Proceedings of the 2003 Winter Simulation Conference
S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, eds.

QUEUEING-NETWORK STABILITY: SIMULATION-BASED CHECKING

Jamie R. Wieland
Raghu Pasupathy
Bruce W. Schmeiser

School of Industrial Engineering
Purdue University
West Lafayette, IN 47907, U.S.A.

ABSTRACT

Queueing networks are either stable or unstable, with stable networks having finite performance measures and unstable networks having asymptotically many customers as time goes to infinity. Stochastic simulation methods for estimating steady-state performance measures often assume that the network is stable. Here, we discuss the problem of checking whether a given network is stable when the stability-checking algorithm is allowed only to view arrivals and departures from the network.

1 THE STABILITY-CHECKING PROBLEM

Given only simulation code of a queueing-network model, we consider the problem of developing a statistical algorithm to check whether the network is *stable*. Informally, stable means that the network behavior does not explode in the limit as time goes to infinity. Note that this problem is about long-run network equilibrium, not initial-transient bias of point estimators.

1.1 Motivation

The need for checking network stability using simulation arises in two contexts. In the first context, the network is being simulated to estimate time-average performance measures. In the second context, the network is being probabilistically analyzed to determine whether stability exists, and simulation is used as an empirical guide to obtain insight.

In the first context, simulation practitioners often assume that the network is stable. Here, the purpose of a stability-checking algorithm is to protect the practitioner when stability is not present. When a simulation run unexpectedly ends with an 'out-of-memory' error message, the practitioner needs to decide whether to provide more memory—perhaps the computational requirements for the simulation run were underestimated—or to conclude that the network is not as intended and is in fact unstable—possibly due to coding error or parameter misspecification. When a long-run simulation ends without error, a stability check could alert the practitioner if the network appears to be unstable.

In the second context, probability researchers have been active during the last decade in studying necessary and sufficient conditions for network stability. This research has flourished since the discovery that a network can be unstable even though every station has instantaneous arrival rate less than instantaneous service rate (traffic intensity less than one). Having a simulation-based algorithm would improve the efficiency of analyzing networks, because simulation experimentation can provide empirical insight and conjectures.

Dai and Meyn (1995), Banks and Dai (1997), and Sharifnia (1997) simulate networks to develop conjectures about the stability of multiclass queueing networks, but they do not develop statistical decision rules for classifying networks as stable or unstable. Banks and Dai plot the expected customer time in the network and show that it is increasing linearly over time. Sharifnia plots the average number of jobs in the network, and also shows that it increases linearly over time. Dai and Meyn plot the average queue lengths at each station and show that they oscillate with increasing magnitude.

1.2 Organization

This paper is organized as follows. In this section, terms and notation are defined, followed by five criteria for comparing stability-checking algorithms and a discussion about the definition of network stability. Section 2 covers issues that make determining stability difficult. Section 3 contains three properties that are fundamental to all queueing networks and therefore might form the basis for a stability-checking algorithm. Section 4 compares simulation and mathematical

analysis for checking stability. Section 5 contains some thoughts about algorithm design. Section 6 presents an example stability-checking algorithm based on batching data from a long-run simulation experiment. Section 7 contains conclusions and thoughts about future research.

1.3 Terms and Notation

A queueing-network model has, by definition, customers arriving and departing across a boundary that separates the network from the outside world. For every non-negative time t, let $A(t)$ denote the number of customer arrivals, $D(t)$ denote the number of customer departures. From these fundamental counting processes, and the initial number of customers $N(0)$, at any time t the number of customers in the network

$$N(t) = N(0) + A(t) - D(t)$$

can be computed, as can the utilization $U(t) = I(N(t) > 0)$, where I is the indicator function.

We assume that, from the given simulation code, there is available a realization of these output-data processes for any interval of time $[0, \tau]$, where τ is the simulation run length. Based on this one realization, our problem is to check whether the network is stable.

Several stability-checking problem instances can co-exist in a single network model and a single simulation run. All of these output-data processes can be indexed by customer type and network boundary. In this case, each choice of customer and each choice of network boundary leads to a different problem instance. For example, the number of priority customers might be stable despite the number of other customers being unstable. Similarly, the first server might be stable despite other servers being unstable. We write as if there is only one problem instance; nevertheless, having the simulation collect output data for each problem instance allows individual, although dependent, conclusions for each instance.

