

Continuous-time Hidden Markov Models for Network Performance Evaluation[★]

Wei Wei, Bing Wang, Don Towsley

{*weiwei,bing,towsley*}@cs.umass.edu
Department of Computer Science
University of Massachusetts
Amherst, MA 01003, USA

Abstract

In this paper, we study the use of continuous-time hidden Markov models for network protocol and application performance evaluation. We develop an algorithm to infer the continuous-time hidden Markov model from a series of end-to-end delay and loss observations of probe packets. This model can then be used to simulate network environments for network performance evaluation. We validate the simulation method through a series of experiments both in *ns* and over the Internet. Our experimental results show that this simulation method can represent a wide range of *real* network scenarios. It is easy to use, accurate, and time efficient.

Key words: continuous-time hidden Markov model, EM algorithm, performance evaluation, network simulation

1 Introduction

Simulation is a common approach to evaluate a network protocol or application. An event driven simulator requires the specification of the topology as well as the parameters of every component of the simulated network. The setting of these parameters requires fine-tuning and needs to be representative of a real world scenario, which is a non-trivial task. Furthermore, simulation

[★] This research was supported in part by the National Science Foundation under NSF grants ANI-0085848, ANI-9973092, EIA-0080119 and by DARPA under contract F30602-00-2-0554. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies.

of very large networks can be very difficult due to excessive memory and CPU time requirements. Consequently, there have been considerable efforts to speed up event-driven simulation. For example, [1] provides a means to distribute an *ns* [2] simulation on several connected workstations. The Scalable Simulation Framework (SSF) aims to transparently utilize parallel processor resources and scale to very large collection of simulated entities [3]. Fluid simulation is another speedup technique that makes simplified assumptions about the system and is studied in [4,5]. Compared to a simulator, an emulation package has the advantage of using real traffic generators. Dummynet [6] and NIST Net [7] are two such examples. Although still requiring users to specify some parameters such as propagation delay and loss rate, they provide greater transparency: from the user's point of view, the network is a black box simulated by the emulator. The challenge here is how to make the black box a good model of a real network. For instance, Dummynet uses an independent uniform random loss model, which differs from the correlated loss observations made by several studies [8,9].

In this paper, we propose the use of a continuous-time hidden Markov model (CT-HMM) for network performance evaluation. We infer a CT-HMM from delay and loss observations seen by a sequence of probes sent from one end host to another end host. This CT-HMM can then be incorporated into a simulator or an emulator to drive the simulation of a network protocol or application by providing losses and delays to packets in the network at arbitrary points of time. By collecting probe traces in a wide variety of network settings, one can construct a library of CT-HMMs, with each model representing a particular network setting. A model can then be selected to simulate a network protocol or application under a particular network environment. This simulation method can thus provide users with a simulation environment representing a wide range of *real* network scenarios.

We carry out experiments in *ns* and over the Internet to validate this simulation method. The experiments allow us to validate the method in both controlled and real network environments. We demonstrate that the CT-HMM is a good model of the network settings by showing that the behavior of a flow driven by the model is similar to that of a flow in the original network. Here the flow can be governed by a network protocol or an application. We validate using both TCP and a streaming video application. Our experiments show that this simulation method is accurate, and time and space efficient. For a simulation time of 3000 seconds, the running time using this method is around 2 minutes on a Pentium 4 regardless of the complexity of the network (topology, number of flows, etc.) being simulated. A simulation of a similar three-router scenario in *ns* can take over 30 minutes.

Previous works on network modeling focus on the loss behavior and use discrete-time models [10–12]. Yajnik, etc. use a k -th order Markov model

for the temporal dependence in packet loss [11]. Salamatian and Vaton use a discrete-time HMM to model packet loss in network channels and show that the number of states required is significantly less than using a k -th order Markov model [12]. For the purpose of network protocol or application simulation, the delay characteristics also have to be modeled. Furthermore, it is necessary to assign a delay or a loss to a packet at an arbitrary point of time. This cannot be provided by a discrete time model.

The rest of the paper is organized as follows. The inference of CT-HMM is described in Section 2. Section 3 provides numerical validation of the model. Section 4 describes methodology for the validation. Section 5 and 6 describe the validation of this simulation method in *ns* and over the Internet, focusing on TCP and a streaming video application respectively. Finally Section 7 concludes the paper and describes future work.

2 Inference of the Continuous-Time Hidden Markov Model

In this section, we assume that a network in steady state can be modeled by a CT-HMM and describe how the parameters of such a model can be inferred given a sequence of observations. These observations are losses or delays seen by a sequence of probes sent from one end host to another end host in the network. Because delays can take arbitrary nonnegative real values, we discretize them into a finite set of values. We shall refer to these values along with loss as a set of observation symbols. Let us suppose that the network can be modeled as a CT-HMM with M distinct observation symbols and N hidden states. Here, a hidden state cannot be observed directly but is reflected by the observations.

Let $\{Z(t)\}_{t \geq 0}$ be the CT-HMM that we wish to learn. Each state $Z(t)$ contains two components: the hidden state $X(t) \in \{1, 2, \dots, N\}$ and the observation symbol $Y(t) \in \{1, 2, \dots, M\}$. That is, $Z(t) = (X(t), Y(t))$. Let $S = \{1, 2, \dots, N\} \times \{1, 2, \dots, M\}$. The stochastic process $\{Z(t)\}_{t \geq 0}$ takes values in S and is governed by a single infinitesimal generator of size $MN \times MN$. Denote this infinitesimal generator as $Q = [q_{ij}]$, where $q_{ij} \geq 0$, $q_{ii} = -\sum_{j \neq i} q_{ij}$, $i, j \in S$.

