A Mixture Density Network Approach to Predicting Response Times in Layered Systems

Zifeng Niu
Department of Computing
Imperial College London
London, UK
zifeng.niu19@imperial.ac.uk

Giuliano Casale
Department of Computing
Imperial College London
London, UK
g.casale@imperial.ac.uk

***Abstract*—Layering is a common feature in modern service-based systems. The characterization of response times in a layered system is an important but challenging analysis dimension in Quality of Service (QoS) assessment. In this paper, we develop a novel approach to estimate the mean and variance of response time in systems that may be abstracted as layered queueing networks. The core step of the method is to obtain the response time distributions in the submodels that are used to analyze the layered queueing networks by means of decomposition. We model the conditional response time distribution as a mixture of Gamma density functions for which we learn the parameters by means of a Mixture Density Network (MDN). The scheme recursively propagates the MDN predictions through the layers using phase-type distributions and performs convolutions to gain the approximation of the system delay. The experimental results show an accurate match between simulations and MDN predictions and also verify the effectiveness of the approach.**

***Index Terms*—Quality of Service, response time distribution, queueing network, mixture density network, layered system**

I. Introduction

The end-to-end response time experienced by customers is an important aspect of service quality assurance. The response time serves as a key performance index, but in some situations the mean value of this metric is insufficient for system designers, who may want to know the quantile statistics or the extent of variability in order to assess compliance to service level objectives. Although the problem of approximating response time distributions in queueing networks (QN) has been extensively studied during the last few decades, it still remains a challenging task because conventional analytic methods are inherently limited by their underpinning technical assumptions, such as those that prescribe overtake-free job behavior under first come first serve (FCFS) scheduling [1]. In addition, many services today such as distributed software applications or business processes feature concurrent services structured according to a layered structure. Such systems are not simple to model using ordinary queueing networks since layering is a form of simultaneous resource possession that does not meet the product-form solution assumptions [2]. The lack of simple closed-form analytical solutions poses a challenge for the estimation of response time distributions.

In this paper, we propose a method to provide accurate estimations on variance as well as mean value of the response time in layered systems. The method requires estimating response time distributions in a set of closed queueing networks that are constructed from a layered queueing network (LQN). However, the current analytic methods are not stable to use since they are restricted by strict assumptions on service discipline, number of servers, and Markovian networks. Fortunately, machine learning (ML) poses a number of opportunities and one of the goals of this paper is to understand how ML can benefit QN and LQN theory. We use a suitable ML method named Mixture Density Network (MDN) to approximate response time distribution conditional on the given information of a closed queueing network. The MDN uses a mixture model to fit probability distribution and uses a fully-connected neural network to map the conditional relation. Armed with MDNs, our approach is capable of estimating response times in real complex layered systems and it shows a high accuracy in experiments. Because our analytic method is based on the ML technique, there is a training phase and a test phase to use the embedded analyzer, which is not typical in conventional LQN analysis. This approach thus offers a novel contribution to the field of QoS assessment for layered systems.

The rest of the paper is organized as follows: Section II conducts a brief review on response time distribution analysis and prediction. Section III describes the concepts and related techniques that are used in this paper. Section IV presents how we apply MDN. Section V proposes the approach to estimating response times in a layered system. Section VI evaluates the performance of our MDNs and approach. Section VII concludes the paper and gives future research directions.

II. State of the Art

In closed queueing networks, the tagged customer method is commonly used to derive the response time distributions [3]. A typical method introduced in [1], [3] is to derive the Laplace–Stieltjes transform (LST) of passage time from start to end for the tagged job and decondition the start state to obtain the product-form LST of steady-state response time distribution. However, this method can only be applied when the service discipline is FCFS. In addition, its state-space analysis suffers from state explosion when the population is large. Fluid approach has been used successfully to approximate the response time distributions in closed queueing networks that consist of delay and queueing stations with processor sharing

(PS) discipline [4]. The discrete number of jobs at the station is approximated by a fluid continuous amount so that the state explosion no longer exists. Here, the fluid method is based on Kurtz's theorem [5] which is hard to use in networks with multiclass FCFS stations. All the methods discussed above are limited to Markovian queueing networks. There are few analytic methods for the networks with non-Markovian service time distributions at the present moment. For what concerns layered queueing networks, they are a type of non product-form queueing network where the service time of one station may depend on other stations [6]. The existing technique approximates the variance of response time in a LQN by taking the sum of variance estimates of associated services given by analytic methods [7]. However, the results can be inaccurate in real complex systems. The limitations of conventional analysis have led to an interest in statistical learning methods which make it possible for us to capture the pattern under more realistic situations. Among these methods, MDN technique is well suited for approximating unknown conditional probability distribution. The MDN was first proposed in [8]. Recently, it has been widely used in distinct areas [9], [10], [11], [12], [13] and proved to be an effective predictor. In the QN area, [14], [15] have used MDNs to predict response time distributions in open queueing networks and service function chains.

III. Background

A. Layered Queueing Networks

Fig. 1 shows a LQN model that contains all of the features considered in this paper. In LQN models, resources exposing a service are called *tasks* and shown as parallelograms. The tasks are executed by hardware resources called *host processors*, represented by the attached circles. The tasks and processors usually have FCFS and PS service discipline respectively. The stacked notation makes a task or a processor be a multiserver and the multiplicity is indicated in brace. For instance, *T5* denotes a task with five threads and runs on *P5* which could be a two-CPU multiprocessor. Distinct classes of service are called *entries* and represented by smaller parallelograms contained by tasks. Each entry is specified by a set of operations called *activities* shown as rectangles. The host demand of each activity is an independent exponential random variable and the mean duration is indicated in square bracket. For ease of presentation, we consider sequential activities in the paper. Our method can be easily generalized to non-sequential patterns.