1.4 Criteria for Comparing Algorithms

Given the simulation code for a network model, from which output-data processes $A(t)$, $D(t)$, $N(t)$, and $U(t)$ can be obtained for the time interval $[0, \tau]$, we wish to develop algorithm(s) to check whether the network is stable. The fundamental output of the algorithm is binary; we arbitrarily say the output is $C = 0$ if the algorithm chooses stable and $C = 1$ if the algorithm chooses unstable.

We consider the following five criteria for comparing stability-checking algorithms.

1. Applicable to many network models. Ideally, the algorithm performs well regardless of network specifics.
2. Easy (or no) algorithm tuning; that is, little (or no) human effort.
3. Fast to compute. Simulation requires $O(\tau)$ computing. A stability-checking algorithm that is slower than $O(\tau)$ takes additional computing time that would be better used to produce additional simulation output data.
4. High probability of correctly classifying (PCC) the network; that is, $P(C = 0)$ is high for a stable model and $P(C = 1)$ is high for an unstable model. PCC depends upon both the given model and run length τ. Whether the algorithm is correct for a particular practitioner is dependent upon the random-number seed chosen by the practitioner, but unstable models that are close to being stable (and vice versa) will require a long run length τ. For a model that is known to be stable, an algorithm developer can use Monte Carlo simulation estimate PCC with

$$\overline{\text{PCC}} = \frac{\sum_{j=1}^{m} C_j}{m},$$

where C_j is the value returned by the stability-checking algorithm for the jth simulated practitioner. Similarly, for an unstable model

$$\overline{\text{PCC}} = \frac{\sum_{j=1}^{m} (1 - C_j)}{m}.$$

5. The final criterion is the ability to provide a useful confidence statement to the practitioner. One approach is to report $\widehat{\text{PCC}}$, an estimated value of PCC for the given model and output-data realization. An algorithm developer can assess the quality of this confidence statement with the mean squared error (mse)

$$\text{E}[(\widehat{\text{PCC}} - \text{PCC})^2] = \text{Var}[\widehat{\text{PCC}}] + \text{bias}^2[\widehat{\text{PCC}}, \text{PCC}],$$

which can be estimated by Monte Carlo simulation with

$$\widehat{\text{mse}} = \frac{\sum_{j=1}^{m} (\widehat{\text{PCC}}_j - \overline{\text{PCC}})^2}{m}.$$

We discuss these criteria in terms of a class of trivial, and unreasonable, coin-flipping algorithms, $\mathcal{A}_{\mathcal{C}}(\alpha)$. The algorithm flips a coin that has probability $1 - \alpha$ of head and α of tail and classifies the given network as stable if head appears and unstable if tail appears. If $\alpha = 0$ this always-stable algorithm concludes that the network is stable, regardless of the given network and realization. If $\alpha = 1/2$, this fair-coin algorithm concludes that the network is stable half the time and unstable half the time, again independent of the network and realization. The always-stable algorithm

applies to any model, requires no tuning, and is fast, so the first three criteria are satisfied. (If the fair-coin algorithm is viewed as a special case of the class of coin-flipping algorithms, the value of α needs to be determined; this process of determining the value is algorithm tuning.)

Criterion 4 is more interesting. The fair-coin algorithm $\mathcal{A}_{\mathcal{C}}(1/2)$ has PCC $= 1/2$ for all models and all realizations, regardless of run length. The always-stable algorithm $\mathcal{A}_{\mathcal{C}}(0)$ has PCC $= 1$ for every stable model and PCC $= 0$ for every unstable model, regardless of the model and realization. Figure 1 shows PCC for two algorithms—the fair-coin algorithm $\mathcal{A}_{\mathcal{C}}(1/2)$ and $\mathcal{A}_{\mathcal{C}}(0.05)$—as a function of some change in the model parameter such as the service rate in a time-homogeneous single-server queue. Stable networks are on the left and unstable are on the right. The $\mathcal{A}_{\mathcal{C}}(0.05)$ values of PCC are 0.95 for stable models and 0.05 for unstable models. The fair-coin algorithm $\mathcal{A}_{\mathcal{C}}(1/2)$, which is not reasonable, demonstrates that a reasonable algorithm has PCC $> 1/2$ for most models. (A reasonable algorithm does not, however, need PCC $> 1/2$ for every model, because the cost of classifying an unstable model as stable might be quite different from classifying a stable model as unstable.) In general, the higher the PCC curve, the better the algorithm.