We develop a mechanism to infer the infinitesimal generator Q , given the sequence of discrete-time observations $\{Y_t\}_{t=1}^T$, where T is the length of the sequence. The value of Y_t is either an observation or unknown, and the time difference between two adjacent observations is Δ . Since the observations are at discrete times, we start from a probability transition matrix reflected by the observations. Let $P(\Delta) = [p_{ij}(\Delta)]$ denote the probability transition matrix derived from the continuous-time Markov chain, where $p_{ij}(\Delta)$ is the probabil-

ity that the Markov chain, currently in state i , is in state j after an additional time Δ , where $i, j \in S$ [13]. Without confusion, we let P represent $P(\Delta)$ in the rest of the paper for simplicity. Our first step is to derive the probability transition matrix P from the observation sequence using an Expectation Maximization (EM) algorithm. Our second step is to calculate the infinitesimal generator Q from the probability transition matrix P . We next describe these two steps in detail.

2.1 An EM algorithm to infer the discrete-time model

We consider a discrete-time model derived from the continuous-time Markov chain with the probability transition matrix P and initial distribution π . For convenience, let $\lambda = (P, \pi)$. We next describe the procedure to infer λ from a sequence of T observations.

When the observation sequence is obtained by periodic probing, our algorithm to infer the parameter λ is derived from the EM algorithm in [14] by representing the hidden state and the observation symbol as a state. We refer to this as the complete-data EM algorithm. In some situations, it is impossible to obtain a periodic probing sequence. For instance, if one of the hosts is behind a firewall, we may have to use TCP probes, which cannot be guaranteed to follow a specified probing process. For this case, we assume the existence of a minimum time interval Δ such that the intervals between observations are multiples of Δ . The probe sequence can then be regarded as a periodic sequence with period Δ with some missing data points. We develop a missing-data EM algorithm to deal with this situation.

We next describe the missing-data EM algorithm; the complete-data EM algorithm being a special case of it. The detailed derivation and description of the algorithms can be found in [15]. We first define some notations conforming to those used in [14]. Let (i, j) represent a state with hidden component i and observation symbol j . An element in the transition matrix P is denoted as $p_{(i,j)(k,l)}$ representing the probability of transition from state (i, j) to state (k, l) . Define $\alpha_t(i, j)$ to be the probability of the observation sequence up to time t and the state being in (i, j) at time t , given λ . That is

$$\alpha_t(i, j) = P(Z_t = (i, j), Y_1 = y_1, Y_2 = y_2, \dots, Y_t = y_t \mid \lambda)$$

Define $\beta_t(i, j)$ to be the probability of the observation sequence from time $t+1$ to T , given state being in (i, j) at time t , given λ . That is

$$\beta_t(i, j) = P(Y_{t+1} = y_{t+1}, \dots, Y_T = y_T \mid Z_t = (i, j), \lambda)$$

Define $\xi_t(i, j, k, l)$ to be the probability of state being in (i, j) at time t and in (k, l) at time $t + 1$, given the observation sequence and λ . Define $\gamma_t(i, j)$ to be the probability of being in state (i, j) at time t , given the observation sequence and λ . That is

$$\begin{aligned}\xi_t(i, j, k, l) &= P(Z_t = (i, j), Z_{t+1} = (k, l) \mid Y_1 = y_1, Y_2 = y_2, \dots, Y_T = y_T, \lambda) \\ \gamma_t(i, j) &= P(Z_t = (i, j) \mid Y_1 = y_1, Y_2 = y_2, \dots, Y_T = y_T, \lambda)\end{aligned}$$

We derive $\xi_t(i, j, k, l)$ from $\alpha_t(i, j)$ and $\beta_{t+1}(k, l)$ as follows

$$\xi_t(i, j, k, l) = \frac{\alpha_t(i, j)p_{(i,j)(k,l)}\beta_{t+1}(k, l)}{\sum_{i=1}^N \sum_{j=1}^M \sum_{k=1}^N \sum_{l=1}^M \alpha_t(i, j)p_{(i,j)(k,l)}\beta_{t+1}(k, l)} \quad (1)$$

Observe that γ_t can be calculated from ξ_t as

$$\gamma_t(i, j) = \sum_{k=1}^N \sum_{l=1}^M \xi_t(i, j, k, l). \quad (2)$$

The EM algorithm is an iterative algorithm in which each iteration consists of two steps: the expectation step and the maximization step. During the expectation step, we compute the expected number of transitions from state (i, j) and the expected number of transitions from state (i, j) to state (k, l) using the model parameters obtained during the previous iteration. Denote these as $n(i, j)$ and $m(i, j, k, l)$ respectively. During the maximization step, we calculate the new model parameters from $n(i, j)$ and $m(i, j, k, l)$. The iteration ends when the difference between the new model and the previous model parameters are less than a certain threshold.

Without loss of generality, we assume the observation values for Y_1 and Y_T are known. Let y_t be the observation value for Y_t . We say $y_t = u$ if the observation for Y_t is missing. In the expectation step, we first calculate α and β using the procedures as follows. The procedure to calculate $\alpha_t(i, j)$, where $1 \leq t \leq T, 1 \leq i \leq N, 1 \leq j \leq M$, consists of the following steps:

(1) Initialization

$$\alpha_1(i, j) = \begin{cases} \pi(i, y_1), & j = y_1. \\ 0, & j \neq y_1. \end{cases}$$

(2) Induction

$$\alpha_{t+1}(i, j) = \begin{cases} 0, & y_{t+1} \neq u, j \neq y_{t+1} \\ \sum_{k=1}^N \sum_{l=1}^M \alpha_t(k, l) p_{(k,l)(i,j)}, & \text{Otherwise} \end{cases}$$

where $t = 1, 2, 3, \dots, T - 1$.

The procedure to calculate $\beta_t(i, j)$, where $1 \leq t \leq T, 1 \leq i \leq N, 1 \leq j \leq M$, contains the following steps:

(1) Initialization

$$\beta_T(i, j) = \begin{cases} 1, & j = y_T. \\ 0, & j \neq y_T. \end{cases}$$

(2) Induction

$$\beta_t(i, j) = \begin{cases} 0, & y_t \neq u, j \neq y_t \\ \sum_{k=1}^N \sum_{l=1}^M p_{(i,j)(k,l)} \beta_{t+1}(k, l), & \text{Otherwise} \end{cases}$$

where $t = T - 1, T - 2, \dots, 1$.

