Analyzing a Proxy Cache Server Performance Model with the Probabilistic Model Checker PRISM [⋆]

Tamás Bérczes[1], tberczes@inf.unideb.hu,
Gábor Guta[2], Gabor.Guta@risc.uni-linz.ac.at,
Gábor Kusper[3], gkusper@aries.ektf.hu,
Wolfgang Schreiner[2], Wolfgang.Schreiner@risc.uni-linz.ac.at,
János Sztrik[1], jsztrik@inf.unideb.hu

[1] Faculty of Informatics, University of Debrecen, Hungary, http://www.inf.unideb.hu
[2] Research Institute for Symbolic Computation (RISC), Johannes Kepler University, Linz, Austria, http://www.risc.uni-linz.ac.at
[3] Esterházy Károly College, Eger, Hungary, http://www.ektf.hu

Abstract. We report our experience with formulating and analyzing in the probabilistic model checker PRISM a web server performance model with proxy cache server that was previously described in the literature in terms of classical queuing theory. By our work various ambiguities and deficiencies (also errors) are revealed; in particular, it is not clear how the reported paper simulates the network bandwidth, as a queue or a delay. To avoid such ambiguities we argue that nowadays performance modeling should make use of (at least be accompanied by) state machine descriptions such as those used by PRISM. On the one hand, this helps to more accurately describe the systems whose performance are to be modeled (by making hidden assumptions explicit) and give more useful information for the concrete implementation of these models (appropriate buffer sizes). On the other hand, since probabilistic model checkers such as PRISM are furthermore able to analyze such models automatically, analytical models can be validated by corresponding experiments which helps to increase the trust into the adequacy of these models and their real-world interpretation.

1 Introduction

The two originally distinct areas of the qualitative analysis (verification) and quantitative analysis (performance modeling) of computing systems have in the last decade started to converge by the arise of stochastic/probabilistic model checking [9]. This fact is recognized by both communities. While originally only individual authors hailed this convergence [7], today various conferences and workshops are intended to make both communities more aware of each others' achievements [4, 12]. One attempt towards this goal is to compare techniques and tools from both communities by concrete application studies. The present paper is aimed at exactly this direction.

In [1], we have shown how the probabilistic model checker PRISM [10, 8] compares favorably with a classical performance modeling environment for modeling and

[⋆] Supported by the Austrian-Hungarian Scientific/Technical Cooperation Contract HU 13/2007.

analyzing retrial queueing systems, especially with respect to the expressiveness of the models and the queries that can be performed on them. In the present paper, we are making one step forward by applying PRISM to re-assess various web server performance models with proxy cache servers that have been previously described and analyzed in the literature.

The starting point of our work is the paper [5], which presents a performance model for a system of a web server and web clients where a “proxy cache server” receives all the requests from the clients of a local network; with a certain probability the data requested by a client are already cached on the proxy server and can be returned without contacting the web server from which the data originate. The paper [5] is based on the seminal paper [11] which introduces a performance model of a web server. In [3], two of the authors of the present paper have further generalized this model by allowing the proxy cache server to receive also requests from external sources.

In this paper, we have constructed a formal model of the informal sketches in the language of PRISM [10]. This language essentially allows to construct in a modular manner a finite state transition system (thus modeling the qualitative aspects of the system) and to associate rates to the individual state transitions (thus modeling the quantitative aspects); the mathematical core of such a system is a Continuous Time Markov Chain (CTMC) which can be analyzed by the PRISM tool with respect to queries that are expressed in the language of Continuous Stochastic Logic (CSL) [9].

The remainder of this paper is structured as follows. In Section 2 we investigate the model described in [5]. First we implement it in PRISM and we try to reproduce their quantitative results. Here we only note that this article contains errors. We believe that this part is the most interesting one for the model checking community. In Section 3 we show how did we find the errors in the investigated paper by using PRISM. This section refers to our technical report [2], where more details are given. Section 4 summarizes our findings.

2 Performance Model of a Proxy Cache Server

The article [5] describes the model of a “proxy cache server” (PCS) to which the clients of a firm are connected such that web requests of the clients are first routed to the PCS. Referring to an illustration redrawn in Figure 1 the model can be described as follows:

Using proxy cache server, if any information or file is requested to be downloaded, first it is checked whether the document exists on the proxy cache server or not. (We denote the probability of this existence by p). If the document can be found on the PCS then its copy is immediately transferred to the user. In the opposite case the request will be sent to the remote web server. After the requested document arrived back to the PCS then a copy of it is delivered to the user.