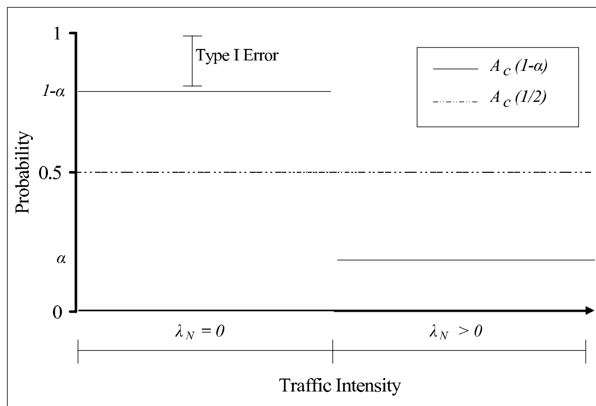

Figure 1: PCC Curve for Coin Flip Algorithm $\mathcal{A}_{\mathcal{C}}$

Criterion 5 is not as fundamental as Criterion 4, but good statistical algorithms provide confidence statements. Here the fair-coin algorithm is better than the always-stable algorithm: $\mathcal{A}_{\mathcal{C}}$ can report, exactly, its PCC value, while $\mathcal{A}_{\mathcal{S}}$ does not know whether its PCC value is zero or one, since it does not know whether the given network is stable or unstable.

1.5 Defining Network Stability

When comparing stability-checking algorithms, *stability* must be defined precisely. With such a definition, a stability-checking algorithm's PCC is defined for a given model and run length τ.

Various definitions have been used, based on both sample-path behavior and moment behavior. For time-homogeneous networks, Dai (1996) has the number of customers diverging to infinity with probability one as the definition of an unstable network. An example definition for a stable network, used by Dai and Jennings (2003), is the long-run instantaneous input rate of the network being equalled by the long-run instantaneous output rate.

As a practical matter, a model that is stable under one definition is likely to be stable under another definition, but for many pairs of stability definitions there is a set of models for which the classification differs. Simulation practitioners will seldom want to choose their own stability definition, while probabilists may need to be more careful.

The definition of stability requires some additional consideration when the network input parameters are seasonal.

Let $\gamma(t)$ denote the queueing network input parameters (e.g., arrival rates, service rates), which is a vector of functions defined for all non-negative times t. The parameters are *time homogeneous* if $\gamma(t)$ is constant over time. The parameters are *seasonal* if there is a constant time δ so that $\gamma(t) = \gamma(t + k\delta)$ for every non-negative integer k. The parameters are *time-heterogeneous* if $\gamma(t)$ is allowed to be any arbitrary function of t.

We do not consider models with time-heterogeneous input parameters, because if the model parameters $\gamma(t)$ can change arbitrarily after time τ, then no finite run length τ is sufficient to reach a conclusion about network behavior as time goes to infinity. The data observed during $[0, \tau]$ must be representative of network behavior after time τ. Ideally the process observed during $[0, \tau]$ is identical to the process after τ. The initial-transient issue is a typical reason why $[0, \tau]$ is only representative rather than identical.

To consider stability of seasonal models a time-averaged definition is reasonable, since the number of customers might not explode and yet

$$\lim_{\tau \to \infty} \mathrm{E}(N(\tau))$$

does not exist. As a working definition, we define stability based on the time-averaged expected number of customers. That is, a network is *stable* if

$$\lim_{\tau \to \infty} \int_0^{\tau} \frac{\mathrm{E}(N(t))}{\tau}\, dt$$

is a finite constant, say θ. We use this definition because of our simulation orientation. A long-run simulation experiment of length τ naturally estimates the performance measure θ with the point estimator

$$\hat{\theta} = \int_0^{\tau} \frac{N(t)}{\tau}\, dt.$$

If the value of the performance measure θ is not finite, then trying to estimate θ using simulation makes no sense, since simulation point estimates are always finite.

Alternatively, and probably with little change in any resulting algorithm, we could define stability as requiring that $\hat{\theta}$ goes to a finite θ with probability one. With either definition, the observed value of $\hat{\theta}$ will play a central role in any reasonable algorithm.