Once α and β are obtained, we calculate ξ and γ using (1) and (2). Then we calculate $n(i, j)$ and $m(i, j, k, l)$ from $\gamma_t(i, j)$ and $\xi_t(i, j, k, l)$ as follows

$$n(i, j) = \sum_{t=1}^{T-1} \gamma_t(i, j) \quad (3)$$

$$m(i, j, k, l) = \sum_{t=1}^{T-1} \xi_t(i, j, k, l) \quad (4)$$

The new model parameter estimates are obtained in the maximization step as follows

$$\hat{p}_{(i,j)(k,l)} = \frac{m(i, j, k, l)}{n(i, j)} \quad (5)$$

$$\hat{\pi}_{(i,j)} = \gamma_1(i, j) \quad (6)$$

2.2 Obtain the infinitesimal generator Q from the transition matrix P

We next describe a method to calculate the infinitesimal generator Q from the transition matrix P . By solving the Kolmogorov equations, we obtain the following relation between P and Q

$$P = e^{\Delta Q} \quad (7)$$

Therefore, when $\ln(P)$ exists, Q can be calculated as follows

$$Q = \frac{1}{\Delta} \ln(P) \quad (8)$$

We extend the Taylor expansion $\ln(x) = \sum_{n=1}^{\infty} (-1)^{n-1} (x-1)^n/n, 0 < x < 2$ to matrices and define

$$\ln(P) = \sum_{n=1}^{\infty} (-1)^{n-1} \frac{(P-I)^n}{n} \quad (9)$$

when the right hand side of (9) converges, where I is the identity matrix, we have

$$Q = \frac{1}{\Delta} \sum_{n=1}^{\infty} (-1)^{n-1} \frac{(P-I)^n}{n} \quad (10)$$

In this paper, we consider the complete metric space of matrices of size $MN \times MN$ with the maximum absolute row sum norm as the metric [16]. We next state a theorem describing the convergence condition of (9).

Theorem 1 *The right hand side of (9) converges if $\forall i, p_{ii} > \frac{1}{2}$.*

Proof: Define $f_n(P) = \sum_{i=1}^n \frac{(-1)^{i-1} (P-I)^i}{i}$. We prove that $f_n(P)$ converges when $n \rightarrow \infty$ by showing that $f_n(P)$ is a Cauchy sequence. First, because $\forall i, p_{ii} > \frac{1}{2}$, we have

$$\begin{aligned} \|P - I\| &= \max_i (|p_{ii} - 1| + \sum_{j \neq i} p_{ij}) = \max_i (1 - p_{ii} + 1 - p_{ii}) \\ &= 2 \max_i (1 - p_{ii}) = 2(1 - \min_i p_{ii}) < 1 \end{aligned}$$

Without loss of generality, assume $m \leq n$, then

$$\begin{aligned} \|f_m(P) - f_n(P)\| &= \left\| \sum_{i=m+1}^n \frac{(-1)^{i-1} (P-I)^i}{i} \right\| \leq \sum_{i=m+1}^n \frac{\|(P-I)^i\|}{i} \\ &\leq \sum_{i=m+1}^n \frac{\|P-I\|^i}{i} \leq \sum_{i=m+1}^n \|P-I\|^i = \|P-I\|^{m+1} \sum_{i=0}^{n-m-1} \|P-I\|^i \\ &\leq \|P-I\|^{m+1} \sum_{i=0}^{\infty} \|P-I\|^i = \frac{\|P-I\|^{m+1}}{1 - \|P-I\|} \end{aligned}$$

For any $\epsilon > 0$, to satisfy $\frac{\|P-I\|^{m+1}}{1-\|P-I\|} < \epsilon$, we need $\ln(\|P-I\|^{m+1}) < \ln \epsilon + \ln(1 - \|P-I\|)$. Therefore,

$$m > \frac{\ln \epsilon + \ln(1 - \|P-I\|)}{\ln \|P-I\|} - 1$$

Hence, $\forall \epsilon > 0, \exists N = \lceil \frac{\ln \epsilon + \ln(1 - \|P-I\|)}{\ln \|P-I\|} \rceil - 1$, such that for all $m, n > N$, $\|f_m(P) - f_n(P)\| < \epsilon$. Therefore, $f_n(P)$ is a Cauchy sequence. Thus $f_n(P)$ converges by the completeness of the metric space. ■

Note that the convergence condition of Theorem 1 is sufficient but not necessary. Our experiments show that very often convergence holds even when the elements on the diagonal of matrix P are much less than $\frac{1}{2}$. We next state a theorem which provides a more relaxed convergence condition than Theorem 1.

Theorem 2 *The right hand side of (9) converges if $\exists c > 0$ and $\delta > 0$, such that there exists $N = N(P, \delta, c)$ satisfying $\forall n > N, \|(P-I)^n\| \leq \frac{c}{n^\delta}$.*

Proof: Since $\forall n > N, \|(P-I)^n\| \leq c/n^\delta$, we have

$$\left\| \sum_{n=N+1}^{\infty} (-1)^{n-1} \frac{(P-I)^n}{n} \right\| \leq \sum_{n=N+1}^{\infty} \frac{\|(P-I)^n\|}{n} \leq \sum_{n=N+1}^{\infty} \frac{c}{n^{1+\delta}}$$

Since $\sum_{n=1}^{\infty} \frac{1}{n^{1+\delta}}$ converges, $\sum_{n=1}^{\infty} (-1)^{n-1} \frac{(P-I)^n}{n}$ converges. ■

When the right hand side of (9) does not converge, we approximate the infinitesimal generator by using

$$Q = \frac{1}{\Delta}(P-I) \tag{11}$$

Our numerical and experimental validations show that the approximation is reasonably good.