A key feature of layered queueing models is that a service may include nested *calls* to other services. The calls model the service requests and they are represented by directed arrows going from one activity to an entry of another task. There are two main types of call: *synchronous* and *asynchronous*. The synchronous call is a blocking request during which the sender waits for the reply (dashed arrow), while the asynchronous call is a send-no-reply request—the execution continues after sending the call. In this paper, we focus on synchronous call, for example, it is the pattern of standard remote procedure calls (RPC) that exists in most layered queueing systems. The amount of calls indicated in parenthesis alongside the request

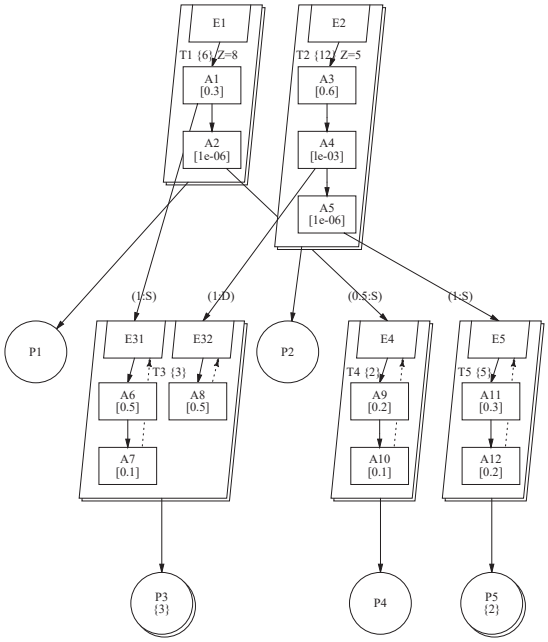


Fig. 1. Example of a LQN model.

arrow is either *deterministic* or *stochastic*, both cases are considered in our method. The amount of stochastic requests is assumed to be geometrically distributed and the number in parenthesis is the mean value. In Fig. 1, the tasks *T1* and *T2* do not receive any calls. They are system workload generators, or used to represent the end users, and named *reference tasks*. The think time of a reference task is denoted by Z. The interested readers are referred to the references [16], [17] for a more detailed description of the LQN models.

In the analysis of a LQN, it is decomposed into a set of submodels (ordinary queueing networks) that can be solved by product-form analysis techniques, such as approximate mean value analysis (AMVA) [18]. The solution iterates among all submodels until meeting certain convergence criteria and then we can obtain the mean performance metrics of the LQN [6]. SRVN-layering [19] and MOL-layering [20] are two ways to construct submodels. In this paper, we adopt SRVN-layering whose submodel consists of a delay and a queueing station.

B. Class Switching and Chains

In a closed queueing network with N stations and K job classes, the routing matrix $\boldsymbol{R} = [r_{iu,jv}]$, $i, j = 1, \ldots, N$, $u, v = 1, \ldots, K$ defines a finite-state discrete-time Markov chain (DTMC). The station-class pair (j, v) can be reached from pair (i, u) with probability $r_{iu,jv}$. Jobs can switch class in the network if there exists $r_{iu,jv} \neq 0$ for $u \neq v$. All job classes can be divided into $0 < L \leq K$ disjoint sets named $chains$. Each chain is a set of classes that can switch jobs with each other, but not with classes in other chains. This means jobs inside a chain will never escape from it. Therefore, the number of jobs in each chain remains constant at all times. It is natural to think of transferring a closed queueing network with L chains to a closed queueing network with L independent job classes and measuring the chain performance [21]. The

methods have been developed to calculate the performance metrics per chain and then per class in each chain [22], [23]. A brief description is as follows. We first derive the population, number of visits at each station, and service time for each of the L chains. Next, the performance metrics per chain can be calculated by AMVA in the same way as for multiclass queueing network. Then, we use the population and throughput of each chain to derive the mean response time and throughput per class in that chain. Based on these two measures, all other metrics per class can be obtained in the end. We refer the interested readers to the reference [24, §7.3.6].

C. Phase-type Distribution and Closure Property

In this paper, we cope with various response time distributions in a layered system by phase-type (PH) distributions. PH distribution is defined as the distribution of time to absorption in a finite continuous-time Markov chain (CTMC) with an absorbing state. It is uniquely determined by a pair (α, T), where α and T are the initial probability vector and subgenerator matrix among the transient states of the CTMC. The PH distribution function and moments can be expressed in terms of α and T [25] as follows

$$\begin{aligned} F(x) &= 1 - \alpha e^{Tx}\mathbf{1} \\ E\left[X^j\right] &= (-1)^j j! \alpha T^{-j}\mathbf{1}, \quad j = 1, 2, \ldots \end{aligned} \tag{1}$$

where $x \geq 0$ and $\mathbf{1}$ is a column vector of ones.

The closure properties of PH distribution are studied in [25]. They are important because these properties allow us to replace numerical representations with matrix ones in complex operations such as finite convolutions, maximum and finite mixtures, which make it possible for us to process probability distribution in an algebraic and tractable form. Here we cite one property relevant to this paper:

If X_i and X_j are statistically independent random variables with PH distributions (α_i, T_i) and (α_j, T_j), then $X_i + X_j$ is a PH random variable with (α, T), where

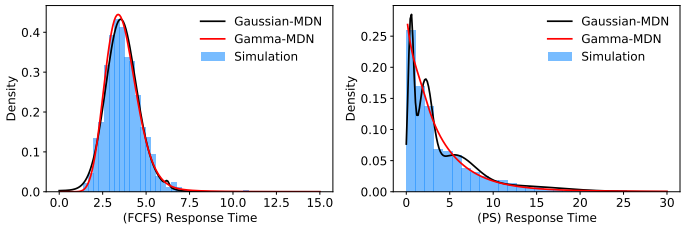
$$\alpha = (\alpha_i, (1 - \alpha_i \mathbf{1})\, \alpha_j), \quad T = \begin{pmatrix} T_i & -T_i \mathbf{1} \alpha_j \\ 0 & T_j \end{pmatrix} \tag{2}$$

PH distribution is an effective fitting tool in matching the first and second moments. Even the underlying distribution is known, it is preferable to fit a PH distribution to the first two moments instead of using the original distribution that may make it hard to incorporate into certain algebraic operations [26]. In summary, the PH fitting technique is always able to return a distribution (α, T) with the same mean and variance. The interested readers are referred to the reference [27] for details of the fitting procedure.

