ACM.

Horvath, G., and M. Telek. 2019. “Markovian Performance Evaluation with BuTools”. In *Systems Modeling: Methodologies and Tools*, edited by A. Puliafito and K. S. Trivedi, 253–268. Springer.

Kobayashi, H. 1974. “Application of the Diffusion Approximation to Queueing Networks I: Equilibrium Queue Distributions”. *J. ACM* 21(2):316–328.

Marzolla, M. 2014. “The Octave Queueing Package”. In *Proc. of QEST*, 174–177.

McKenna, J., and D. Mitra. 1984, April. “Asymptotic Expansions and Integral Representations of Moments of Queue Lengths in Closed Markovian Networks”. *JACM* 31(2):346–360.

Meyer, C. D. 1989. “Stochastic Complementation, Uncoupling Markov Chains, and the Theory of Nearly Reducible Systems”. *SIAM Review* 31:240–272.

Pérez, J. F., and G. Casale. 2013. “Assessing SLA Compliance from Palladio Component Models”. In *Proc. of the 2nd MICAS*.

Pérez, J. F., and G. Casale. 2017, Sept. “Line: Evaluating Software Applications in Unreliable Environments”. *IEEE Tran. on Reliability* 66(3):837–853.

Reiser, M. 1979. “A Queueing Network Analysis of Computer Communication Networks with Window Flow Control”. *IEEE Trans. on Communications* 27(8):1199–1209.

Reiser, M., and H. Kobayashi. 1975. “Queueing Networks with Multiple Closed Chains: Theory and Computational Algorithms”. *IBM J. Res. Dev.* 19(3):283–294.

Reiser, M., and S. S. Lavenberg. 1980. “Mean-value Analysis of Closed Multichain Queueing Networks”. *JACM* 27(2):312–322.

Robertazzi, T. G. 2000. *Computer Networks and Systems*. Springer.

Rolia, J. A., and K. C. Sevcik. 1995. “The Method of Layers”. *IEEE Trans. on Software Engineering* 21(8):689–700.

Ross, K., D. Tsang, and J. Wang. 1994. “Monte carlo summation and integration applied to multiclass queueing networks.”. *JACM* 41(6):1110–1135.

Seidmann et al. 1987. “Computerized closed queueing network models of flexible manufacturing systems: A comparative evaluation”. *Large Scale Systems* 12:91–107.

Wang, W., G. Casale, and C. A. Sutton. 2016. “A Bayesian Approach to Parameter Inference in Queueing Networks”. *ACM Trans. Model. Comput. Simul.* 27(1):2:1–2:26.

Zahorjan, J. 1979. “An Exact Solution Method for the General Class of Closed Separable Queueing Networks”. In *Proc. of ACM SIGMETRICS*, SIGMETRICS ’79, 107–112. New York, NY, USA: Association for Computing Machinery.

Zhu et al. 2020. “Fluid approximation of closed queueing networks with discriminatory processor sharing”. *Performance Evaluation* 139:102094.

ACKNOWLEDGMENTS

The paper is supported by the Horizon 2020 project RADON (825040). The development of LINE has been partially funded in the past by the EC grants FP7-318484, H2020-644869, and by the EPSRC grant EP/M009211/1.

AUTHOR BIOGRAPHY

GIULIANO CASALE joined the Department of Computing at Imperial College London in 2010, where he is currently a Reader in modeling and simulation. Previously, he worked as a scientist at SAP Research UK. He teaches and does research in performance engineering and cloud computing, topics on which he has published more than 100 refereed papers. He has served on the technical program committee of over 80 conferences and workshops. He serves on the editorial boards of IEEE TNSM and ACM TOMPECS and as current chair of ACM SIGMETRICS. His email address is g.casale@imperial.ac.uk.