Analysis of Windowing Mechanisms with Infinite-State Stochastic Petri Nets

Alexander Ost, Boudewijn R. Haverkort

Abstract— **In this paper we present a performance evaluation of windowing mechanisms in world-wide web applications. Previously, such mechanisms have been studied by means of measurements only, however, given suitable tool support, we show that such evaluations can also be performed conveniently using infinite-state stochastic Petri nets. We briefly present this class of stochastic Petri nets as well as the approach for solving the underlying infinite-state Markov chain using matrix-geometric methods. We then present a model of the TCP slow-start congestion avoidance mechanism, subject to a (recently published) typical world-wide web workload. The model is parameterized using measurement data for a national connection and an overseas connection. Our study shows how the maximum congestion window size, the connection release timeout and the packet loss probability influence the expected number of buffered segments at the server, the connection setup rate and the connection time.**

Keywords— **Matrix-geometric methods, stochastic Petri nets, window flow control, congestion control.**

I. Introduction

When modeling and evaluating the performance of modern distributed systems, complex system behavior (involving networks, switches, servers, flow control mechanisms, etc.) as well as very complex workloads (often a mix of batch data, interactive data and real-time data for voice and video) need to be taken into account to come up with trustworthy performance measures. This most often leads researchers to either use a measurement-based approach or simulation as performance evaluation technique. As both these techniques are rather costly, it is worthwhile to study the suitability of analytic/numerical approaches based on stochastic Petri nets (SPNs) as well.

As a challenging application for such a suitability study we have chosen the handling of WWW (World Wide Web) requests by the hypertext transfer protocol (HTTP) [1]. Clearly, there are many factors involved in this process: the client issuing the request, the server handling the request, the type of request (just text, text with embedded graphics, pictures, or even video), the Internet connecting the client and the server, as well as the transport protocols used by client and server (TCP/IP). Taking all these aspects into account would render an analytical solution impossible, as also witnessed by the many measurement-based performance studies in this area ([2], [3]). To the best of the knowledge of the authors, no analytical performance studies for such systems have been reported yet.

In this paper we propose to use a special class of SPNs for the construction and efficient numerical solution of performance models for the handling of WWW requests. Our models include client characteristics (request pattern) and request types, the Internet delays, the server speed, and the influence of the underlying TCP/IP protocol. In particular, our model includes the explicit connection set-up (and release) phase of TCP/IP as well as its congestion avoidance windowing mechanism (known as *slow start*). Despite all these details, after a suitable abstraction process, the models can still be solved efficiently with numerical means, thereby using the tool SPN2MGM to exploit the special quasi-birth-death (QBD) structure of the stochastic process underlying the SPN.

The contribution of this paper is twofold. First, it shows the suitability of the SPN-based approach for the performance evaluation of complex systems, provided a suitable model class and solution method is chosen. Secondly, the performance results (obtained with realistic parameters derived from measurements) show the impact of lower-layer protocols (TCP/IP including its windowing mechanism) on the perceived performance at the application level (WWW).

This paper is further organized as follows. In Section II, we concisely describe the class of SPNs and the employed solution methods, as well as the tool support we have developed. We then describe the handling of WWW requests in detail, as well as the corresponding SPN model in Section III. Section IV is then devoted to a wide variety of numerical case studies. Section V concludes the paper.

II. The Modeling Framework

The basis of our modeling framework is a special class of SPNs (so-called *infinite-state* SPNs), which is suitable for an efficient numerical analysis. We present the main properties of these SPNs in the following section. Then, we discuss some issues concerning the numerical solution of the underlying Markov chain, and finally present the tool we developed to support infinite-state-SPN based modeling and analysis.

A. Matrix-Geometric Stochastic Petri Nets

Infinite-state SPNs are especially suited for modeling systems which involve large buffers or queues. Based on the class of (generalized) SPNs proposed by Ciardo in [4], they allow one distinguished place of unbounded capacity (graphically represented as a double-circled place) and represent an extension of the initial work by Florin and Natkin [5]. The unbounded place is usually used to model infinite buffers, or to approximate large finite buffers. While all features of Ciardo's original SPN class are still

The authors are with the Laboratory for Distributed Systems, RWTH Aachen, 52056 Aachen, Germany. E-mail: {arost,haverkort}@informatik.rwth-aachen.de

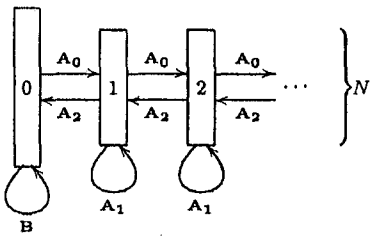

Fig. 1. Leveled structure of the underlying Markov chain of an infinite-state SPN.

available (like immediate transitions, enabling functions, inhibitor arcs and arc multiplicities), there are some restrictions concerning the unbounded place:

- The multiplicity of incoming and outgoing arcs is limited to one.
- Transition rates and weights of immediate transitions may not depend on the number of tokens in the unbounded place.

Due to the unbounded place, the Markov chain underlying the Petri net has an infinite number of states. The advantage of using infinite-state SPNs instead of classical ones is that the infinite state space has a special structure, which makes it eligible to very efficient numerical solution techniques, leading much quicker to a steady-state solution than by investigating a large, finite state space.