IV. Mixture Density Networks

Mixture density network (MDN) is an elegant combined structure of a mixture model and a fully-connected neural network. We start from introducing the mixture model—a weighted sum of N individual probability density functions

$$f(y) = \sum_{i=1}^{N} \pi_i p_i(y|\theta_i) \tag{3}$$



(a) (b)

Fig. 2. A comparison between simulation and predicted conditional density functions of response time given the SRVN submodel information: N = 90, Z = 3, D = 0.6, c^2 = 1.25, S = 8 with (a) FCFS and (b) PS service discipline.

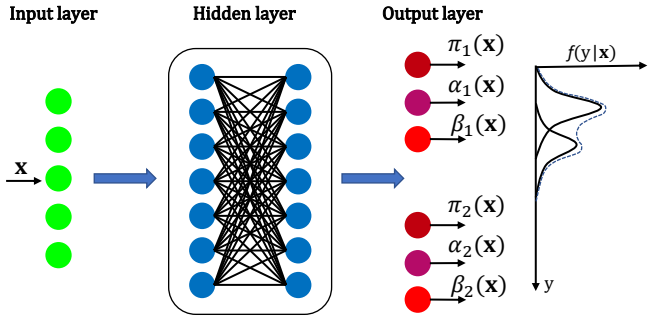


Fig. 3. The structure of Gamma-MDN.

where π_i are mixing coefficients such that $0 \leq \pi_i \leq 1$ and $\sum_i \pi_i = 1$, and θ_i are parameters that characterize N kernel functions. Mixture model has the potential to express arbitrary probability distributions [8], thus we can use it to model various response time distributions in queueing networks.

Given the LQN submodel information as an input vector $\mathbf{x}$, we wish to obtain its response time distribution expressed by a mixture model. In other words, our purpose is to build a conditional mixture density function of y given $\mathbf{x}$

$$f(y|\mathbf{x}) = \sum_{i=1}^{N} \pi_i(\mathbf{x}) p_i(y|\theta_i(\mathbf{x})) \tag{4}$$

In this case, the weight and density parameters become functions of $\mathbf{x}$: $\pi_i(\mathbf{x})$ and $\theta_i(\mathbf{x})$. The input $\mathbf{x}$ determines a unique mixture model through these two functions. We therefore need to find the functional relationships $\pi_i(\cdot)$ and $\theta_i(\cdot)$ respectively. The central idea of MDNs is to model both functions by a fully-connected neural network which has the flexible fitting capability for complicated relationships. Let O denote the number of observed samples, a MDN is trained by the dataset $\{(\mathbf{x}_j, y_j) \mid 1 \leq j \leq O\}$ and learns the weights of the neural network through maximizing the likelihood

$$L = \prod_{j=1}^{O} f(y_j|\mathbf{x}_j) \tag{5}$$

In MDNs, there are different types of kernel functions that can be used. The majority of applications choose Gaussian kernels as mixture components [9], [10], [11], [12], [14], [15],

a few applications have ever used other kernels such as Gamma or Binomial [13]. In this paper, we use Gamma-MDNs to predict response time distributions in submodels. With the same number of components, a MDN with Gamma kernels provides a better performance compared to that with Gaussian kernels, especially for the models under the PS scheduling discipline. For instance, Fig. 2 shows a comparison between one histogram and two density functions obtained from simulation and five-component MDNs respectively. As can be seen from Fig. 2, both MDNs provide accurate predictions for the FCFS case, but for the PS case, the MDN with Gamma kernels provides a more accurate approximation.

The Gamma kernel γ is a two-parameter (θ=$\{\alpha, \beta\}$) probability density function with $\alpha > 0$, $\beta > 0$. Therefore, a Gamma-MDN needs to model three functional relationships $\alpha_i(\cdot)$, $\beta_i(\cdot)$, and $\pi_i(\cdot)$ to determine the conditional density function of variable of interest

$$f(y|\mathbf{x}) = \sum_{i=1}^{N} \pi_i(\mathbf{x}) \gamma_i(y|\alpha_i(\mathbf{x}), \beta_i(\mathbf{x})) \tag{6}$$

Fig. 3 shows the structure of Gamma-MDN. The output layer consists of three types of nodes that return the function values $\pi_i(\mathbf{x})$, $\alpha_i(\mathbf{x})$, and $\beta_i(\mathbf{x})$. The first type employs softmax as an activation function to turn real values into mixing coefficients that sum to one. For the second and third types, the return values should be always positive. We employ the following activation function [12] to ensure this condition

$$g(x) = 1 + ELU(\xi, x) \tag{7}$$

where $ELU(\xi, x)$ is Exponential Linear Unit proposed by [28], and ξ is a positive hyperparameter

$$ELU(\xi, x) = \begin{cases} \xi\,(e^x - 1) & x < 0 \\ x & x \geq 0 \end{cases} \tag{8}$$

This activation function is more stable compared to exponential activation function because the growth is not fast on the positive side. It helps to avoid instability during the training process [12]. The training makes the neural network learn its weights through minimizing our objective loss function that is defined as the negative log-likelihood.