Leibniz's Rule applied to θ shows that any seasonal model that is stable under the time-average expected-number-of-customers definition satisfies $\lambda_N = 0$, where

$$\lambda_N = \lim_{\tau \to \infty} \frac{\mathrm{E}(N(\tau))}{\tau}$$

is the time-averaged rate of change of the expected number of customers. For seasonal models, this condition is equivalent to $\lambda_A = \lambda_D$, where

$$\lambda_A = \lim_{\tau \to \infty} \frac{\mathrm{E}(A(\tau))}{\tau}$$

and

$$\lambda_D = \lim_{\tau \to \infty} \frac{\mathrm{E}(D(\tau))}{\tau}.$$

Notice that for stable seasonal models $\lambda_A = \lambda_D$, despite the instantaneous arrival rate not being equal to the instantaneous departure rate. In general, the choice of stability definition for seasonal models needs to depend upon time averages rather than instantaneous rates.

2 DIFFICULTY IN ESTABLISHING STABILITY

Until the past decade, it was thought that having traffic intensity less than one at every network station (we refer to this as the *usual* traffic condition) was sufficient for establishing stability of queueing networks, where traffic intensity is the ratio of the effective arrival rate to the service rate at a given station. This condition does hold for single-class networks and single-server multiclass networks, but several counterexamples have been presented that show that the usual traffic condition is not sufficient for establishing stability in multi-server multiclass queueing networks. Some counterexamples can be found in, for example, Kumar and Seidman (1990), Rybko and Stolyar (1993), Seidman (1994), and Bramson (1994a, b).

These counterexamples illustrate that stability is affected by factors other than the traffic intensity. These network factors include network routing, scheduling policy, differences in service rates between classes at the same server, and dependence among arrival, service, and routing processes.

A common element in many of these counterexamples is inefficient use of resources. For example, Banks and Dai (1997) simulate the Bramson (1994a) network, which is a multiclass two-station network with re-entrance. This network has the traffic intensity of 0.90, scheduling policy FIFO, and the respective station utilizations of approximately 0.76 and 0.85. These low utilizations are unexpected because the scheduling policy is non-idling and the number of customers in the network is consistently growing with time (run length is 100,000 customer departures).

From a simulation perspective, we take a slightly different view on counterexamples (such as the Bramson network) that are unstable despite the fact that the traffic intensity is less than one at each station. Rather than saying that the usual traffic condition is not sufficient for stability, say that the effective arrival rate is affected by the network factors. Although calculating analytically the effective arrival rate might be difficult, simulation algorithms need only to estimate the effective arrival rate. With this view, the usual traffic condition for stability still holds.

The implications of these counterexamples are as follows: (1) If the network factors are fundamental in causing instability, the most efficient way to stabilize the network may not be obvious. In fact, adding service capacity may not be necessary to achieve stability. (See, for example, Kumar and Seidman (1990) or Sharifnia (1997)). This result is also reinforced by Dai (1996) who shows that increasing the service rate for a given class of customers does not necessarily stabilize a multiclass network because the global stability region is not monotone with respect to the service time vector. (2) Utilization is not necessarily representative of stability because unstable networks can have 'bottleneck' servers with low utilization. The problem, however, does not result from too little service capacity, but rather the network factors. (3) When scheduling priority customers, instability can result if the first customer class is always prioritized over the second class with no corrective measures in place for situations where the second class of customers consistently accumulates for long periods of time without receiving service.

3 WHAT IS FUNDAMENTAL?

The following three queueing-network properties are fundamental in assessing whether a network is stable, at least if stability is based on time-average number of customers.

3.1 Subnetworks

A network is stable if all, say $s < \infty$, stations within the network are stable. If $N_i(t)$ is the number of customers in queue i at time t, then $N(t) = \sum_{i=0}^{s} N_i(t)$. Therefore, if any N_i is unstable, then N is unstable. The converse is also true, in that if N is unstable, then one or more individual stations are unstable.

3.2 Little's Law

Stability-checking algorithms can be based on either number of customers or customer times in the network. Little's Law is usually written as $L = \lambda W$ for a time-homogeneous network, with λ being the instantaneous arrival and departure rate for a stable network. The time-average analogy, which includes seasonal models, is

$$L = \lim_{\tau \to \infty} \int_0^{\tau} \frac{\mathrm{E}(N(t))}{\tau}\, dt$$

and therefore

$$W = \frac{L}{\lambda_A}$$

defines a time-averaged mean time in the network. As always with Little's Law, the result can be applied to the network model as a whole or to any part. If, however, time-homogeneous data are collected from a seasonal network model every δ time units, then collecting number of customers is well defined, whereas collecting times in the network is not.