B. Numerical Solution

The continuous-time Markov chain underlying an infinite-state SPN is a QBD process [6], [7]. Its state space can be represented two-dimensionally as a strip which is unbounded in one direction. In the case of infinite-state SPNs, the position of a state in the unbounded direction corresponds to the number of tokens in the unbounded place. All states which belong to markings having the same number of tokens in the unbounded place are said to be in one *level*. Due to the restrictions mentioned in the previous section, transitions can only take place between states which belong to the same level, or between states in adjacent levels (see Fig. 1). Furthermore, the transition rates between levels are independent of the level index (except for a boundary level). This leads to an (infinite) generator matrix with the following block-tridiagonal structure:

$$\mathbf{Q} = \begin{pmatrix} \boxed{\mathbf{B}} & \mathbf{0} & \mathbf{0} & \cdots \\ \mathbf{A_0} & \mathbf{0} & \mathbf{0} & \cdots \\ \mathbf{0} & \mathbf{A_2} & \mathbf{A_1} & \mathbf{A_0} & \cdots \\ \mathbf{0} & \mathbf{0} & \mathbf{A_2} & \mathbf{A_1} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}.$$

Except for the boundary, all transition rates can be kept in three square matrices $\mathbf{A_0}, \mathbf{A_1}, \mathbf{A_2}$. The size of these matrices equals the number of states in the (non-boundary) levels (denoted by N in Fig. 1).

There exist several efficient techniques for deriving the steady-state solution π with $\pi\mathbf{Q} = \mathbf{0}$ of these Markov chains. The boundary conditions involving $\mathbf{B}$ represent a normal set of linear equations, and the non-boundary part

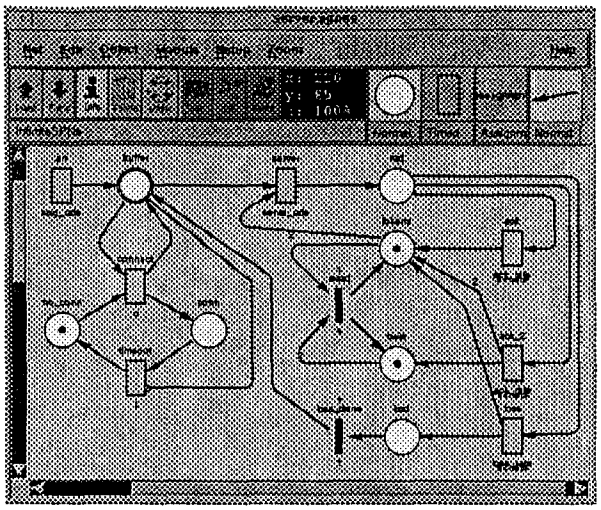

Fig. 2. Snapshot of an editing session with agnes.

is reduced to the quadratic matrix polynomial

$$\mathbf{A_0} + \mathbf{R}\mathbf{A_1} + \mathbf{R}^2\mathbf{A_2} = \mathbf{0}.$$

The solution of this polynomial for $\mathbf{R}$ is the core of the solution techniques. While most methods provide an iterative solution to this problem (like those presented in [6], [8], [9]), the approach suggested in [10] transforms the problem to an Eigenvalue problem. Though the latter method has some interesting properties, the iterative approaches proved to be numerically more stable for large models so far (see [11] for a comparison). The results presented in this paper were obtained using the LR method [8].

C. Tool Support

We developed extensive tool support to simplify the use of infinite-state SPNs for modeling and analysis, and for abstracting details of the underlying Markov chain. Using SPN2MGM ([12], [13]), it is possible to specify the desired performance measures easily at the Petri net level in a reward-based way. Concerning the numerical solution, the user can choose between the algorithms mentioned in the previous section; all of them have been implemented using high-performance linear algebra packages.

For the easy graphical specification of the Petri net, we adopted the generic net editing system agnes [14] (see Fig. 2). In addition, also a textual specification is still possible.

III. Modeling Windowed Traffic Control Mechanisms

The TCP protocol relies on two window traffic control mechanisms to handle flow and congestion control. We focus on TCP's congestion control mechanism, which is described in the following section. Then, an appropriate SPN model is proposed, after which we suggest several approaches for modeling the traffic which has to be delivered by TCP. Finally, we describe how the model parameters used in our experiments are derived.

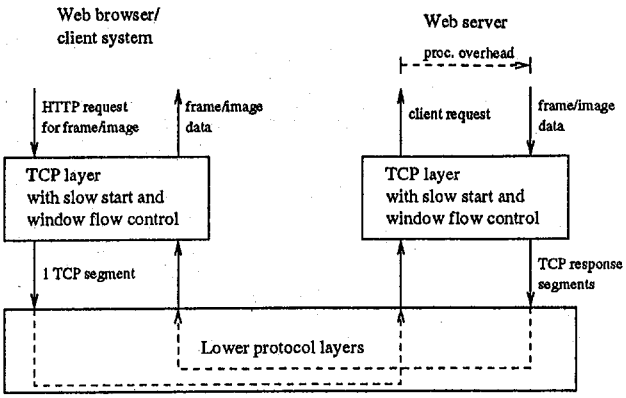

Fig. 3. Accessing WWW documents via TCP/IP.

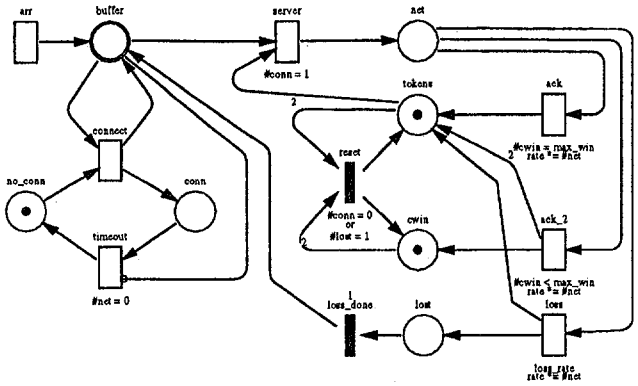

Fig. 4. Petri net for modeling TCP behavior.