V. Methodology

In this section, we propose an approach to estimate response time distributions in layered systems. We first study SRVN submodels with 1-2 job classes because later we will reduce submodels with >2 classes to this case. To illustrate the principle, four MDNs are trained by four distinct datasets: I. Single class, FCFS, II. Single class, PS, III. Two classes, FCFS, IV. Two classes, PS. Therefore, the input is $\mathbf{x}$ = (N, Z, D, c^2, S) from I, II; and $\mathbf{x}$ = (N_1, Z_1, D_1, c_1^2, N_2, Z_2, D_2, c_2^2, S) from III, IV. The definitions of input parameters are shown in Table I. When applying our methodology to a target layered system, we build a dataset by simulating randomly generated pairs in which the values of parameters N, Z, D, c^2, S fall within the valid parameter ranges of the potential problems we are trying to solve, and train the embedded MDN analyzer. In this way, our approach possesses the generalization capabilities to analyze real complex layered systems.

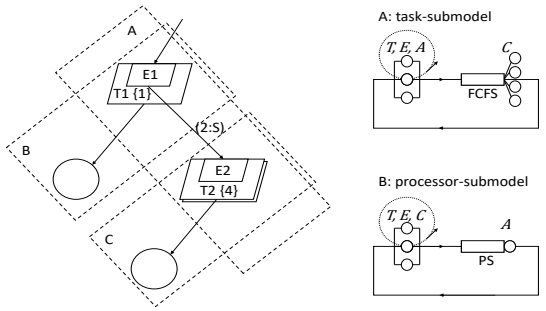


Fig. 4. Task-submodel and Processor-submodel.

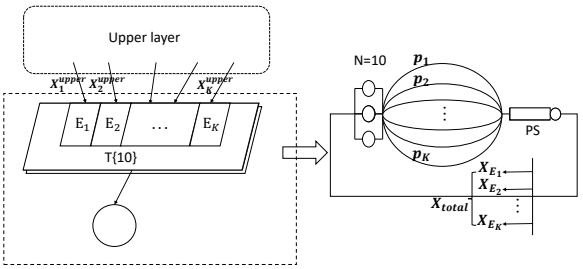


Fig. 5. A submodel with multiple entries.

A. Mechanism

As mentioned earlier, we use SRVN-layering to construct submodels. Each SRVN submodel can be characterized by a closed queueing network with a delay and a queueing station. The queueing station is either a processor or a task. According to the type of queueing station, we define two terms: $processor\text{-}submodel$ or $task\text{-}submodel$. Our method starts from analyzing processor-submodels (Algorithm 1) and then performs convolution of the response time distributions layer by layer, from bottom to top (Algorithm 2). We give each submodel four main job classes: $Task(T)$, $Entry(E)$, $Activity(A)$, and $Call(C)$; the switching happens between each other within the closed network. In a layered system, the operations of each task are executed on the host processor. For this reason, in a processor-submodel we let queueing station serve A-class jobs and delay station serve the rest of the classes. Meanwhile, a task may send requests to the services exposed by other tasks. Therefore, in a task-submodel we let queueing station serve C-class jobs and delay station serve the rest. As the example shown in Fig. 4, the switching circle for processor-submodel B is $(Delay, T) \rightarrow (Delay, E) \rightarrow (Queueing, A) \rightarrow (Delay, C) \rightarrow (Delay, T)$, while the switching circle for task-submodel A is $(Delay, T) \rightarrow (Delay, E) \rightarrow (Delay, A) \rightarrow (Queueing, C) \rightarrow (Delay, T)$. We first obtain mean performance metrics for each submodel through the iterations [6]. Next we use the technique introduced in Section III-B to calculate the metrics for each job class (lines 1-3 in Algorithm 1). In this step we get think times of classes T, A, and C named Z_T, Z_A, and Z_C. The think

times of processor-submodel and task-submodel are $Z_T + Z_C$ and $Z_T + Z_C + Z_A$ (Z_C is zero if there is no call sent outside the submodel) respectively. After this we fit PH to the service time distribution of each activity and then convolve all associated PH distributions for each entry (line 8 in Algorithm 1). Then we feed submodel information to a Gamma-MDN and get a mixture distribution with mean and variance

$$\mu = \sum_{i=1}^{N} \pi_i \mu_i, \quad \sigma^2 = \sum_{i=1}^{N} \pi_i \Big(\sigma_i^2 + \mu_i^2 - \mu^2 \Big) \tag{9}$$

where π_i, μ_i, and σ_i^2 are the kernel parameters which are returned by this MDN. We therefore can calculate the mean and variance of the obtained mixture model by (9). Next, we begin analyzing *bottom task-submodels* (line 4 in Algorithm 2). We define this term as a task-submodel whose server has a known response time distribution. For instance, in Fig. 4 the submodel A becomes a bottom task-submodel after calculating the processor-submodel C because Gamma-MDN gives us the response time distribution of E2, the server of submodel A. Let w and Φ denote the response time of server and the number of requests to server, we define a new variable $\delta = \Phi w$. Assuming Φ and w are independent random variables, we can derive mean and variance of δ

$$\mu_\delta = \mu_\Phi \mu_w, \quad \sigma_\delta^2 = \sigma_\Phi^2 \sigma_w^2 + \sigma_\Phi^2 \mu_w^2 + \sigma_w^2 \mu_\Phi^2 \tag{10}$$

If the request is deterministic, Φ should be an integer with $\sigma_\Phi^2 = 0$. If it is stochastic, Φ will be a geometric random variable with $\mu_\Phi = (1-p)/p$ and $\sigma_\Phi^2 = (1-p)/p^2$. Thus

$$\sigma_\delta^2 = \begin{cases} \sigma_w^2 \mu_\Phi^2 & \text{deterministic} \\ \dfrac{\sigma_w^2 \Big(p^2 - 3p + 2 \Big) + \mu_w^2 \Big(1 - p \Big)}{p^2} & \text{stochastic} \end{cases} \tag{11}$$

We use $(\mu_\delta, \sigma_\delta^2)$ to characterize the service time distribution in a bottom task-submodel. Then we use a Gamma-MDN to forecast the response time distribution and fit PH to this prediction. One entry may have several servers that execute its requests in distinct task-submodels. Once we complete such analysis for each of these, we convolve all of the fitted distributions in its host processor and task-submodels and finally get the response time distribution of this entry (lines 16-18 in Algorithm 2). We repeat this procedure until we obtain the response time distribution for every entry.