The solid line in Fig 1. ($\lambda_1 = p * \lambda$) represents the traffic when the requested file is available on the PCS and can be delivered directly to the user. The $\lambda_2 = (1-p) * \lambda$ traffic depicted by dotted line, represents those requests which could not be served by the PCS, therefore these requests must be delivered from the remote web server.

If the size of the requested file is greater then the Web server’s output buffer it will start a looping process until the delivery of all requested file’s is completed. Let

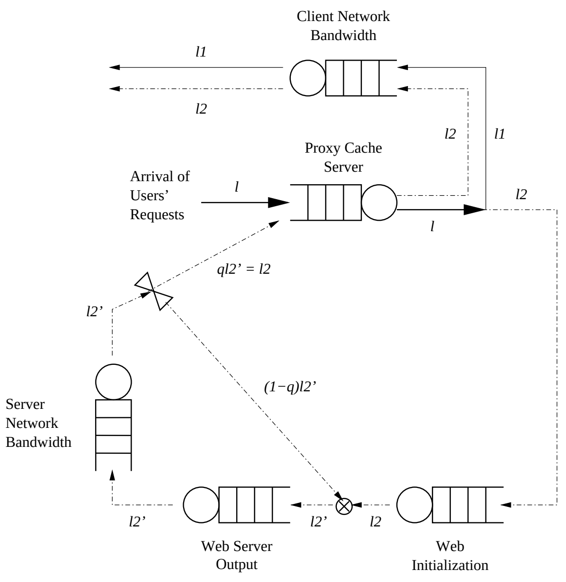

Fig. 1. Queueing Network Model of a Proxy Cache Server (redrawn from [5])

$q = \min\{1, (B_s/F)\}$ be the probability that the desired file can be delivered at the first attempt. Consequently, a $(1-q)$ proportion of the requests will loop back to the remote Web server for further processing.

In equilibrium, the traffic coming out of the remote Web server toward the PCS after branching should equal the original incoming traffic, λ_2. Hence $q\lambda_2'$ equals λ_2 where λ_2' is the traffic leaving server network bandwidth *before* branching.

The performance of the model is characterized by the parameters ($\alpha = \beta = \gamma = 1$):

Network Arrival Rate (λ)	
Average File Size ($F = 5000$)	
Buffer Size ($B_s = 2000$)	PCS buffer size ($B_{xc} = \alpha B_s$)
Initialization Time ($I_s = 0.004$)	PCS initialization time ($I_{xc} = \gamma I_s$)
Static Server Time ($Y_s = 0.000016$)	Static PCS time ($Y_{xc} = \beta Y_s$)
Dynamic Server Rate ($R_s = 1310720$)	Dynamic PCS rate ($R_{xc} = \beta R_s$)
Server Network Bandwidth ($N_s = 193000$)	
Client Network Bandwidth ($N_c = 16000$)	

The overall response time in the presence of the PCS is given as

$$T_{xc} = \frac{1}{\frac{1}{I_{xc}}-\lambda} + p\left\{ \frac{1}{\frac{1}{\frac{F}{B_{xc}}[Y_{xc}+\frac{B_{xc}}{R_{xc}}]}-\lambda_1} + \frac{F}{N_c} \right\}$$
$$+(1-p)\left\{ \frac{1}{\frac{1}{I_s}-\lambda_2} + \frac{1}{\frac{1}{\frac{F}{B_s}[Y_s+\frac{B_s}{R_s}]}-\lambda_2/q} + \frac{F}{N_s} + \frac{1}{\frac{1}{\frac{F}{B_{xc}}[Y_{xc}+\frac{B_{xc}}{R_{xc}}]}-\lambda_2} + \frac{F}{N_c} \right\}$$

In this formula, the first term denotes the lookup time to see if the desired files are available from the PCS, the second term (with factor p) describes the time for the content to be delivered to the requesting user, and the third term (with factor $1-p$) indicates the time required from the time the PCS initiates the fetching of the desired files to the time the PCS delivers a copy to the requesting user.

Furthermore, it is stated that without a PCS the model reduces to the special case

$$T = \frac{1}{\frac{1}{I_s} - \lambda} + \frac{1}{\frac{1}{\frac{F}{B_s}[Y_s + \frac{B_s}{R_s}]} - \lambda/q} + \frac{F}{N_s} + \frac{F}{N_c}$$

The response times for the PCS model with various arrival rates λ and probabilities p as well as the response time for the model without PCS, are depicted in Figure 2.