A. System Description

A TCP connection is associated with two windows: a receiver window to synchronize the sender's speed with the speed a receiver is able to process incoming data, and a congestion window to avoid network overload. The amount of data sent is given by the minimum of both windows. Assuming that the receiving station can handle incoming traffic sufficiently fast, the size of the congestion window is the limiting factor.

TCP avoids network congestion by limiting the number of packets traveling through the network simultaneously. This is accomplished by introducing the congestion window, holding the remaining tolerable amount of data which can be fed into the network. Once a packet is acknowledged, the available congestion window space is increased by the appropriate number of bytes. Since the tolerable number of packets which can be fed into the network prior to being acknowledged is not known *a priori*, TCP adopts the *slow start* mechanism suggested by Jacobson [15].

Slow start initializes the congestion window size to one packet. Whenever a packet gets acknowledged, the window size is increased by one packet. Thus, the window size effectively grows exponentially. This exponential growth phase is bounded by a *maximum* window size parameter. Once it is exceeded, the congestion window size increases in a linear fashion. If a packet gets lost, the maximum window size parameter is set to half the current window size, and the current window size is set to one packet.

While slow start provides a suitable algorithm for congestion avoidance, the initial phase of "probing for bandwidth" becomes a problem if a TCP connection is set up for transferring only a small amount of data. In particular, the HTTP protocol suffers from this fact, since getting a HTML document and the images referenced therein is accomplished by building separate TCP connections for each of them (see Fig. 3). Since the amount of data per item is typically small, the slow start mechanism leads to a situation where just a fraction of the available bandwidth is used. Furthermore, the establishment of each connection involves the usual TCP three-way handshake, increasing delay by one round trip time. In version 1.1 of the HTTP protocol, the use of persistent connections for several requests (P-HTTP) is possible and recommended, thus minimizing the number of necessary slow start phases and connection setups. Other approaches like T/TCP [16] and UDP-based mechanisms like ARDP [17] have been addressed in [3].

B. Model Development

The model presented in this section has been developed having three main aims in mind. First, the model should account for the window flow control mechanism and capture the influence of the slow start mechanism on transmission performance. Second, connection setups have to be considered, since we are interested in the gain obtained by using persistent protocol versions. Third, the complexity of the overall model should not exceed the numerical capabilities of our analysis tool, so less important details had to be omitted.

The model concentrates on the server-side of a client-server relationship. We assume that the main amount of traffic arises from server to client (as in the HTTP case), thus we neglect the impact of flow- and congestion control mechanisms on traffic which is sent to the server. Instead, different types of clients will lead to different patterns of generating segments (or packets) to be processed and transferred to the client. For the time being, we assume that the generation takes place according to a Poisson process.

The model is illustrated in Fig. 4. The left hand side of the model deals with connection setup, while the right hand side represents windowing and the slow start mechanism. The segments to be delivered to the client are generated by the transition `arr` and put in the unbounded buffer place `buffer`. If there is no connection between server and client yet (as indicated by a token in `no_conn`), transition `server` is disabled due to its enabling function and a connection setup is performed by firing transition `connect`. The connection will not be released until all segments in `buffer` have been delivered (inhibitor arc to `timeout`). Once the buffer is empty and all segments have been transferred (place `net` empty), the connection will be released after an additional delay, modeled by transition `timeout`.

Each segment in `buffer` is the result of some operation of

the server. In the HTTP case, the server has at least to perform some kind of database (or disk) access to retrieve the desired document. This overhead is modeled by the transition `server`.[1] Since connection-setup and server overhead have been accounted for now, segments are ready for transmission. A prerequisite for submitting a segment to the network is that the congestion window is large enough. The total size of the congestion window is reflected by the place `cwin`. It initially holds one token, representing the size of one segment. The available size of the congestion window which currently remains for transmission is represented by the place `tokens`. Each segment submitted to the network takes one token from this place. After the segment has been acknowledged successfully (represented by transitions `ack` and `ack_2`), the token is put back. Transition `ack_2` is enabled if the maximum allowed window size (`max_win`) has not been reached yet. In this case, *two* tokens are put back into place `tokens`, and one additional token is put into `cwin` to reflect the enlarged congestion window. Transition `ack` is only enabled if the maximum congestion window size has been reached. In this case, the congestion window is not enlarged any further, and just one token is put back into `tokens` (we do not account for a further linear increase as featured by the original slow start procedure). As a last possibility, a segment may be lost in the network, represented by transition `loss`. In that case, the segment is put back in the buffer (this implies that the corresponding lost segment experiences the server delay induced by `server` for a second time; for excessive packet loss ratios, the server rate has to be adjusted appropriately) and the congestion window size is reset to one by enabling the immediate transition `reset` (actually, the congestion window is only reduced by the size of tokens still available in the congestion window to limit the size of the model). This transition is also enabled if no connection exists, thus effectively initializing the congestion window for the next connection.

The exponentially distributed timing of transitions in the model surely is an approximation of the real-world system; it may be appropriate to model network delays, but timeout values usually have a less stochastic character. The impact of correctly modelling deterministic timeout values is, however, out of the scope of this paper; see e.g. [18] for an alternative to model connection release timeout values by Erlangian distributions.

Note that the firing rates of `ack`, `ack_2` and `loss` are proportional to the number of segments submitted to the network (place `net`). This approximation of an infinite-server behavior for network delays accounts for the fact that the increase of network load by submitting single segments is negligible. In future models, network delays could also be described more accurately, e.g. by using appropriate phase-type distributions as suggested in [19].

The most striking difference between this model and the real slow start mechanism is that the maximum congestion window is set to a constant size (occurring as `max_win` in the model), instead of setting it to half its value each time a segment is lost. Introducing an additional place for holding the current maximum window size would have led to a too large state space and has thus been omitted. Since the maximum window size can not shrink, the results obtained by analyzing the presented model provide an optimistic approximation.