B. Class Aggregation

A submodel may have multiple chains. In this case, we use a two-class MDN to predict response time distribution for each chain. Suppose there are $n+1$ chains in the model and we are interested in the response time of a certain chain, then we need to aggregate the remaining n chains so that a two-class MDN can be used. We employ the following aggregation method inspired by the utilization law

$$N_{aggr} = \sum_{i=1}^{n} N_i, \quad M_{aggr} = \sum_{i=1}^{n} X_i M_i \Big/ \sum_{i=1}^{n} X_i \tag{12}$$

TABLE I
SIMULATION PARAMETERS

Parameter	Definition	Range of Value
N	Population	5-100
Z	Think time	0.5-10
D	Host demand mean	0.1-1
c^2	Host demand SCV	0.25-2.5
S	Number of servers	1-32

TABLE II
SYMBOL TABLE FOR ALGORITHM

Symbol	Meaning
s	submodel index
c	chain index
t	task index
e	entry index
$\varepsilon(t)$	The set of entries for task t
$Z_T(t)$	The mean idle time spent by task t from response until next invocation
$Z_C(t)$	The mean delay a task t spends acquiring services from other tasks
$Z_A(t)$	The mean delay a task t spends on its own processor
$\mathcal{A}(e)$	The set of activities for entry e
$\mathcal{D}(e)$	The set of fitted PH service time distributions for $\mathcal{A}(e)$
$\mathcal{R}_e$	Response time distribution of entry e
$\mathcal{D}_e$	The set of probability distributions that determine $\mathcal{R}_e$
$\mathcal{E}_e$	The set of servers the entry e communicates with
Δ	Service time distribution in a task-submodel
$\mathcal{P}$	The set of processor-submodels for LQN
$\mathcal{T}$	The set of task-submodels for LQN
$\mathcal{B}$	The set of bottom task-submodels at the present
$\mathcal{C}$	The set of chains in a submodel
Ψ	Service discipline (FCFS, PS)
$\lvert \cdot \rvert$	Number of elements in a set
$x \leftarrow a$	Assign a to x
$x \xrightarrow{f} y$	Map x onto y by f

where N_i and X_i are the population and throughput of the i-th chain, M_i can be Z, D, or c^2 that is defined in Table I.

Fig. 5 shows a special case that we need to consider. The task T has K entries and they receive requests from the upper layer. In this case, we model the processor-submodel by a closed network where T switches to $E_1, E_2, \ldots, E_K$ with probabilities $p_1, p_2, \ldots, p_K$. The probability is determined by the throughput of upper layer, that is $p_k = X_k^{upper} / X_{total}^{upper}$. As the Gamma-MDN method requires constant multiclass populations, we need to apply the following transformation to the model

$$N_k = \left(\frac{X_{E_k}}{X_{total}} \right) N_T + N_{E_k} \tag{13}$$

where N_k is the population of the k-th chain, X_{total}, X_{E_k}, N_T and N_{E_k} are the mean performance metrics obtained at the beginning of Algorithm 1.

VI. EVALUATION

In our approach, the key to estimate response times in a layered system is to obtain response time distributions in each closed queueing submodel. Therefore, we first evaluate if Gamma-MDNs are capable of providing good predictions for unknown response time distributions. We generate training

Algorithm 1

Input: A LQN model
Output: $\mathcal{D}_e$ for each e

construct $\mathcal{P}$ and $\mathcal{T}$, create empty $\mathcal{D}_e$ for every e
derive the parameters of chains for each s
iterate to solve LQN, obtain the mean performance metrics per chain in s and then per class in each chain
for each $s \in \mathcal{P}$ **do**
  $Z = Z_T(t) + Z_C(t)$
  create a set $\mathcal{C}$ such that $|\mathcal{C}| = |\varepsilon(t)|$, and determine the population of each c by (13)
  for each $e \in \varepsilon(t)$ **do**
    convolve $\mathcal{D}(e)$ by repeated execution of (2)
    extract mean and SCV from the convolution result by (1), build the input $\mathbf{x}$
  end for
  for each $c \in \mathcal{C}$ **do**
    if $|\mathcal{C}| > 1$ **then**
      aggregate the chains by (12)
      two-class input $\mathbf{x}$ with $\Psi(\text{PS}) \overset{MDN^{\text{IV}}}{\longrightarrow} \mathcal{W}$
    else
      single class input $\mathbf{x}$ with $\Psi(\text{PS}) \overset{MDN^{\text{II}}}{\longrightarrow} \mathcal{W}$
    end if
    extract mean and SCV from $\mathcal{W}$ by (9)
    $\mathcal{W} \leftarrow fitPH(\text{mean}, \text{SCV})$
    put $\mathcal{W}$ into $\mathcal{D}_e$
  end for
end for