3 Numerical validation of the model

In this section, we apply the model inference procedure to observation traces generated by known CT-HMMs. After a model is inferred, we examine if it is close to the original model for validation. We first describe the results from one

model in detail. This model has 2 hidden states and 3 observation symbols. Its infinitesimal generator Q is shown below:

$$Q = \begin{bmatrix} -4 & 2 & 1 & 1 & 0 & 0 \\ 1 & -5 & 3 & 0 & 1 & 0 \\ 2 & 3 & -6 & 0 & 0 & 1 \\ 2 & 0 & 0 & -7 & 2 & 3 \\ 0 & 2 & 0 & 3 & -8 & 3 \\ 0 & 0 & 2 & 2 & 5 & -9 \end{bmatrix}$$

We use the model to generate a 1000-second long trace. Using sampling intervals of 1ms and 100ms, we obtain observation sequences O_1 (of length 1,000,000) and O_{100} (of length 10,000) respectively. The sequence O_{100} is used for model inference. The sequence O_1 is used to check the quality of the inferred CT-HMM, as described later. Since we cannot observe the number of hidden states N directly from the observation sequence O_{100} , we explore the sensitivity of the models to the choice of N . In particular we consider N in $\{2, 3, \dots, 10\}$ for model inference.

We observe that the inferred CT-HMM is not guaranteed to be exactly the same as the original model, since the results of the EM algorithm are local maxima and the inference is from a discrete-time sample of the original model. For example, the inferred infinitesimal generator \hat{Q} when $M = 3$ and $N = 2$ is different from the original model Q as shown below

$$\hat{Q} = \begin{bmatrix} -3.24 & 1.57 & 0.03 & 0.01 & 0.00 & 1.64 \\ 1.53 & -4.44 & 0.05 & 0.01 & 0.00 & 2.85 \\ 2.07 & 0.00 & -7.67 & 0.00 & 5.53 & 0.07 \\ 0.00 & 1.20 & 1.43 & -4.56 & 1.93 & 0.00 \\ 0.00 & 2.81 & 3.49 & 3.00 & -9.29 & 0.00 \\ 0.06 & 0.00 & 0.33 & 2.32 & 2.97 & -5.67 \end{bmatrix}$$

However, the inferred models are very close to the original model in the following two senses. First, inferred models generate sequences whose autocorrelation functions (ACF) are very close to that of a sequence generated by the original model. Fig. 1 shows the ACFs of a sequence of length 10,000 generated from the inferred models for various values of N and that of the original sequence O_{100} . The lag is in multiples of 100ms. The models for $N = 4$ and $N = 6$ are from (9) while the model for $N = 9$ is from the approximation (11). We observe that the ACFs of these models are very similar to that of the original.

Second, symbols lying between observations can be estimated from an inferred model accurately. For instance, we obtain the MLE sequence at the interval of

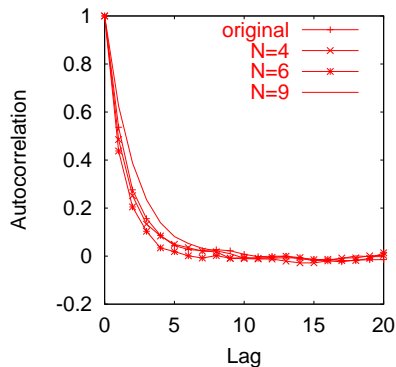


Fig. 1. ACFs for the original and various models, $M = 3$.

1ms using the sequence O_{100} and an inferred model¹. Our results show that 82% of the MLE symbols match the observation symbols in O_1 . Moreover, most of the MLE sequences for different values of N are identical, demonstrating the similarity among the inferred models.

We generate another 9 models for further validation. Each model has 2 hidden states and 3 observation symbols. The transition rates of each model are randomly chosen between 1 and 10. For each model, we investigate the range of N from 2 to 10 for a total of 81 experiments. Among them, (9) converges for 62 cases. In the remaining 19 cases, the sum in (9) diverges mostly for high values of N ($N \geq 6$). The reason why higher value of N leads to divergent results can be explained as follows. The number of states in the CT-HMM increases linearly with N . For large values of N , the probability of the process remaining in a given state after one transition tends to be very small. Although Theorem 1 is not a necessary condition for convergence, very small values of p_{ii} tends to lead to divergence in (9). We obtain approximate models for the 19 divergent cases using (11). In the MLE sequences obtained from the models, 82% to 90% of the MLE symbols match the observation symbols in O_1 . Furthermore, the MLE sequences from the models for different values of N are mostly identical.

We also examine the behavior of an inferred model of size smaller than the original model and observe similar results as earlier. The details are in [15]. In summary, our numerical validation shows that, for a given model, the models inferred by our algorithm for different values of N are close to the original model and close to each other.

¹ The method to obtain MLE sequence is described in [15]

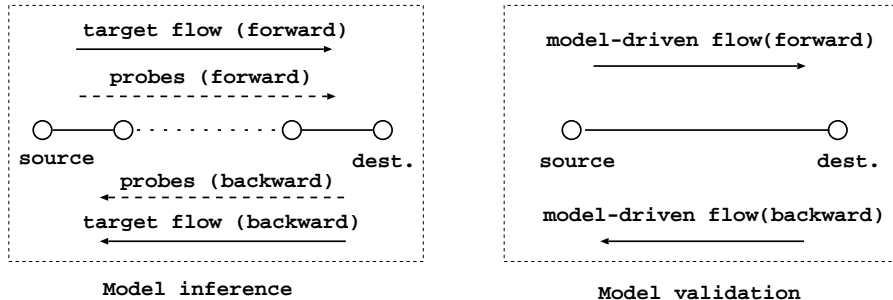


Fig. 2. Model inference and validation

4 Validation of the the method

In this section, we describe the methodology used to validate our simulation method. The validation is based on a series of experiments carried out in *ns* and over the Internet. In *ns*, we investigate a variety of settings characterized by different traffic and topologies. Over the Internet, we examine several settings characterized by different delay and loss behaviors. We believe that these experiments show our approach to be quite promising.

As shown in the model inference part of Fig. 2, in each experiment, we send probes from the source to the destination along with a *target* flow. The behavior of the target flow is governed by a network protocol or an application. When necessary, probe packets are also sent along the backward direction of the target flow. The observations of the forward and backward probes are used to infer CT-HMMs for the forward and backward paths respectively. The models are then used to generate end-to-end losses and delays for packets in a flow, as shown in the model validation part of Fig. 2. We validate this simulation method by comparing the behavior of the target flow in the original setting with the model-driven flow. Note that the link controlled by the model represents the whole path in the original network. Hence the running time required by our simulation method is independent of the complexity of the network being simulated.