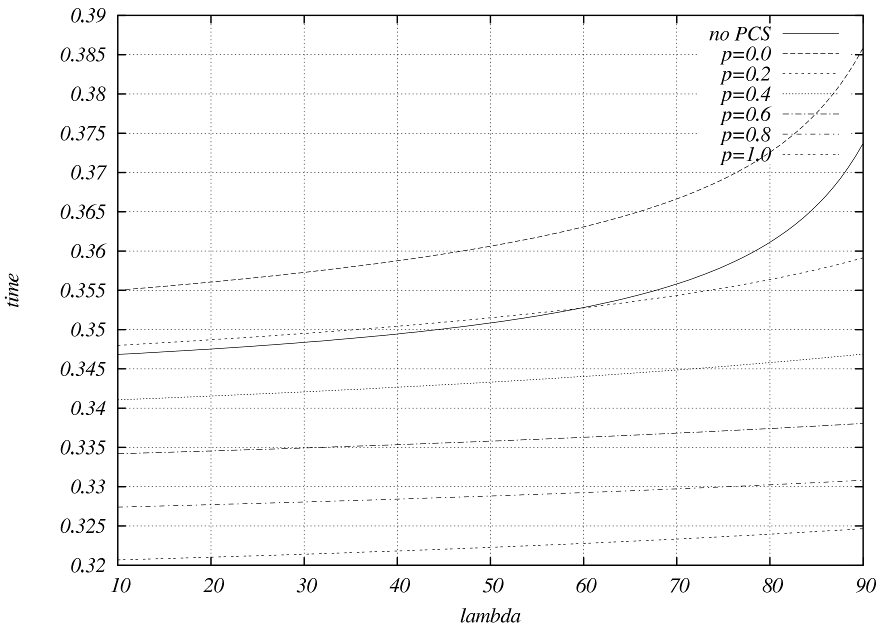

Fig. 2. Response Times With and Without PCS (Analytical Model)

We have reconstructed this system in PRISM, and we found out that actually the two equations (for T and T_{xc}) and also the visual model, i.e., Figure 1 are wrong. First we present the corrected equation for T_{xc}, the corrected visual model and the PRISM implementation of the corrected model. Only after this we tell how did we find those errors.

2.1 PRISM Implementation

First we present the corrections and the PRISM implementation of the corrected model, because we believe that this is the most interesting part of our work for the community.

Figure 3 shows the actual queueing network described in [5], but it is not the same as Figure 1, because the visual model in [5] contains errors. Figure 1 contains five queues, both the ‘server network bandwidth” and “client network bandwidth” are depicted as queues, although they are not treated as queues in the later analysis. Furthermore, it contains no queue to model the “client output”. More detailed description of this issue can be found in Section 3 and in our technical report [2].

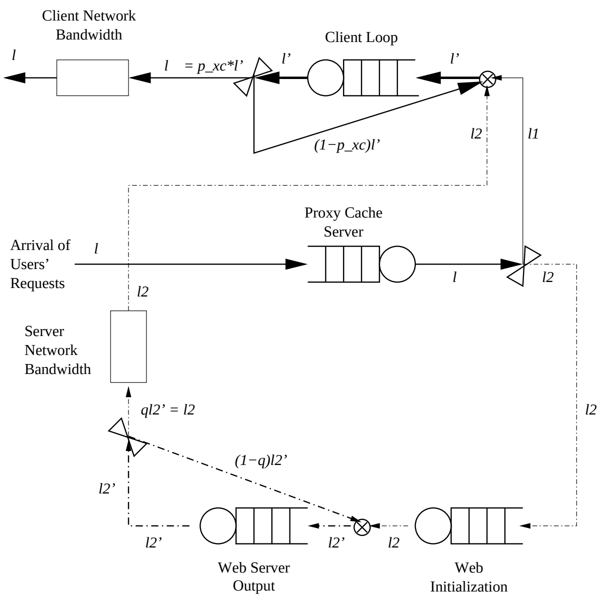

Fig. 3. Queueing Network Model of Proxy Cache Server

The equation for the overall response time is also wrong in [5]. The correct one is:

$$T'_{xc} = \frac{1}{\frac{1}{I_{xc}}-\lambda} + p\left\{\left(\frac{F}{B_{xc}}\right)\frac{1}{\frac{1}{Y_{xc}+\frac{B_{xc}}{R_{xc}}}-\lambda/p_{xc}} + \frac{F}{N_c}\right\}$$
$$+(1-p)\left\{\frac{1}{\frac{1}{I_s}-\lambda_2} + \left(\frac{F}{B_s}\right)\frac{1}{\frac{1}{Y_s+\frac{B_s}{R_s}}-\lambda_2/q} + \frac{F}{N_s} + \left(\frac{F}{B_{xc}}\right)\frac{1}{\frac{1}{Y_{xc}+\frac{B_{xc}}{R_{xc}}}-\lambda/p_{xc}} + \frac{F}{N_c}\right\}$$

where $p_{xc} = B_{xc}/F$ is the probability that the repetition loop is terminated. The corresponding numerical results are depicted in Figure 4.