3.3 Linear Growth

For seasonal and time-homogeneous networks with

$$\lim_{\tau \to \infty} \frac{\mathrm{E}(A(\tau))}{\tau} = \lambda_A$$

and

$$\lim_{\tau \to \infty} \frac{\mathrm{E}(D(\tau))}{\tau} = \lambda_D$$

the expected number of customers has linear growth rate with asymptotic slope

$$\lambda_N = \lambda_A - \lambda_D.$$

This result holds for both stable and unstable networks. If the network is stable, $\lambda_N = 0$. If the network is unstable, $\lambda_N > 0$. Therefore, a stability-checking algorithm could base its answer on an estimate of λ_N. From Little's Law, the expected time in the network also grows linearly.

4 DISADVANTAGES OF SIMULATION FOR CHECKING STABILITY

Both the simulation and analysis approaches are useful for determining network stability. The analytical approach considers a class of network models, answering with certainty whether the class is stable, or providing no answer because the network class is intractable. The simulation approach considers a particular network, always answering, but with some associated sampling-error uncertainty. Compared to probability analysis, using simulation to check stability has the advantages that it applies to any queueing network, it always provides an answer, and the practitioner needs only to provide a simulation code. There are, of course, disadvantages.

For three reasons, a simulation-based algorithm can be wrong, concluding that the given network is stable when it is unstable, and vice versa. First, the initial transient can make the data collected from $[0, \tau]$ appear unstable, especially when $N(0)$ is much smaller than the expected number of customers. Second, data from heavily loaded stable networks have high autocorrelations, so the run length τ must be quite large to estimate long-run performance measures. Third, sample paths for a heavily loaded stable network and and a barely unstable network are quite similar; no discontinuity occurs, for example, in M/M/1 models as traffic intensity increases from less than one to greater than one.

Another disadvantage is that, because a simulation-based algorithm is correct only some fraction of the time, a confidence statement is required. Such statements often are true only asymptotically and, even if the assumptions hold, are easily misunderstood by the practitioner.

Finally, simulation seems to be restricted to stability definitions based on rates and moments. Stability definitions based on asymptotic almost-sure bounds on sample paths are tractable only with mathematical analysis. On the other hand, the first-order stability definition, based on average number of customers directly can be extended to

$$\theta_k = \lim_{\tau \to \infty} \int_0^{\tau} \frac{\mathrm{E}(N^k(t))}{\tau}\, dt,$$

for positive integers k, allowing stability to be based (for example) on the variance of customer numbers.

5 ALGORITHM CONSIDERATIONS

The stability-checking problem assumes that a simulation code is given for the network model of interest. To obtain a particular stability-checking problem instance, we need to have three additional types of information about the problem context.

First, is a definition of stability to be specified? A reasonable default definition is that the time-averaged number in the network has finite limit θ.

Second, is the run length τ fixed? For a simulation practitioner whose experiment has ended with an 'out-of-memory' error message, τ is fixed. For a researcher who is investigating stability issues, τ can be chosen, with the option of increasing the run length to obtain better algorithm performance.

Third, is the length of the network seasonality, δ, known to the algorithm? If the network is time-homogeneous, does the algorithm know? Asymptotically, many algorithms will

perform the same with or without seasonality knowledge, but for short run lengths τ, seasonality knowledge should help an algorithm.

Now, given a problem instance, a specific stability-checking algorithm can be designed. There are at least three high-level design decisions to be made.

First, the algorithm can take either a Bayesian or frequentist framework. Ideally, the algorithm should be able to reflect the prior probabilities of whether the network is stable or unstable, which favors a Bayesian approach. Additionally, the algorithm should be able to reflect the costs of incorrect classifications, both concluding stable for an unstable network and vice versa, which either framework can do.

Second, the algorithm can be based on either number in the network or times in the network (or possibly both). Law (1975) shows that direct estimation of L or of W is less efficient than estimating mean queueing time, and then indirectly estimating L and W. Law considered only single-station networks, but the same statistical advantage can be obtained in general networks by estimating mean time in the network by substituting mean service time for observed service time. If the assumption that the simulation code is given is taken literally, however, so that only the processes A, D, and N are observed, indirect estimation is not possible.