The probes are either sent periodically with the spacing of Δ or $l\Delta$, where l is a geometrically distributed random variable. We refer to these as periodic probing and geometric probing respectively. For periodic probing, Δ is set to 20ms, 50ms or 100ms. For geometric probing, let τ be the average inter-sending time. We set τ to be 20ms, 50ms or 100ms and Δ to 1ms or 10ms. The random variable l thus has a geometric distribution with the parameter of Δ/τ . We next describe the experimental methodology in detail.

4.1 Loss and delay as symbols

The CT-HMM uses M symbols to represent the observed delay and loss sequence. One way to discretize the delay and loss sequence is to use $(M - 1)$ symbols to represent delay and a single symbol to represent loss. Another way is to combine loss and large delays into one symbol since large delays and loss are usually closely related. We find that the first method requires very fine probing interval since the duration of a loss behavior is shorter than the processing time of one packet. We therefore adopt the second approach. More specifically, we divide the range of delay values into M equal length bins. A delay value falling in the i th ($1 \leq i \leq M$) bin is represented by symbol i . More formally, suppose we have a probing observation sequence $\{d_t\}_{t=1}^T$. Let the minimum delay and the maximum delay be d_{min} and d_{max} respectively. We use $d_t = \infty$ to denote that the observation at t is a loss. We say $d_t = u$ if the observation at t is missing. The interval $[d_{min}, d_{max}]$ is divided into M bins. The length of each bin is denoted as b . Then $b = (d_{max} - d_{min})/M$. We convert the delay and loss observation sequence to a symbol sequence $\{y_t\}_{t=1}^T$ as follows

$$y_t = \begin{cases} M, & d_t = \infty, \\ \lceil \frac{d_t - d_{min}}{b} \rceil, & d_{min} \leq d_t \leq d_{max}, \\ u, & d_t = u. \end{cases}$$

At the same time, we record the probability that symbol M corresponds to a loss. Note that, in this way, we regard that loss is only correlated with the largest delay symbol M . In practice, the correlation of loss and large delays may not be strong in some situations, for instance, when RED is used for queue management in the routers. In [15], we extend the EM algorithm in Section 2.1 so that there is a component of loss associated with each symbol.

4.2 Mapping symbols to delay and loss values

Once a CT-HMM is inferred, we can determine an observation symbol for any point of time from the model. For the purpose of simulation, this symbol has to be mapped back into delay or loss. When mapping a symbol i to delay or loss, if $i = M$, we first decide if it is a loss according to the loss probability in symbol M . If it is not a loss, we generate a delay for it according to the conditional distribution of delays given that the symbol is i , which is obtained from the probing trace. The details are in [15].

4.3 Usage of virtual probes in ns

For a given setting, we need to investigate the simulation results over a range of probing intervals. In order to save time and space, and to make the results at different probing intervals comparable, we use virtual probes to study a setting with a single link first. Unlike a real probe, a virtual probe does not generate traffic. It records the loss or delay value at a certain point of time by analyzing the *ns* traces. A virtual probe sees a loss if the packet before it is dropped. Otherwise, the delay seen by the probe is calculated by keeping track of the enqueue time and processing time of all of the previous packets. This simulation method allows us to impose “probes” at a range of probing intervals to discover the granularity of the probing required for a network setting. We, for instance, confirmed that very fine probing intervals are required if loss is represented as a single symbol as described in Section 4.1.

4.4 Measurement of one-way delay over the Internet

If the clocks at the sender and receiver are synchronized, the subtraction of the timestamp of a packet at the receiver and the sender is the one-way delay. However, we do not have an external universal time reference to synchronize the two clocks. Given this, we use the method proposed in [17] to remove clock offset and skew. The resulting delay sequence represents the variable portion of the one-way delays over the network. To obtain the actual delay value, a constant portion has to be added, which contains the one-way transmission delay, propagation delay, and the minimum queuing delay experienced by the probe packets during the chosen time period. We describe the estimation of this value when describing the experiments (in Sections 5.2 and 6).

5 Experimental Validations for TCP

In this section, we validate the simulation method through a series of experiments for TCP carried out in *ns* and over the Internet. That is, in each experiment, we send probes along with a target TCP flow on the forward and backward paths. The forward and backward probes are used to infer the models for the forward and backward paths respectively. The inferred models are then used to provide delay and loss to packets along the forward and backward paths of the model-driven TCP flow.

The metrics we use are *average loss rate*, *average throughput*, and *ACF of the throughput sequence*. The average loss rate records the loss ratio of data

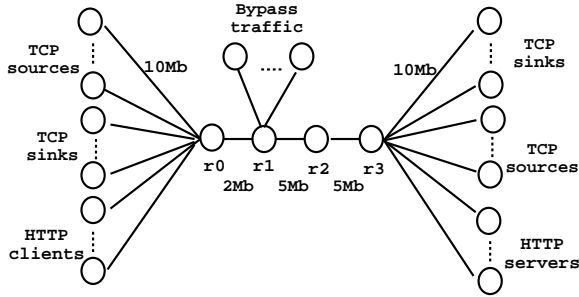


Fig. 3. TCP and HTTP sources with multiple links in *ns*.

packets from the TCP source. The average throughput records the amount of data reaching the TCP sink from the source. The ACF of the throughput sequence records the temporal dependence of throughput in the TCP flow.

5.1 Validation in *ns*

We next validate our simulation method through experiments under a variety of network settings in *ns*. We study topologies containing a single link and multiple links. The traffic combinations include infinite TCP sources only, infinite TCP sources with on-off UDP sources, and infinite TCP sources with HTTP sources. The target TCP flow uses Reno or SACK. For all the settings we examined, the loss and throughput predictions made by the model-driven simulation falls within 10% of those of the target TCP flow in the original setting if M and N are chosen properly. In the interest of space, we only describe results for traffic combinations of TCP and HTTP sources with multiple links. More results are described in [15].