Algorithm 2

Input: Output of Algorithm 1
Output: $\mathcal{R}_e$ of LQN

$Done \leftarrow$ empty set
while $|Done| < |\mathcal{T}|$ **do**
  for each $s \in \mathcal{T} \setminus Done$ **do**
    if $s \in \mathcal{B}$ **then**
      obtain the set $\mathcal{C}$
      for each $c \in \mathcal{C}$ **do**
        $Z = Z_T(t) + Z_C(t) + Z_A(t)$
        calculate mean and SCV of Δ by (10), (11), build the input $\mathbf{x}$
        if $|\mathcal{C}| > 1$ **then**
          aggregate the chains by (12)
          two-class input $\mathbf{x}$ with $\Psi(\text{FCFS}) \overset{MDN^{\text{III}}}{\longrightarrow} \mathcal{W}$
        else
          single class input $\mathbf{x}$ with $\Psi(\text{FCFS}) \overset{MDN^{\text{I}}}{\longrightarrow} \mathcal{W}$
        end if
        implement line 18-20 of Algorithm 1
        if $|\mathcal{D}_e| = |\mathcal{E}_e| + 1$ **then**
          convolve $\mathcal{D}_e$ by repeated execution of (2)
          $\mathcal{R}_e \leftarrow$ convolution result
        end if
      end for
      put s into $Done$ set
    end if
  end for
end while

and test pairs falling within the ranges shown in Table I and employ the tools LINE 2.0.8 [29] with JMT 1.0.5 [30] to simulate the pairs under FCFS and PS discipline respectively. For each pair, we tag a particular job and measure its response time at the queueing station. Because the network is closed, the measurements can be made constantly so that we obtain a steady response time distribution. The single class and two-class training sets consist of 22000 and 14000 pairs and the evaluations are conducted on 4400 and 2800 unseen pairs[1].

We consider two performance measures: *mean absolute percentage error* (MAPE) and *mean KS distance* (MKS). The first measure is defined as

$$\text{MAPE} = \frac{100\%}{N_{test}} \sum_{n=1}^{N_{test}} \left| \frac{P_n - G_n}{G_n} \right| \tag{14}$$

where P and G are the predicted and ground-truth values, N_{test} is the number of test pairs. We calculate MAPE for mean, SCV, and 95th percentile of response time respectively.

KS distance is defined as the maximum vertical distance between two cumulative density functions (CDFs), thus our second measure is

$$\text{MKS} = \frac{1}{N_{test}} \sum_{n=1}^{N_{test}} \max_x \left| F_n^G(x) - F_n^P(x) \right| \tag{15}$$

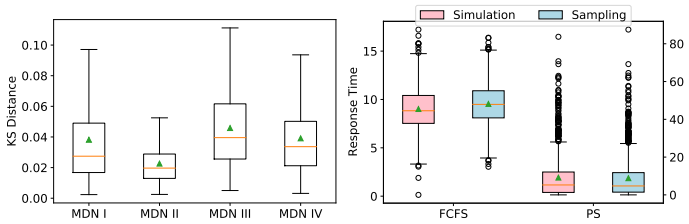
[1]The datasets are available at https://zenodo.org/record/5528034

TABLE III
THE MAPE AND MKS RESULTS OF MDN

Model	Training pairs	Test pairs	MAPE			MKS
			Mean	SCV	95-th	
MDN I	22000	4400	3.7%	5.5%	4.4%	0.038
MDN II	22000	4400	4.9%	6.5%	5.7%	0.022
MDN III	14000	2800	3.3%	7.0%	3.6%	0.045
MDN IV	14000	2800	6.0%	7.6%	5.7%	0.039

where $F^G(x)$ is the empirical CDF from observed samples and $F^P(x)$ is the CDF of a mixture model from prediction.

Each of our MDN models consists of seven Gamma kernels and the evaluation results are shown in Table III. As can be observed, all MAPE values are below 10% and MKS values are less than 0.05, the trained MDNs provide overall good performance. Fig. 6a shows an interesting phenomenon that the vast majority of KS distances are below 0.1, which means the MDNs really learned some relationships and try to get as close to the true response time distributions as they can. Fig. 6b gives a comparison between simulations and sampling distributions from MDN predictions. In this experiment, we simulate the pair and obtain 1169 and 1170 response time samples for cases FCFS and PS. Then we employ MDN I and II to make predictions and generate 1169 and 1170 samples from the returned Gamma mixture distributions. As can be seen from this figure, the MDNs cope well with learning to reproduce unknown distributions. The emulated distribution



(a) (b)

Fig. 6. Visualizing prediction performance with box plots (the orange line and green triangle inside box express the median and mean values). (a): Summary statistics on the KS distance for all test pairs of Table III. (b): Summary statistics on the response time given $\mathbf{x}$ = (85, 8, 0.4, 1.5, 2). The left and right y-axis represent the response time under FCFS and PS respectively.

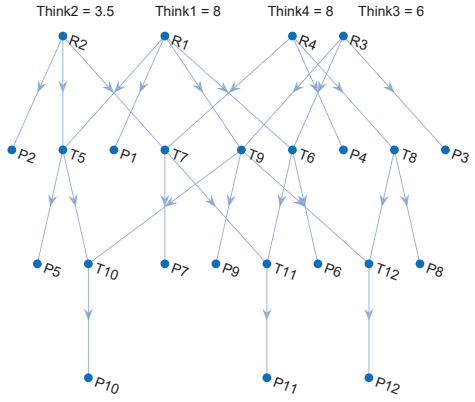


Fig. 7. A larger interconnected layered system.

captures a similar symmetric-like or right-skewed trend as the simulation. The next experiment is to compare mean and variance between true distributions and predictions through varying a set of input parameters [13]. Fig. 8a to 8f show a "what if" instance where we can observe that our MDNs give acceptable approximations for mean and variance over the change of distinct types of parameters. This means MDNs are capable of discriminating variables and calculating how each type influences the response time. Fig. 8g and 8h describe the response time distributions in a submodel having two job classes with different service rates. This is the situation where conventional BCMP theorem cannot be used [2]. We can see that the true distribution of class 1 is very well predicted by the MDN. The prediction for class 2 is worse than class 1 (the SCV percentage error for class 2 is greater than 10%), but the overall trend of prediction is still reasonable.