[1] Note that this represents the overhead per segment of the server reply, not the overhead per client request. Therefore, per-request overhead has to be split between the number of segments resulting from this request.

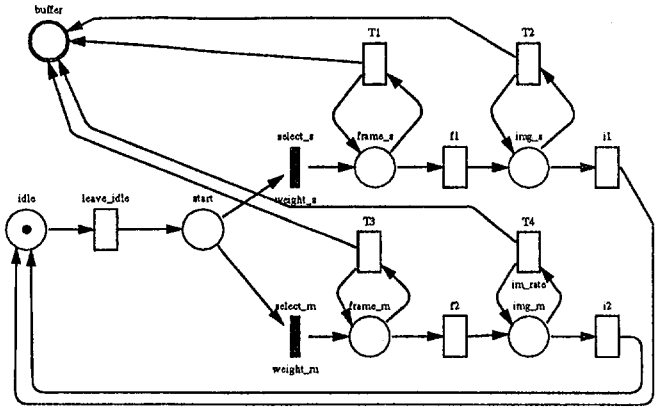

Fig. 5. Traffic generation model for HTTP.

C. Workload Models

As mentioned in the previous section, the slow start mechanism leads to poor bandwidth utilization when there is only a small amount of data to be transferred per TCP connection. In order to evaluate the merits of persistent connection approaches (like P-HTTP) in this case, an appropriate traffic model has to be developed. We focussed on modeling HTTP workload, using a characterization similar to the one presented in [3].

A user request for a web page is satisfied in two successive steps. First, the HTML document referenced by the URL is fetched from the server. Afterwards, all images referenced in this "frame" document are requested from the server. The traffic model must appropriately generate the segments which correspond to the server replies to these individual requests, and put them in the place `buffer` (Fig. 4).

The traffic model we propose is shown in Fig. 5. It is able to account for two different user request types (a small and a medium one, see next section for details). After an exponentially distributed user-idle time has passed (transition `leave_idle`), a probabilistic choice between small and medium request type takes place. Each request type consists of two phases, corresponding to generating segments to satisfy the frame request and segments belonging to the image requests. While the duration of each phase corresponds to the time needed to submit the corresponding request to the server, the rate at which segments are generated during that phase (transitions `T1`,`T2`,`T3` and `T4`) corresponds to the number of segments the server will send in reply to the request.

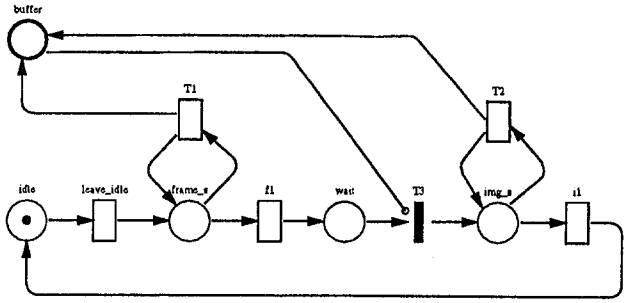

Fig. 6. Arrival model for one request type with blocking after submitting a frame request.

As an alternative to the arrival model shown in Fig. 5, we also investigate simplified versions of it, consisting of just one request type, simple IPP arrivals, and Poisson arrivals. Since these models do not account for the fact that prior to submitting an image request the reply to the preceding frame request has to be completed, we also investigate a blocking arrival model (see Fig.6). It deals with one request type only, where all frame segments have to be delivered before image segments are generated. This is realized by introducing an additional immediate transition which is disabled as long as there are any segments left in the server's outgoing buffer.

Server characteristics	
Computation time per request [s]	t_{comp}
Disk seek time [s]	t_{seek}
Disk transfer speed [B/s]	λ_{disk}
Network characteristics	
Round trip time [s]	t_{rtt}
Bandwidth [B/s]	λ_{bw}
Loss ratio	p_{loss}
Max. TCP segment size [B]	n_{MSS}
Connection release timeout [s]	t_{release}
Max. congestion window size [segs]	`max_cwin`
Workload characteristics	
User idle time [s]	t_{idle}
Small request probability	p_{small}
Small frame request size [B]	b_{sf}
Number of small image requests	n_{si}
Small image request sizes [B]	$b_{\text{si},1}, \ldots, b_{\text{si},n_{\text{si}}}$
Medium frame request size [B]	b_{mf}
Number of medium image requests	n_{mi}
Medium image request sizes [B]	$b_{\text{mi},1}, \ldots, b_{\text{mi},n_{\text{mi}}}$

TABLE I
OVERVIEW ON ALL MODEL PARAMETERS.

D. Parameterization

The parameters for the overall model can be split in three groups: server, network, and workload parameters (see Table I). The following paragraphs explain how the SPN transition rates can be derived from these.

D.1 Server Performance

Server-specific performance data is introduced in the model by the transition `server`. Its mean firing time corresponds to the average workload per segment of an answer to a client request. We assume that it is due to three parameters: computational effort per request, disk seek time per request and disk transfer time per segment. Per-segment workloads are obtained by dividing the per-request overheads by the mean number of segments per request, denoted by s_{mean} (derived in the workload parameterization section). In conclusion, the firing rate of transition `server` is given by

$$\left(\frac{t_{\text{seek}} + t_{\text{comp}}}{s_{\text{mean}}} + \frac{n_{\text{MSS}}}{\lambda_{\text{disk}}}\right)^{-1}.$$

D.2 Network parameters

Apart from the usual packet transfer latencies, a connection setup requires an additional round trip time to account for the TCP three-way handshake, so the rate of `connect` is t_{rtt}^{-1}. The rate of the connection release transition `timeout` is given by t_{release}^{-1}.