In this setting, we connect four routers r_0 , r_1 , r_2 and r_3 as shown in Fig. 3. The bandwidth between router r_0 and r_1 is set to be 2Mb. The bandwidth between the other consecutive routers (router r_1 to r_2 and router r_2 to r_3) is set to be 5Mb. The bandwidth for all the other links is set to be 10Mb. The maximum queue length and propagation delay between two consecutive routers are set to be 100 packets and 30ms respectively. There are two types of TCP flows in this setting: flows traversing all the routers and flows pass part of the routers. we call the first type of flows *traversing* flows and the second type of flows *bypass* flows. The numbers of traversing TCP flows from router r_0 to r_3 and from r_3 to r_0 are set to be 3 and 2 respectively. Each traversing flow starts from a TCP source connected to router r_0 (r_3), with their corresponding TCP sink connected to router r_3 (r_0). Each bypass flow starts from a source connected to router r_i and ends at another router r_j , where $0 \leq i, j \leq 3$. The number

Table 1

Number of bypass TCP flows in the multiple-link setting in *ns*

from \rightarrow to	$r_0 \rightarrow r_1$	$r_0 \rightarrow r_2$	$r_1 \rightarrow r_2$	$r_1 \rightarrow r_3$	$r_2 \rightarrow r_3$
number	3	1	2	2	4

from \rightarrow to	$r_1 \rightarrow r_0$	$r_2 \rightarrow r_0$	$r_2 \rightarrow r_1$	$r_3 \rightarrow r_1$	$r_3 \rightarrow r_2$
number	1	2	3	3	2

Table 2

Relative errors for TCP and HTTP sources with multiple links in *ns*

$M_f(= M_b)$	$N_f(= N_b)$	throughput error	loss error
3	2	-7.7%	15.6%
4	2	8.1%	-0.0%
5	2	17.0%	1.9%
2	3	-9.9%	8.5%
3	3	5.6%	-10.7%
4	3	9.3%	-3.2%
2	4	-3.9%	1.7%
3	4	-3.5%	7.2%

of bypass flows along the path from router r_i to r_j are listed in Table 1. The propagation delay between a TCP source or sink to its corresponding router is uniformly distributed in [10, 20]ms. There are 10 HTTP clients connected to router r_0 , with their corresponding HTTP servers connected to router r_3 . The HTTP traffic follows the empirical data provided by *ns*. The propagation delay from an HTTP server or client to its corresponding router is also set to be uniformly distributed in [10, 20]ms.

We choose a traversing TCP flow from r_0 to r_3 as the target flow and send probe packets with the TCP flow along both the forward and backward directions at the probing interval of 20ms. The target TCP flow encounters a loss rate of 1.49% along the path of r_0 to r_3 and its average throughput is 0.144Mbps. The errors of both throughput and loss rate of the model-driven TCP flow relative to the target TCP flow are shown in Table 2. We observe that in general the performance of a model improves as the number of states increases. The relative errors of the throughput and the loss rate fall within 17% for various settings.

Fig. 4 compares the ACFs of the throughput of a model-driven TCP and the target TCP flow, where lags are measured in seconds. The model used in the graph is inferred using $M = 3$ and $N = 2$. The figure demonstrates a good match of the two functions.

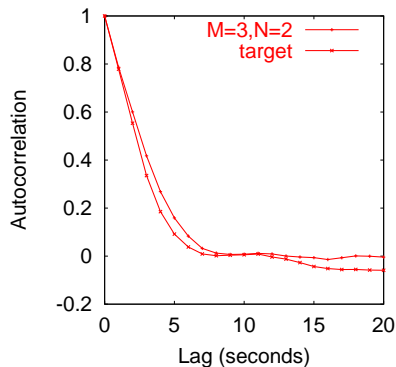


Fig. 4. ACF of throughput for TCP and HTTP sources with multiple links in *ns*.

Table 3

Two validation experiments for TCP over the Internet

Trace	Time	Probing	Loss rate	Throughput
T_1	03/08/02 22:24	Periodic, $\Delta=20\text{ms}$	1.51%	0.214Mbps
T_2	03/16/02 14:06	Geometric, $\Delta=10\text{ms}$, $\tau=50\text{ms}$	0.41%	1.578Mbps

5.2 Validation over the Internet

We next describe our validation of the simulation method over the Internet. Two traces T_1 and T_2 are shown in Table 3. Each trace is collected by a 1-hour run and contains two-way (along the forward and the backward paths) traces. For both traces, the sender of the target TCP is at our site (UMass) and the receiver is at University of Southern California (USC). The number of hops from UMass to USC is 20 and the number of hops from USC to UMass is 14. The measurement of one-way delays is described in Section 4.4. We take half of the minimum round-trip time of the target TCP flows as the constant portion to obtain the actual delays. To incorporate the models into *ns* and drive a TCP flow, we need to obtain some key parameters used in the target TCP flow. We find that the target TCP flow uses TCP SACK by the method described in [18]. Most of the packets in the target TCP flow (over 99%) are 1448 bytes. The receiver window size changes over the session while it is fixed in *ns*. We obtain a rough estimate of the window size from [19] and fine-tune it during the validation process. We next present the results for traces T_1 and T_2 .

In trace T_1 , probes are sent at regular intervals Δ of 20ms on the directions from UMass to USC and from USC to UMass. We focus on a stationary segment of 300 seconds. On the route from UMass to USC, the average loss rate and throughput of the target TCP flow are 1.51% and 0.214Mbps respectively, as shown in Table 3. In *ns*, we set the one-way delay to be 100ms and

Table 4

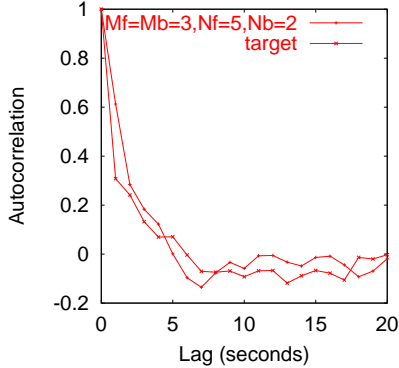
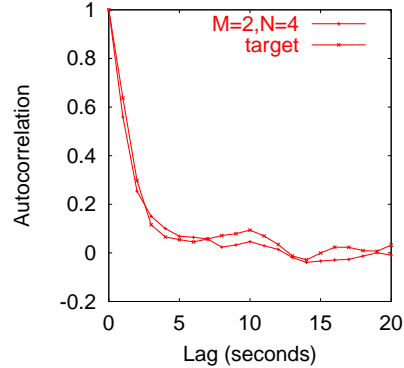
Relative errors of throughput and loss rate for trace T_1