Third, the algorithm can use data from the entire network or data from individual stations. If an s-station network has only one station that is unstable, then the instability signal from that station alone will be easier to detect than from the aggregate network.

6 A STABILITY-CHECKING ALGORITHM

To illustrate the key issues in designing a simulation-based stability-checking algorithm, we present an example frequentist algorithm, $\mathcal{A}_{\mathcal{B}}$. This algorithm is based on estimating the time-average network growth rate, $\lambda_N = E(N(\tau))/\tau$, with batching used to estimate sampling error. The algorithm is based on classical hypothesis testing, with null hypothesis H_0: $\lambda_N = 0$ and alternative hypothesis H_1: $\lambda_N > 0$. That is, H_0 is that the network is stable and H_1 is that the network is unstable. This algorithm is presented with no computational results or claim of particularly good performance; rather it is to serve as a basis for discussion of algorithmic issues in Section 6.2.

6.1 Algorithm $\mathcal{A}_{\mathcal{B}}$

Given: Observed output data $N(t)$ for $0 \le t \le \tau$.

Step 0. Choose a number of batches, b, with a default value of $b = 10$. Choose a nominal probability of type I error, with a default value of $\alpha = 0.05$.

Step 1. For each batch $j = 2, \ldots, b$, compute the batch observation

$$\hat{\lambda}_{N,j} = \int_{(j-1)\tau/b}^{j\tau/b} \frac{N(t)}{\tau/b}\, dt,$$

the time-average number in the network during the jth batch.

Step 2. Compute the difference between the last and second batch observations:

$$\hat{\lambda}_N = \hat{\lambda}_{N,b} - \hat{\lambda}_{N,2}$$

Step 3. Compute the variance of the batch observations:

$$s^2 = \frac{\sum_{j=2}^{b} \hat{\lambda}_{N,j}^2 - b\hat{\lambda}_N^2}{b-2}.$$

Step 4. Conclude unstable if and only if H_0 is rejected; that is, reject whenever

$$\frac{\hat{\lambda}_N}{\sqrt{2}s} > t_{1-\alpha,b-2},$$

where $t_{1-\alpha,b-2}$ is the $1-\alpha$ Student-T quantile with $b-2$ degrees of freedom. (For the default values of $b = 10$ and $\alpha = 0.05$, $t_{1-\alpha,b-2} = 1.86$ and stability is rejected if $\hat{\lambda}_N > 2.63s$.)

6.2 Critique of Algorithm $\mathcal{A}_{\mathcal{B}}$

How does Algorithm $\mathcal{A}_{\mathcal{B}}$ fare in terms of the five criteria for comparing algorithms discussed in Section 1.4? First, it is general and applicable to any seasonal (including time-homogeneous) network. Second, the number of batches, b, and the probability of type I error, α, are 'magic' parameters, with values that the practitioner should not be asked to choose; the default values of $b = 10$ and $\alpha = 0.05$ are not surprising, but other values could well be better. Third, the algorithm is fast and easy to implement. Fourth, the nominal value of PCC is $1-\alpha$ if the network is stable and unknown if the network is unstable. Fifth, the p value (the probability that a T value is greater than the observed $\hat{\lambda}_N/(s/\sqrt{b})$) is often taken as a kind of confidence statement.

Algorithm $\mathcal{A}_{\mathcal{B}}$ leaves much to be desired in terms of Criterion 4. For a fixed run length τ, Figure 2 shows PCC as a function of the network model; think of the horizontal axis as being the traffic intensity of an $M/M/1$ model in this example. The figure's curves assume that the underlying assumptions all hold: that is, that the initial transient is negligible, the batch values $\hat{\lambda}_{N,j}$ are essentially normally distributed and independent. These assumptions, if true, would provide PCC $= 1-\alpha$ for any stable network, which

is traffic intensity $\rho < 1$ for this M/M/1 example. Because there is no discontinuity of sample-path behavior at $\rho = 1$, there is a discontinuity in PCC, with suddenly PCC $= \alpha$. The value of PCC then increases monotonically to one for networks that are more and more unstable. Even here with all assumptions being true, the effect of increasing the run length τ is only to raise the PCC curve for values of $\rho > 1$; the value for stable models can be raised only by lowering the values for unstable networks.