The time needed to acknowledge a segment sent from server to client consists of one round trip time plus the segment transfer time, which depends on segment size and bandwidth. Since we also consider packet losses, the rates of transitions `ack` and `ack_2` are given by $(t_{\text{rtt}} + n_{\text{MSS}}/\lambda_{\text{bw}})^{-1} \cdot (1 - p_{\text{loss}})$, and the rate of transition `loss` equals $(t_{\text{rtt}} + n_{\text{MSS}}/\lambda_{\text{bw}})^{-1} \cdot p_{\text{loss}}$.

D.3 Workload parameters

Due to the small size of client requests, we assume that the submission of a request takes on average one round trip time. Thus, the firing rates of transitions `f1` and `f2` equal t_{rtt}^{-1}. We also assume that all image requests are submitted simultaneously, thus again involving just one round trip time and yielding the rate t_{rtt}^{-1} for transitions `i1` and `i2` as well.

The number of segments to be put into the server's `buffer` place depends on the size of the reply corresponding to a request. For small requests, it is given by $s_{\text{sf}} = \lceil b_{\text{sf}}/n_{\text{MSS}} \rceil$ for the initial frame request. The corresponding total number of segments to be transferred in reply to the n_{si} image requests is $s_{\text{si}} = \sum_{k=1}^{n_{\text{si}}} \lceil b_{\text{si},k}/n_{\text{MSS}} \rceil$. These are the numbers of segments to be generated on average while a token is in places `frame_s` and `img_s`, so the rates of `T1` and `T2` are $s_{\text{sf}}/t_{\text{rtt}}$ and $s_{\text{si}}/t_{\text{rtt}}$, respectively. The parameters for the medium size request type can be computed similarly. Also, using n_{si}, n_{mi} and p_{small}, the mean number of segments per request can be computed as

$$s_{\text{mean}} = p_{\text{small}} \frac{s_{\text{sf}} + s_{\text{si}}}{1 + n_{\text{si}}} + (1 - p_{\text{small}}) \frac{s_{\text{mf}} + s_{\text{mi}}}{1 + n_{\text{mi}}}.$$

Parameter	National	International
t_{rtt}	0.020[s]	0.270[s]
p_{loss}	0.008	0.125
λ_{bw}	10^5 [B/s]	
n_{MSS}	536 [B]	

TABLE II
NETWORK PARAMETERS USED IN THE EXPERIMENTS.

IV. Performance Evaluation

The SPN model proposed in the previous section features a wealth of model parameters, and a large number of performance measures can be obtained from its analysis. We will thus keep many parameters constant throughout all experiments, and focus on the investigation of a few aspects only. We present the numerical results after describing the selected parameters in the next section. Finally, some information on the model's complexity and the computational solution effort are given.

A. Parameter selection

Since the following investigations focus on the variation of a few parameters only, many of the model parameters given in Table I are kept constant.

Server characteristics. We assume that the computation time per request is $t_{\text{comp}} = 0.01$[s]. The performance parameters of the disk are $t_{\text{seek}} = 0.01$[s] and $\lambda_{\text{disk}} = 5 \cdot 10^6$[B/s].

Network characteristics. Parameters concerning network performance have been taken from [19], where extensive Internet performance investigations have been accomplished. We selected two reference connections, a national one (RWTH Aachen to University of Karlsruhe, Germany) and an international connection (RWTH Aachen to Stanford University, U.S.A.). See Table II for the parameters of these connections.

If not mentioned otherwise, the connection release timeout t_{release} has been set to 10 seconds.

Workload characteristics. The parameters for small and medium request types occurring in the model's workload characterization have been taken from [3], representing the structure of some popular Web pages. The small request type consists of an initial 6651 byte frame page, referencing two images of size 3883 and 1866 bytes. Medium requests are formed by a 3220 byte frame and three images of size 57613, 2344 and 14190 bytes. We assume that the probability for a small request is $p_{\text{small}} = 0.6$.

B. Numerical Results

Our investigations focus on three main areas. We first investigate the impact of different workload models on the obtained performance measures. Here, we were also interested to which extent the maximum congestion window size (`max_cwin`) influences the performance characteristics.

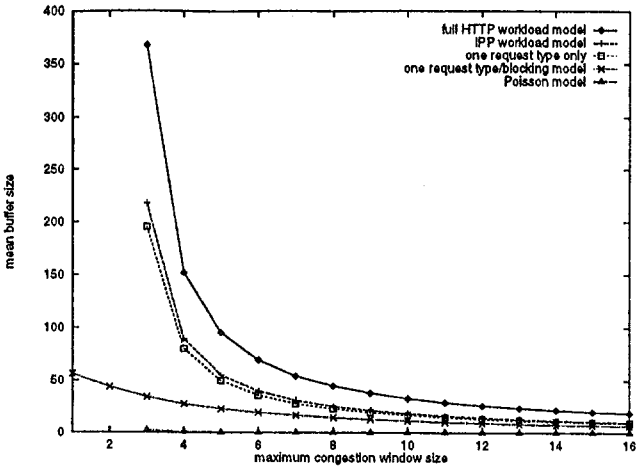

Fig. 7. Expected buffer size for the international connection and different workload models.