M_f	N_f	M_b	N_b	throughput error	loss error	M_f	N_f	M_b	N_b	throughput error	loss error
2	3	2	2	-0.4%	-12.0%	2	3	2	3	-0.3%	10.1%
2	4*	2	2	-4.2%	8.0%	2	4*	2	3	-4.7%	-6.7%
2	5*	2	2	-5.6%	9.4%	2	5*	2	3	-5.5%	10.4%
3	3	3	2	-1.0%	16.0%	3	3	3	3	-0.6%	11.0%
3	4*	3	2	2.9%	-3.3%	3	4*	3	3	1.1%	-9.0%
3	5*	3	2	3.1%	-0.6%	3	5*	3	3	2.6%	2.3%

Table 5

Relative errors of throughput and loss rate for trace T_2

M_f	N_f	throughput error	loss error	M_f	N_f	throughput error	loss error
2	2	-0.1%	-3.0%	2	6	-0.3%	-1.1%
2	3	2.8%	-12.4%	2	7	-1.4%	4.0%
2	4	2.4%	-10.4%	2	8	-1.5%	3.6%
2	5	-0.6%	0.3%	2	9	1.4%	-6.8%

(a) ACF of throughput for T_1 .(b) ACF of throughput for T_2 .Fig. 5. ACFs of throughput for traces T_1 and T_2 .

the window size to be 8. Table 4 summarizes the errors of throughput and loss rate of the model-driven TCP flows relative to the target TCP flow for different settings of the models, where entries marked by * are obtained by the approximation (11). We observe that, for this trace, the number of observation symbols as 2 to 3 and the number of hidden states as 2 to 5 are sufficient to achieve good results. The comparison of the ACF of the throughput for the model-driven TCP and the target TCP flow is shown in Fig. 5(a), where lags are measured in seconds and the model is obtained by setting $M_f = 3$, $N_f = 5$, $M_b = 3$, $N_b = 2$.

For trace T_2 (see Table 3), we use geometric probing on the forward and the

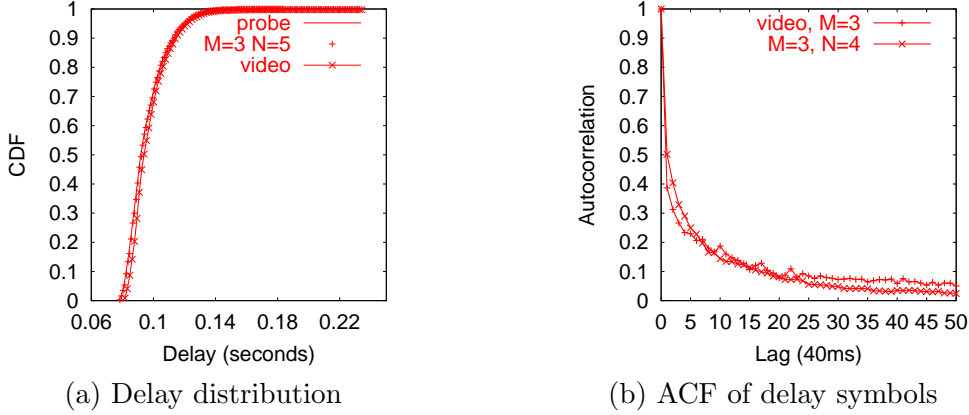


Fig. 6. Results for a streaming video application.

backward paths with the average probing interval τ of 50ms. Due to limitation in the time granularity of Linux, the minimum probing interval Δ is set to be 10ms. We focus on a stationary segment of 1200 seconds and use the missing-data EM algorithm in the model inference process. As shown in Table 3, the average loss rate in this trace is 0.41%, significantly lower than in trace T_1 . The average throughput is 1.578Mbps. In *ns*, we set the one-way delay to be 43ms and the window size to be 18. We cannot find a convergent solution for the model on the backward path even for $M_b = 2$ and $N_b = 2$. The reason is that there are no losses on the backward path and the delay variations are in a very short range (≤ 2 ms). We therefore use a constant delay (the average delay on the backward path) for the backward direction. The errors of the model-driven TCP flow relative to the target TCP are shown in Table 5. The relative errors of throughput and loss rate of all the settings in the table fall within 3% and 13% respectively. We also observe good conformance of the model-driven TCP flow and the target TCP from the ACFs of the throughput as shown in Fig. 5(b), where lags are measured in seconds.

6 Validation for a streaming video application

In this section, we validate the simulation method using a streaming video application. We send probe packets along with a streaming video application from one end host to another end host. The streaming video application uses UDP and there are no acknowledgments along the backward path. Therefore we only need to infer the model along the forward path. Since streaming media applications are sensitive to delays and losses, we use the following metrics:

- average loss rate, average delay and CV (coefficient of variation) of the delays
- distribution of the delays
- ACF of the delay and loss symbols

Table 6
Relative error for a streaming video application

M	N	delay error	CV error	M	N	delay error	CV error
2	2	-2.1%	-3.5%	3	2	-2.3%	0.4%
2	3	-2.0%	1.9%	3	3	-2.4%	-0.6%
2	4	-2.1%	0.9%	3	4	-2.4%	-0.3%
2	5	-2.0%	1.8%	3	5	-2.2%	-0.6%

The one-way delays from the sender to the receiver are obtained as described in Section 4.4. To calculate the ACF of the delay symbols, the obtained variable portion of the one-way delay is sufficient. To calculate the other delay metrics, we need to obtain the actual delay by adding a constant portion to the variable portion. However, unlike with TCP, this constant portion does not affect the behavior of the application. We estimate the constant portion roughly from the round-trip time reported by *ping* packets. We send *ping* packets lasting for 30 minutes before the experiments and take half of the minimum round-trip time to be the constant portion.