Then, our algorithms are evaluated on the LQN model in Fig. 1 and a larger interconnected layered system in Fig. 7. The parameters of the second system are shown in Table V. In this layered system, the requests made by reference tasks R1, R2, R3, R4 are stochastic with a mean value 0.5 except for the requests R3→T6, R4→T7, the mean for these two is 1. The remaining requests made by non-reference tasks are deterministic with a value 1. We calculate the end-to-end delays for both two models and compare the results with lqsim and lqns 6.0 [17], a simulation tool and a de facto standard analytic tool for LQN. As shown in Table IV, the response

TABLE IV
PERCENTAGE ERROR OF MEAN AND SCV

Task		T1	T2	R1	R2	R3	R4
Mean	**lqns**	1.073	3.135	1.492	0.485	0.658	0.909
	mdn	1.145	3.221	1.502	0.494	0.667	0.917
	lqsim	1.139	3.241	1.506	0.495	0.663	0.918
SCV	**lqns**	1.321	0.126	0.620	1.370	1.462	0.854
	mdn	1.356	0.747	0.689	1.717	1.838	1.126
	lqsim	1.350	0.702	0.740	1.605	1.852	1.066
PE Mean	**lqns**	5.79%	3.27%	0.93%	2.02%	0.75%	0.98%
	mdn	0.53%	0.62%	0.27%	0.20%	0.60%	0.11%
PE SCV	**lqns**	2.15%	82.0%	16.2%	14.6%	21.0%	19.8%
	mdn	0.44%	6.41%	6.89%	6.98%	0.76%	5.63%

times of entries in reference tasks are well predicted by our approach. The MATLAB implementation of the algorithms takes 0.96s (including 27 predictions and 9 convolutions) on the second model after getting the mean values by AMVA. The percentage errors (PE) for SCV and mean are less than 10% and 1% respectively. This illustrates the effectiveness of our methodology which serves as a potential solution to response time distribution analysis in layered systems.

VII. CONCLUSION

In this paper, we have developed an approach to estimate the distributions of response time in a layered system. The approach copes with a set of carefully built submodels and performs convolution of the response time distributions layer by layer, from bottom to up. In this process, Gamma-MDNs predict the conditional response time distribution for each submodel. They have the promising potential to be used as emulators for complex queueing studies. In the end, the mean and variance of system delay are obtained and we employ fitted PH as the approximated response time distribution. The results give users a detailed QoS description upon single mean values. It also allows us to calculate the probability that a customer will get a response in less than a specific time, which can be used to provide service promises in the future. In future work we plan to extend this approach to include other details of layered systems such as asynchronous calls.

REFERENCES

[1] P. G. Harrison and W. J. Knottenbelt, "Passage time distributions in large markov chains," in *Proceedings of the 2002 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, 2002, pp. 77–85.

[2] F. Baskett, K. M. Chandy, R. R. Muntz, and F. G. Palacios, "Open, closed, and mixed networks of queues with different classes of customers," *Journal of the ACM (JACM)*, vol. 22, no. 2, pp. 248–260, 1975.

[3] P. G. Harrison, "Response time distributions in queueing network models," in *Performance Evaluation of Computer and Communication Systems*. Springer, 1993, pp. 147–164.

[4] L. Zhu, G. Casale, and I. Perez, "Fluid approximation of closed queueing networks with discriminatory processor sharing," *Performance Evaluation*, vol. 139, p. 102094, 2020.

[5] T. G. Kurtz, "Solutions of ordinary differential equations as limits of pure jump markov processes," *Journal of applied Probability*, vol. 7, no. 1, pp. 49–58, 1970.

[6] R. G. Franks, "Performance analysis of distributed server systems." Ph.D. dissertation, Carleton University, 2000.

[7] G. Franks, T. Al-Omari, M. Woodside, O. Das, and S. Derisavi, "Enhanced modeling and solution of layered queueing networks," *IEEE Transactions on Software Engineering*, vol. 35, no. 2, pp. 148–161, 2008.

[8] C. M. Bishop, "Mixture density networks," 1994.