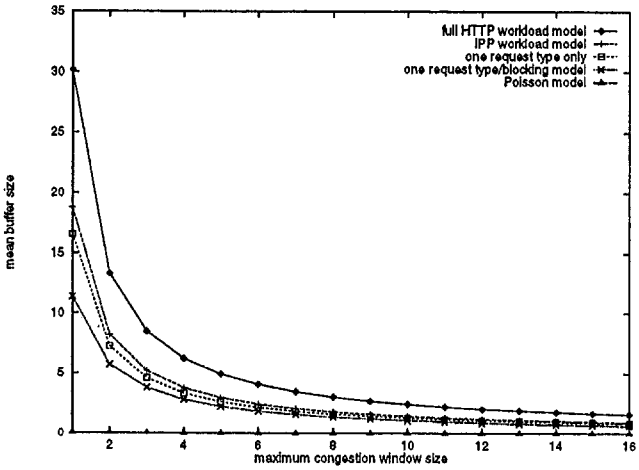

Fig. 8. Expected buffer size for the national connection and different workload models.

The connection release timeout plays an important role when dealing with protocols like persistent HTTP. We thus show how changing this parameter affects the properties of the overall system in the next experiment. Finally, the influence of the segment loss probability on the system has been investigated.

B.1 Workload models and congestion window size

As a first point, we were interested in how far detailed workload models influence the results of our investigation. Since the workload model greatly enlarges the number of states of the Markov chain underlying the SPN (e.g., the QBD level size of the arrival model shown in Fig. 5 is five times as large as a model with simple Poisson arrivals), it is interesting to see whether this effort pays off.

Fig. 7 shows the mean buffer size (number of tokens in place `buffer`) for different workload models. Clearly, increasing the maximum window size leads to a higher segment throughput, and thus reduces the buffer filling.

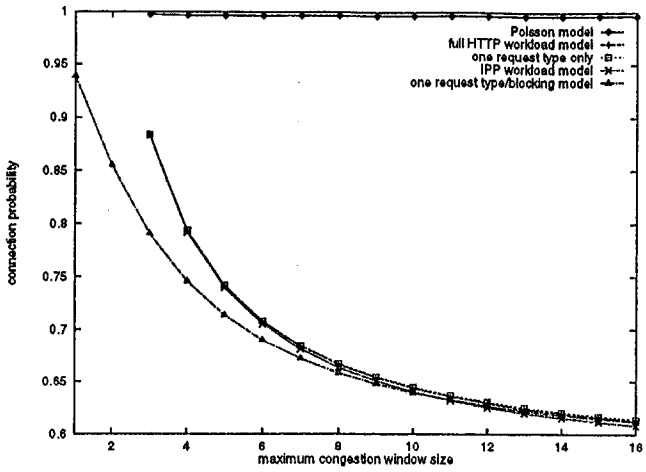

Fig. 9. Probability for an existing connection for different workload models.

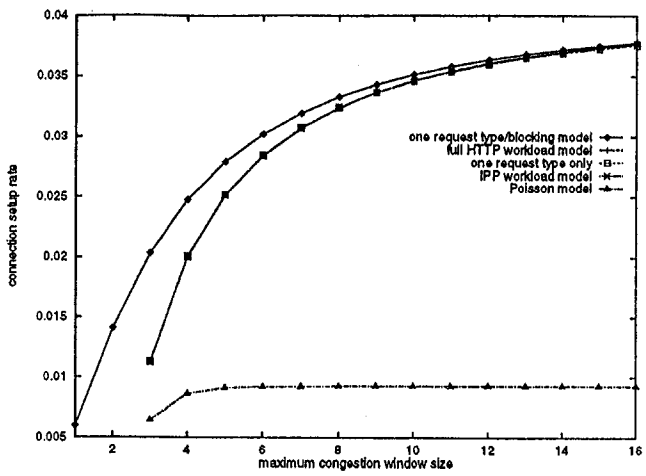

Fig. 10. Connection setup rates for different workload models.

Concerning the different workload models, the results for the full HTTP model (as shown in Fig. 5) differ significantly from those of the simplified versions. The results of the one-request and IPP workload models (with identical mean segment arrival rate) are very similar and still capture the qualitative behavior of the original model. However, due to ignoring the bursty arrival pattern accounted for by the other workload models, the approximation of the workload by a Poisson process leads to a dramatic underestimation of the expected buffer size. As an interesting point, the results for the blocking workload model (Fig. 6) do not differ too much from the non-blocking one-request-type model, especially for larger maximum connection window sizes. The absolute values are smaller, since the mean segment generation rate for this workload model is lower than for the other models (due to the additional waiting time in place `wait`).

Fig. 8 illustrates the same performance measures for the national Internet connection. Due to the lower round trip time, the average buffer filling is much lower than in the international case. Again, the Poisson workload model yields much too low results.

From now on focusing on the international connection type in all experiments, we investigate the steady-state probability for an existing connection (i.e., a token in place conn) in Fig. 9 . While the results for all bursty workload models coincide, the Poisson workload distributes segments much more in time, leading to a situation where the connection timeout almost never expires. Due to the increased usable bandwidth for larger maximum connection windows, the segments in the server's buffer are delivered quicker to the client, which leads to the connection to be released quicker. This results in lower connection probabilities for larger values of `max_cwin`.

Fig. 10 shows the connection setup rate for the different arrival models, which may represent a cost-factor. As can be observed, the setup rate increases for larger congestion window sizes, since requests are satisfied quicker and the

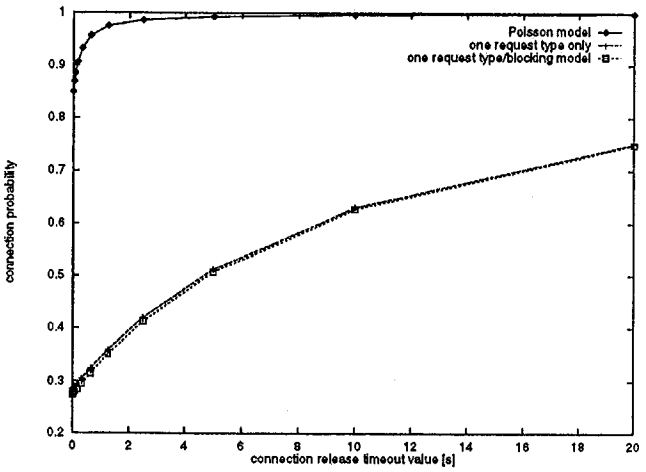

Fig. 11. Probability for an existing connection for different connection release timeouts.

release timeout expires more often in this case. Again, the Poisson model leads to significantly different results.