We next describe an experiment from our site (UMass) to a site in Universidade Federal do Rio de Janeiro (UFRJ), Brazil, which was carried out at 10:25am on April 1, 2002 and lasted for 1 hour. The number of hops from UMass to UFRJ is 14. We use a 200Kbps CBR (Constant Bit Rate) video. The frame rate of the video is 25 frames per second. For this experiment, we use geometric probing with the average probing interval τ as 50ms and the minimum time interval Δ as 10ms. The constant portion of the delay is estimated to be 78ms.

We choose a 1200-second stationary segment from the video stream as the target stream and use the missing data EM algorithm in the model inference process. For the target stream, the average loss rate is 0.2%, the average delay is 99ms and the CV of the delays is 0.12. The errors of delay and CV of delays of the model-driven stream relative to the target stream are shown in Table 6, where M is from 2 to 3 and N is from 2 to 5. For various settings of M and N , the delay errors are within 3% and the errors of CV of the delays are within 4%. The loss rates for various settings are around 0.2%.

The delay cumulative distribution functions (CDFs) of the target video stream, probes and a model-driven video stream are shown in Fig. 6 (a). We observe that the delay distribution of the model-driven stream is almost identical to that of the probes. The difference in the delay distribution of the model-driven stream and the target stream is caused by the different observations seen by the probes and the target stream.

We compare the ACFs of the delays of the target stream and model-driven stream for different settings of M and N . We observe that, for the same M ,

the ACF becomes closer to that of the target stream as N increases. Fig. 6 (b) shows the ACFs of the delay symbols of the target video stream and that of model-driven streams. Each lag is 40ms. The model is inferred by setting $M = 3$ and $N = 4$.

7 Conclusion and Future work

In this paper, we explored the use of CT-HMMs for network protocol and application evaluation. We develop a mechanism to infer a CT-HMM from a sequence of observations of probe packets. Our validations of this simulation method in *ns* and over the Internet demonstrate that this method has the potential to represent a wide range of *real* network scenarios. It is accurate, and time and space efficient. We observe that for real network settings we examined, models of small size are sufficient to achieve good results. As future work, we are pursuing the following directions: (i) examine the use of the model in more real network scenarios, (ii) investigate how to use the probing trace in combination with the CT-HMM to model transient behavior in the network.

Acknowledgments

We would like to thank J. Kurose, E. A. de Souza e Silva and the anonymous reviewers for their insightful comments. We would also like to thank L. Golubchik, C. Papadopoulos and E. A. de Souza e Silva for providing us accounts for the Internet experiments; W. Cheng, F. J. Silveira Filho, A. Lee, A. Papagiapiou and A. P. Couto Silva for their cooperations in the experiments. B. Melander laid the foundation for our *ns* validations. D. R. Figueiredo, B. Liu, J. Padhye, T. Bu and H. Zhang helped us with the *ns* and Internet experiments. We would like to express our thanks to all of them.

References

- [1] G. Riley, R. Fujimoto, and M. Ammar, "Parallel/Distributed NS." <http://www.cc.gatech.edu/computing/compass/pdns/index.html>.
- [2] S. McCanne and S. Floyd, "ns-LBNL network simulator," <http://www-nrg.ee.lbl.gov/ns/>.
- [3] J. H. Cowie, "Scalable simulation framework API reference manual," March 1999.

- [4] Y. Guo, “Time-stepped hybrid simulation (TSHS) for large scale networks,” in *Proc. IEEE INFOCOM*, March 2000.
- [5] B. Liu, D. R. Figueiredo, Y. Guo, J. Kurose, and D. Towsely, “A study of networks simulation efficiency: fluid simulation vs. packet-level simulation,” in *INFOCOM*, 2001.
- [6] L. Rizzo, “Dummynet: a simple approach to the evaluation of network protocols,” *ACM Computer Communication Review*, vol. 27, no. 1, pp. 31–41, 1997.
- [7] M. Carson, “NIST Net,” <http://snad.ncsl.nist.gov/itg/nistnet>.
- [8] V. Paxson, “End-to-end Internet packet dynamics,” *IEEE/ACM Transactions on Networking*, vol. 7, no. 3, pp. 277–292, 1999.
- [9] D. Rubenstein, J. Kurose, and D. Towsley, “Detecting shared congestion of flows via end-to-end measurement,” in *Proc. ACM SIGMETRICS*, June 2000.
- [10] M. Yajnik, J. Kurose, and D. Towsely, “Packet loss correlation in the Mbone multicast network,” in *IEEE Global Internet*, November 1996.
- [11] M. Yajnik, S. Moon, J. Kurose, and D. Towsley, “Measurement and modelling of the temporal dependence in packet loss,” in *Proc. IEEE INFOCOM*, March 1999.
- [12] K. Salamatian and S. Vaton, “Hidden Markov modelling for network communication channels,” in *Proc. ACM SIGMETRICS*, June 2001.
- [13] S. Ross, *Stochastic Process*. John Wiley & Sons, 1996.
- [14] L. Rabiner, “A tutorial on hidden Markov models and selected applications in speech recognition,” in *Proceedings of the IEEE*, vol. 77, pp. 257–285, February 1989.
- [15] W. Wei, B. Wang, and D. Towsley, “Continuous-time hidden Markov models for network performance evaluation,” Tech. Rep. 02-15, Department of Computer Science, University of Massachusetts, Amherst, 2002.
- [16] I. Ryzhik, I. S. Gradshteyn, and A. Jeffrey, *Table of Integrals, Series, and Products*. Academic Press, 6th ed., 2000.
- [17] S. Moon, P. Skelly, and D. Towsley, “Estimation and removal of clock skew from network delay measurements,” in *Proc. IEEE INFOCOM*, March 1999.
- [18] J. Padhye and S. Floy, “On inferring TCP behavior,” in *Proc. ACM SIGCOMM*, August 2001.
- [19] M. Mathis, J. Semke, and J. Mahdavi, “The macroscopic behavior of the TCP congestion avoidance algorithm,” *Computer Communications Review*, vol. 27, no. 3, 1997.