The verbal descriptions (which is however correct in [5]) gives rise to the following PRISM code which introduces by the keyword `stochastic` a continuous time Markov chain (CTMC) model [9]:

```
stochastic
...
module jobs     // generate requests at rate lambda
  [accept] true -> lambda : true ;
endmodule
module PCS      // proxy cache server
  pxwaiting: [0..IP] init 0;
  pxaccepted: bool init true;
  [accept] pxwaiting = IP -> 1 : (pxaccepted' = false);
  [accept] pxwaiting < IP -> 1 :
    (pxaccepted' = true) & (pxwaiting' = pxwaiting+1);
  [sforward] (pxwaiting > 0) & (1-p > 0) -> (1/Ixc)*(1-p) :
    (pxwaiting' = pxwaiting-1);
  [panswer]  (pxwaiting > 0) & (p   > 0) -> (1/Ixc)*p :
    (pxwaiting' = pxwaiting-1);
endmodule
module S_C      // client queue
  icwaiting: [0..IC] init 0;
  [panswer] icwaiting < IC -> 1 : (icwaiting' = icwaiting+1);
  [sanswer] icwaiting < IC -> 1 : (icwaiting' = icwaiting+1);
  [done] (icwaiting > 0) & (pxc > 0) ->  1/(Yxc+Bxc/Rxc)*pxc :
    (icwaiting' = icwaiting-1);
endmodule
module S_I      // server arrival queue
  waiting: [0..IA] init 0;
  [sforward] waiting < IA -> 1 : (waiting' = waiting+1);
  [forward] waiting > 0 -> (1/Is) : (waiting' = waiting-1);
endmodule
module S_R      // server output queue
  irwaiting: [0..IR] init 0;
  [forward] irwaiting < IR -> 1 : (irwaiting' = irwaiting+1);
  [sanswer] (irwaiting > 0) & (q > 0) -> 1/(Ys+Bs/Rs)*q :
    (irwaiting' = irwaiting-1);
endmodule
```

The full code is given in [2] in Appendix B.2. The model consists of one process ("module") *jobs* generating requests and four processes *PCS*, S_C, S_I, and S_R. We describe them later in this section. Each process contains declarations of its state variables (bounded integers or booleans) and state transitions of form

```
  [label] guard -> rate : update ;
```

A transition is enabled to execute if its *guard* condition evaluates to true; it executes with a certain (exponentially distributed) *rate* and performs an *update* on its state variables. Transitions in different processes with the same *label* execute synchronously as a single combined transition whose rate is the product of the rates of the individual transitions.

Since a product of rates rarely makes sense in a model, it is a common technique to give all but one of the individual transitions the rate 1 and let the remaining transition alone determine the combined rate (we follow this practice in all our PRISM models).

Each node models a queue with a counter, which is the number of request in the queue, i.e., we make no distinction between requests, and each node has (generally) two transitions. One (or more) for receiving requests and one (or more) for serving requests. The first one increases the counter, the second one decreases it. If two queues, say A and B, are connected, i.e., a served request from A goes to B, then the server transaction of A and the receiver transaction of B have to be synchronous, i.e., they have the same label.

The rate of the server transactions has generally this shape: $1/t * p$, where t is the time for processing a request and p is the probability of the branch for which the transaction corresponds. Note that if t is a time, then $1/t$ is a rate. The rate of the receiver transactions are always 1 in this PRISM implementation, because product of rates rarely makes sense.

If a new request arrives and the queue is not full, i.e., its counter has not yet reached its upper bound, then the counter is increased and we can set an "acceptance" flag; otherwise, clear the flag (see "pxaccepted" in module *PCS*). We can use this flag to approximate the acceptance ratio of the queue.