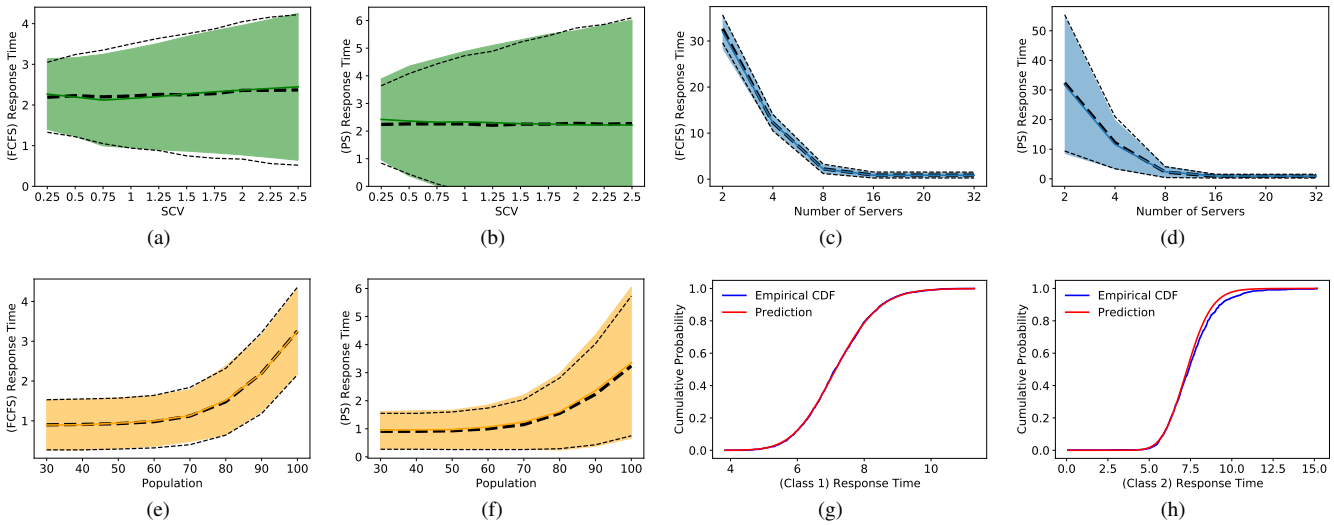


Fig. 8. (a) to (f): Starting with $\mathbf{x}$ = (90, 8, 0.9, 0.5, 8) and changing SCV, number of servers, population over a range of values, comparing the mean (simulation = heavy dashed line, prediction = colored line) and variance (simulation = light dashed line, prediction = colored region) between the true distributions and predictions. (g) (h): Response time distribution functions of two job classes in the submodel $\mathbf{x}$ = (40, 3.5, 0.6, 0.5, 55, 2, 1, 1, 8) with FCFS service discipline.

TABLE V
PARAMETERS OF THE LAYERED SYSTEM IN FIG. 7

	R1	R2	R3	R4	T5	T6	T7	T8	T9	T10	T11	T12
Multiplicity (Task)	30	55	70	40	10	15	10	20	30	25	25	28
Multiplicity (Processor)	16	16	4	4	8	8	8	16	16	20	20	28
Host Demand Mean	0.83	0.12	0.05	0.29	0.05	0.17	0.26	0.33	0.15	0.34	0.1	0.2

[9] C. Li and G. H. Lee, "Generating multiple hypotheses for 3d human pose estimation with mixture density network," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 9887–9895.

[10] X. Wang, S. Takaki, and J. Yamagishi, "An autoregressive recurrent mixture density network for parametric speech synthesis," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2017, pp. 4895–4899.

[11] L. Bazzani, H. Larochelle, and L. Torresani, "Recurrent mixture density network for spatiotemporal visual attention," *arXiv preprint arXiv:1603.08199*, 2016.

[12] A. Brando Guillaumes, "Mixture density networks for distribution and uncertainty estimation," Master's thesis, Universitat Politècnica de Catalunya, 2017.

[13] C. N. Davis, T. D. Hollingsworth, Q. Caudron, and M. A. Irvine, "The use of mixture density networks in the emulation of complex epidemiological individual-based models," *PLoS computational biology*, vol. 16, no. 3, p. e1006869, 2020.

[14] M. Raeis, A. Tizghadam, and A. Leon-Garcia, "Predicting distributions of waiting times in customer service systems using mixture density networks," in *2019 15th International Conference on Network and Service Management (CNSM)*. IEEE, 2019, pp. 1–6.

[15] M. Raeis, A. Tizghadam, and A. Leon-Garcia, "Probabilistic bounds on the end-to-end delay of service function chains using deep mdn," in *2020 IEEE 31st Annual International Symposium on Personal, Indoor and Mobile Radio Communications*. IEEE, 2020, pp. 1–6.

[16] M. Woodside and G. Franks, "Tutorial introduction to layered modeling of software performance," *Carleton University, Ottawa, Canada*, 2002.

[17] G. Franks, P. Maly, M. Woodside, D. C. Petriu, A. Hubbard, and M. Mroz, "Layered queueing network solver and simulator user manual," *Dept. of Systems and Computer Engineering, Carleton University (December 2005)*, pp. 15–69, 2005.

[18] K. M. Chandy and D. Neuse, "Linearizer: A heuristic algorithm for queueing network models of computing systems," *Communications of the ACM*, vol. 25, no. 2, pp. 126–134, 1982.

[19] M. Woodside, J. E. Neilson, D. C. Petriu, and S. Majumdar, "The stochastic rendezvous network model for performance of synchronous client-server-like distributed software," *IEEE Transactions on Computers*, vol. 44, no. 1, pp. 20–34, 1995.

[20] J. A. Rolia and K. C. Sevcik, "The method of layers," *IEEE transactions on software engineering*, vol. 21, no. 8, pp. 689–700, 1995.

[21] S. C. Bruell, G. Balbo *et al.*, *Computational algorithms for closed queueing networks*. Elsevier North-Holland, 1980.

[22] M. Reiser and H. Kobayashi, "Queuing networks with multiple closed chains: theory and computational algorithms," *IBM journal of Research and Development*, vol. 19, no. 3, pp. 283–294, 1975.

[23] M. Reiser and S. S. Lavenberg, "Mean-value analysis of closed multichain queuing networks," *Journal of the ACM (JACM)*, vol. 27, no. 2, pp. 313–322, 1980.

[24] G. Bolch, S. Greiner, H. De Meer, and K. S. Trivedi, *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications*. John Wiley & Sons, 2006.

[25] M. F. Neuts, *Matrix-geometric solutions in stochastic models: an algorithmic approach*. Courier Corporation, 1994.

[26] W. J. Stewart, *Probability, Markov chains, queues, and simulation: the mathematical basis of performance modeling*. Princeton university press, 2009.

[27] R. Marie, "Calculating equilibrium probabilities for λ (n)/ck/1/n queues," in *Proceedings of the 1980 international symposium on computer performance modelling, measurement and evaluation*, 1980, pp. 117–125.

[28] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *arXiv preprint arXiv:1511.07289*, 2015.

[29] G. Casale, "Integrated performance evaluation of extended queueing network models with LINE," in *2020 Winter Simulation Conference (WSC)*. IEEE, 2020, pp. 2377–2388.

[30] M. Bertoli, G. Casale, and G. Serazzi, "JMT: performance engineering tools for system modeling," *ACM SIGMETRICS Performance Evaluation Review*, vol. 36, no. 4, pp. 10–15, 2009.