Summarizing, it can be said that ignoring the workload burstiness dramatically alters the performance measures obtained from the model, however, thanks to our modeling environment, bursty arrival patterns can easily be accounted for. Furthermore, increasing the maximum congestion window above a minimum value of about 8-10 segments leads to much better bandwidth utilization, thus reducing the buffer size and the time connections are held, albeit at the cost of increased connection setup rates. Though increasing the bandwidth utilization by higher maximum congestion window sizes is generally desirable, this may also increase network congestion and packet losses. However, this aspect can not be considered with the single client-server model presented here.

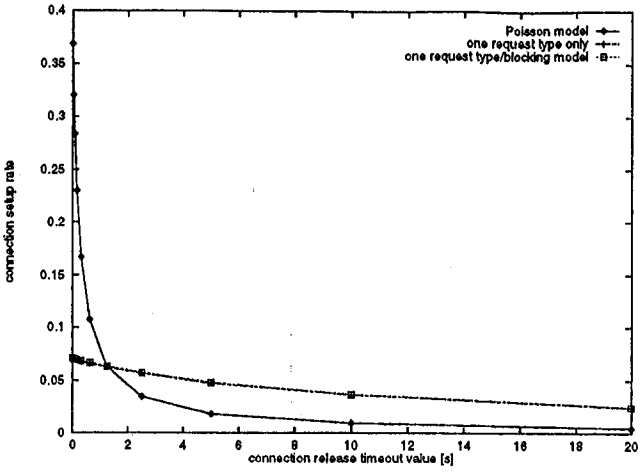

Fig. 12. Connection setup rates for different connection release timeouts.

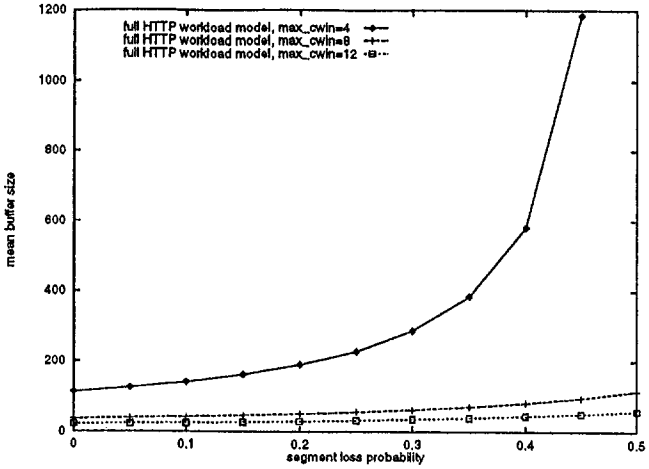

Fig. 13. Expected buffer size for different loss probabilities.

B.2 Influence of connection release timeout.

The introduction of a connection release timeout is crucial for the reduction of connection-setups and delays when protocols like P-HTTP are employed. Clearly, when choosing this parameter it is important to compare the gain of less connection setups with the higher costs imposed by maintaining a (mainly unused) connection.

Fig. 11 illustrates the probability of an existing connection for different values of t_{release} for a maximum connection window size of 12. Obviously, the probability increases for larger timeout values. Since the amount of data to be transferred remains constant, an existing connection is often unused. On the other hand, the connection setup rate decreases for larger timeout values, as illustrated in Fig. 12.

B.3 Influence of segment loss ratio

Apart from bandwidth and average round trip time, the packet loss probability heavily influences the performance of an Internet connection. The impact of losses on the windowing system is shown in Fig. 13 for different values of `max_cwin`. It can be observed that the mean buffer size of the system increases dramatically for high loss ratios if `max_cwin` is chosen too small. The system's behavior is much more robust concerning packet losses if the connection window size is large enough. This behavior is due to the fact that higher packet loss ratios effectively increase the amount of segments to be delivered by the network, since lost segments are re-submitted for transmission by transition `loss_done`. For a loss ratio of 0.5, every second packet has to be retransmitted. Since lost packets are again subject to loss in the next transmission try, the number of segments to be delivered effectively doubles. Since small values of `max_cwin` lead to small effective transmission performance, the buffer size is particularly sensitive to packet losses.

C. Computational effort

The Markov chain underlying the investigated SPN can grow remarkably large. For example, the SPN shown in Fig. 4 with the full workload model as shown in Fig. 5 leads to an underlying QBD process with 765 states *per level*. In some experiments (see e.g. Fig. 13), we obtained *mean* buffer sizes around 1000. Consequently, the evaluation of this system by using a large finite Markov chain would involve the investigation of several thousand levels, leading to a total number of several million states. While deriving steady-state measures of Markov chains of this size becomes a problem when using common numerical or simulation methods, the QBD-based solution approach leads to results in a quick and memory-efficient way.

We were able to accomplish the solution of the above-mentioned model in around 2.5 hours on a SUN SparcStation 20 clocked at 75 MHz. This time includes generating the state space, recognizing the QBD structure, solving the QBD Markov chain, and computing the desired performance measures. However, it should be noted that models of this size currently represent the upper limit we are able to solve due to numerical instabilities.