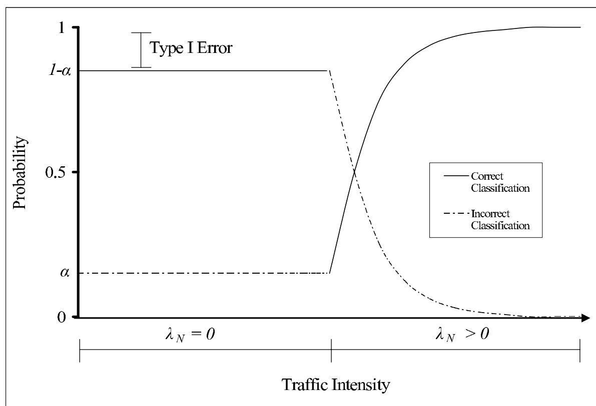

Figure 2: Example PCC Curve for Algorithm $\mathcal{A}_{\mathcal{B}}$ under Assumptions

The assumptions, of course, never hold exactly. There is an initial transient, which tends to look like instability. For this reason, $\mathcal{A}_{\mathcal{B}}$ uses the second batch rather than the first in Step 2. For stable networks close to instability, large autocorrelations cause dependence between the batch values $\hat{\lambda}_{N,j}$, causing the standard-error estimate $s/\sqrt{b}$ to be too small for any given run length τ and number of batches b. For asymptotically large batches, however, the batch observations $\hat{\lambda}_{N,j}$ are normal and independent for both stable and unstable networks. Under H_0, the Step-2 point estimator,

$$\hat{\lambda}_N = \hat{\lambda}_{N,b} - \hat{\lambda}_{N,2},$$

is asymptotically normal with mean zero.

Our use of the time-average batch observations $\hat{\lambda}_{N,j}$ is important to obtaining the appropriate asymptotic performance. Initially, our batch observations in Step 1 were the simpler

$$\tilde{\lambda}_{N,j} = \frac{N(j\tau/b) - N((j-1)\tau/b))}{\tau/b},$$

the rate of change in $N(t)$ during the jth batch. Then corresponding point estimator of λ_N, the average of the batch observations, is

$$\tilde{\lambda}_N = \frac{N(\tau)}{\tau}.$$

Because this point estimator is based only on a snapshot of the number of customers at time τ, rather than a time average, it is not asymptotically normal, despite being the mean of batch observations in Step 2. To provide a specific example, assume that the model is $M/M/1$ with traffic intensity ρ. Then at steady state, $N(t)$ has a geometric distribution with mean $\mathrm{E}(N(t)) = \rho/(1-\rho)$ and variance $\mathrm{Var}(N(t)) = \rho/(1-\rho)^2$. Therefore, the point estimator $\tilde{\lambda}_N$ is not asymptotically normal for this important special case.

If Algorithm $\mathcal{A}_{\mathcal{B}}$ had been defined using $\tilde{\lambda}_{N,j}$ rather than $\hat{\lambda}_{N,j}$, asymptotic normality could have been achieved by using independent replications (rather than batching) to unlink the $\tilde{\lambda}_{N,j}$ values. The cost, however, would be that each replication would incur the initial transient, a severe disadvantage in an algorithm whose purpose is to check stability.

Criterion 5 is also an issue. In fact, no confidence statement is provided. The p value easily could be computed, but for stable networks with all assumptions holding the distribution of p values is uniform over [0, 1], so a better confidence statement is needed. The confidence interval

$$0 \le \hat{\lambda}_N \le t_{1-\alpha,b-2}\sqrt{2}s$$

has nominal confidence $1-\alpha$, but is only a restatement of the hypothesis-testing and p-value computations. Better would be a confidence statement that is directly about the algorithm's PCC.

7 CONCLUSIONS AND EXTENSIONS FOR FUTURE RESEARCH

Developing an excellent algorithm for checking stability might be quite difficult. Although many given models are quickly seen to be either stable or unstable, stable models that are close to unstable are difficult to simulate because of high autocorrelation in the output data. Furthermore, the discontinuity in the probability of correct classification at the stable-unstable boundary almost guarantees that any algorithm based on hypothesis testing will be wrong more than half the time for some models.