V. Conclusions

To our best knowledge, we presented the first performance evaluation study of the TCP slow-start mechanism under P-HTTP workload which is based on the numerical analysis of a SPN model. It has been shown that the choice of a reasonably high limit for the maximum congestion window is crucial for efficiently utilizing the communication infrastructure. This is especially true for connections with high packet losses.

We also demonstrated the strong influence of the employed workload model on the results of the system analysis. In particular, the approximation of bursty workloads by a Poisson process yields misleading results.

Our modeling environment, based on powerful yet user-friendly tools for using efficient numerical techniques and

hiding them behind a SPN-based interface proved to be of great use in the experiments. By employing QBD-based methods for the analysis of the underlying Markov chain we were able to derive numerical results much quicker than by conventional methods.

Concerning the application investigated here, far more extensive experiments are possible. We did not present the impact of varying the speed of the server or the impact of changing workloads and user behavior. Further investigations could also focus on derivations of the slow-start algorithm. Of course, while the results presented here are reasonable, the validation of (especially more detailed) models by measurements and simulations is an important topic.

Future work will also focus on the improvement of our modeling environment. In particular, the limit of about 1000 states in the repeating levels of the underlying QBD process is often a problem (for example, it led to the relatively small choice of the maximum congestion window parameter in the presented experiments). Future work will concentrate on the combination of sparse matrix computations with the spectral expansion solution method for QBD processes [10] to alleviate this problem. Furthermore, this method would also allow us to drop the first requirement on the infinite SPN place mentioned in Section II.

ACKNOWLEDGMENTS

This work has been supported by the doctorate program *computer science and technology* at the RWTH Aachen. We also thank the department of Prof. Hromkovic for donating some spare computing resources.

REFERENCES

[1] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Bernes-Lee, "Hypertext transfer protocol – HTTP/1.1," Internet Request for Comments RFC 2068, Jan. 1997.

[2] M.F. Arlitt and C.L. Williamson, "Internet Web Servers: Workload Characterization and Performance Implications," *IEEE/ACM Transactions on Networking*, vol. 5, no. 5, pp. 631–645, 1997.

[3] J. Heidemann, K. Obraczka, and J. Touch, "Modeling the performance of HTTP over several transport protocols," *IEEE/ACM Transactions on Networking*, vol. 5, no. 5, pp. 616–630, Oct. 1997.

[4] G. Ciardo, J. Muppala, and K. S. Trivedi, "SPNP: Stochastic Petri net package," in *Proceedings of the 3rd International Workshop on Petri Nets and Performance Models*. 1989, pp. 142–151, IEEE Computer Society Press.

[5] G. Florin and S. Natkin, "One place unbounded stochastic Petri nets: Ergodicity criteria and steady-state solution," *Journal of Systems and Software*, vol. 1, no. 2, pp. 103–115, 1986.

[6] M. F. Neuts, *Matrix Geometric Solutions in Stochastic Models: An Algorithmic Approach*, Johns Hopkins University Press, 1981.

[7] R. Nelson, "Matrix geometric solutions in Markov models: A mathematical tutorial," Research Report 16777, IBM, 1991.

[8] G. Latouche and V. Ramaswami, "A logarithmic reduction algorithm for quasi birth and death processes," *Journal of Applied Probability*, vol. 30, pp. 650–674, 1993.

[9] D. Wagner, V. Nauomov, and U. Krieger, "Analysis of a finite capacity multi-server delay-loss system with a general Markovian arrival process," in *Matrix-Analytic Methods in Stochastic Models*, S.S. Alfa and S. Chakravarthy, Eds. Marcel Dekker, 1995.

[10] I. Mitrani and R. Chakka, "Spectral expansion solution for class of Markov models: Application and comparison with the matrix-geometric method," *Performance Evaluation*, vol. 23, no. 3, pp. 241–260, September 1995.

[11] B. R. Haverkort and A. Ost, "Steady-state analysis of infinite stochastic Petri nets: A comparison between the spectral expansion and the matrix-geometric method," in *Proc. 7th International Workshop on Petri Nets and Performance Models*. 1997, pp. 36–45, IEEE Computer Society Press.

[12] B. R. Haverkort, "SPN2MGM: Tool support for matrix-geometric stochastic Petri nets," in *Proceedings of the 2nd International Computer Performance and Dependability Symposium*. 1996, pp. 219–228, IEEE Computer Society Press.

[13] SPN2MGM Web Page, `http://www.informatik.rwth-aachen.de/tools/spn2mgm.html`.

[14] K. Koischwitz, "Entwurf und Implementierung einer parametrisierbaren Benutzeroberfläche für hierarchische Netzmodelle," M.S. thesis, TU Berlin, Institut für Technische Informatik, 1996.

[15] V. Jacobson, "Congestion avoidance and control," in *Proc. ACM SIGCOMM '88*, 1988, vol. 18 of *Computer Communications Review*, pp. 314–329.

[16] R. Braden, "Extending TCP for transactions – concepts," Internet Request for Comments RFC 2068, Nov. 1992.

[17] B. C. Neuman, *The virtual system model: A scalable approach to organizing large systems*, Ph.D. thesis, University of Washington, Seattle, 1992, available at ftp://prospero.isi.edu/pub/papers/prospero/prospero-neuman-thesis.ps.Z.

[18] A. Ost, C. Frank, and B. R. Haverkort, "Untersuchungen zum Verbindungsmanagement bei Videoverkehr mit Matrix-geometrischen stochastischen Petrinetzen," in *Proc. 9th ITG/GI Workshop on Measurement, Modeling and Evaluation*. 1997, number 1 in Aktuelle Probleme der Informatik, pp. 71–85, VDE Verlag.

[19] A. Fasbender, *Messung und Modellierung der Dienstgüte paketvermittelnder Netze*, Ph.D. thesis, RWTH Aachen, 1998.