We have focused on seasonal models, with time-homogeneous models as a special case. Even if the model is time homogeneous, simulation experiments estimate steady-state performance measures using time averages. Once time averages are used, a stability-checking algorithm can naturally be extended to seasonal models.

We prefer a Bayesian framework, especially if the prior distribution meaningfully can reflect the practitioner's belief. Additionally, a Bayesian posterior distribution allows a meaningful confidence statement about the algorithm's PCC.

Another research problem related to stability checking is that of estimating the stable-unstable boundary for a class of network models and one model parameter. Here the algorithm would have the ability to simulate whatever networks it chooses, possibly extrapolating to obtain its estimate. In the continuous version, the problem is much like stochastic root finding (e.g., Chen and Schmeiser (2001)), except that the problem is to find the model parameter where some performance measure becomes infinite, rather than a specified finite value. In a discrete version, such an algorithm would help the practitioner to determine the minimum numbers of servers required to serve a fixed load or determine the maximum release rate in a production network. (See, for example, Schruben (1997).)

ACKNOWLEDGMENTS

The first author's research was supported by an Eastman Kodak Fellowship from the Center for Collaborative Manufacturing at Purdue University.

REFERENCES

Banks, J., and Dai, J.G. 1997. Simulation studies of multiclass queueing networks. *IEEE Transactions* 29: 213–219.

Bramson, M. 1994a. Instability of FIFO queueing networks. *The Annals of Applied Probability* 4, 2: 414–431.

Bramson, M. 1994b. Instability of FIFO queueing networks with quick service times. *The Annals of Applied Probability* 4, 3: 693–718.

Chen H. and Schmeiser, B. 2001. Stochastic root finding via retrospective approximation. *IIE Transactions* 33: 259–275 (special issue of *Operations Engineering* honoring Alan Pritsker).

Dai, J.G. 1996. A fluid limit model criterion for instability of multiclass queueing networks. *The Annals of Applied Probability* 6, 3: 751–757.

Dai, J.G., and O.B. Jennings. 2003. Stability of general processing networks. Chapter 7 in *Stochastic Models and Optimization*, 193–243: Springer Series, New York.

Dai, J.G., and Meyn, S.P. 1995. Stability and convergence of moments for multiclass queueing networks via fluid limit models. *IEEE Transactions on Automatic Control* 40, 11: 1889–1903.

Kumar, P.R., and Seidman, T.I. 1990. Dynamic instabilities and stabilization methods in distributed real-time scheduling of manufacturing systems. *IEEE Transactions on Automatic Control* 35, 3: 289–298.

Law, A.M. 1975. Efficient estimators for simulated queueing systems. *Management Science* 22, 1: 30–41.

Rybko, A.N., and Stolyar, A.L. 1993. On the ergodicity of random processes that describe the functioning of open queueing networks. *Problems of Information Transmission* 28: 199–220.

Schruben, L.W. 1997. Simulation optimization using simultaneous replications and event time dilation. *Proceedings of the Winter Simulation Conference*, ed. S. Andradóttir, K.J. Healy, D.H. Withers, and B.L. Nelson, 177–180.

Seidman, T.I. 1994. First come, first served can be unstable!. *IEEE Transactions on Automatic Control* 39, 10: 2166–2171.

Sharifnia, A. 1997. Instability of the join-the-shortest-queue and FCFS policies in queueing systems and their stabilization. *Operations Research* 45, 2: 309–314.

AUTHOR BIOGRAPHIES

JAMIE R. WIELAND is a Master's student in the School of Industrial Engineering at Purdue University. She received a B.S. in Industrial Engineering from Northwestern University in 2001. Her primary research interests are in applied stochastic modeling. Her e-mail address is `<jwieland@purdue.edu>`.

RAGHU PASUPATHY is a Ph.D. student in the School of Industrial Engineering at Purdue University. His dissertation has focused on designing efficient algorithms for the solution of general stochastic root-finding problems. His broad research interests are in the area of stochastic operations research. His email address is `<pasupath@purdue.edu>`.

BRUCE W. SCHMEISER is a professor in the School of Industrial Engineering at Purdue University. His interests lie in applied operations research, with emphasis in stochastic models, especially the probabilistic and statistical aspects of stochastic simulation. He is an active participant in the Winter Simulation Conference, including being Program Chair in 1983 and chairing the Board of Directors from 1988-1990. His email address is `<bruce@purdue.edu>`.