Module *PCS* models the proxy cache server, module S_C the client, module S_I the initialization queue of the web server, module S_R the output queue of the web server with the following behavior:

- *PCS* returns with probability q an answer to the client (transition *canswer*) and forwards with probability $1-q$ the request to the server (transition *sforward*). The corresponding transitions "carry" the initialization time I_{xc} of the server.
- S_I buffers the incoming server request and forwards it after the initialization for further processing (transition *forward*); the transition carries the initialization time I_s of the server.
- S_R generates an output buffer with rate $1/(Y_s + \frac{B_s}{R_s})$ according to the model. However, since the request is repeated with probability $1-q$ (where $q = F/B_s$), the final result is only produced with probability q which contributes as a factor to the rate of the corresponding transition (transition *sanswer*).
- S_S models the repetition behavior of the client; a buffer of size B_{xc} is received from the PCS with rate $1/(Y_{px} + \frac{B_{xc}}{R_{xc}})$. However, the request for a buffer is repeated with probability $1 - p_{xc}$ such that only with probability p_{xc} the final buffer is received and the request is completed (transition *done*).

While it would be tempting to model the repetition in S_C by generating a new request for *PCS*, this is actually wrong, since such a repetition request is only triggered after the PCS has already received the complete file from the web server, it is not to be treated like the incoming requests (that with probability $1-p$ generate requests for the web server); rather we just consider the probability p_{xc} with which the *final* block is received from the PCS in the rate of the termination transition *done*.

If we could compute N, the number of requests in the system, and P, the probability that a request is "rejected" (i.e. dropped from the system because it encounters some full buffer), and λ, the arrival rate of the requests, then we could apply “Little’s Law” from queueing theory [6] to determine the average response time T for a request

$$T = \frac{N}{(1-P)\lambda}$$

Actually PRISM can be used to compute such quantitative properties of the model by using its reward system. Rewards have the form:

```
rewards "name of the reward"
  condition : numerical expression;
endrewards
```

This reward attaches to each state the value of the numerical expression where the condition is true. One can use the CSL query

```
R{"name of the reward"}=? [ S ]
```

to compute the long term average of this reward, where by operator R we introduce a reward-based property and by operator S we query the long-term average (”steady state value”) of this property.

Now we have to compute N and P (λ is given as a parameter of the model). For this we introduce the following rewards of the model:

```
rewards "pending"
  true : waiting + irwaiting + pxwaiting + icwaiting;
endrewards
rewards "time"
  true : (waiting + irwaiting + pxwaiting + icwaiting)/lambda;
endrewards
rewards "time0"
  true : (waiting + irwaiting + pxwaiting + icwaiting)/lambda
         + (FS/Nc) + (1-p)*(FS/Ns);
endrewards
rewards "accepted"
  pxaccepted: 1;
endrewards
```

Actually we can compute N by the reward “pending” because it assigns to every state the number of requests in the system.

It is difficult to compute P, the probability that a request is “rejected”, in PRISM. We can approximate it by using “accepted” flags, see the details in our technical report [2]. But if $P \simeq 0$ then $T \simeq \frac{N}{\lambda}$. We can compute this as it is done in the reward “time”. Only one step is remaining, we have to adjust this time with the delays of the network (the network bandwidth are simulated by delays in [5]) as it is done in the reward “time0”.

Using the CSL query

```
R{"time0"}=? [ S ]
```

we can compute the long term average of this value, which is the average response time. To be more correct, it is the average response time if the acceptance ratio for each queue is almost 1. In this paper we examine only the acceptance ratio of the PCS using the reward “accepted”.

2.2 Test Results

In the following, we present the results of analyzing our model in PRISM (choosing the Jacobi method for the solution of the equation systems and a relative termination epsilon of 10^{-4}; the analysis only takes a couple of seconds). As it turns out, it suffices to take the queue capacities $IP = 5, IC = 3, IA = IR = 1$ to keep the response times essentially invariant. With this configuration the model has 192 states and 656 transitions. Note that the actual value of p and λ do not effect the number of states and transitions.

Figure 5 gives the acceptance ratio for various arrival rates λ and proxy hit rates p; Figure 6 depicts the corresponding average number of requests N in the system. From this, we can estimate the total time a requests spends in the system (including the file transfer) as $N/\lambda + \frac{F}{N_c} + (1-p)\frac{F}{N_s}$, see Figure 7 and compare with the curve given from the equation of T'_{xc} in Figure 4. The results are virtually identical; only for arrival rates $\lambda > 70$ and $p = 0$, we can see differences (because the web server gets saturated and the request rejection rate starts to get significant).

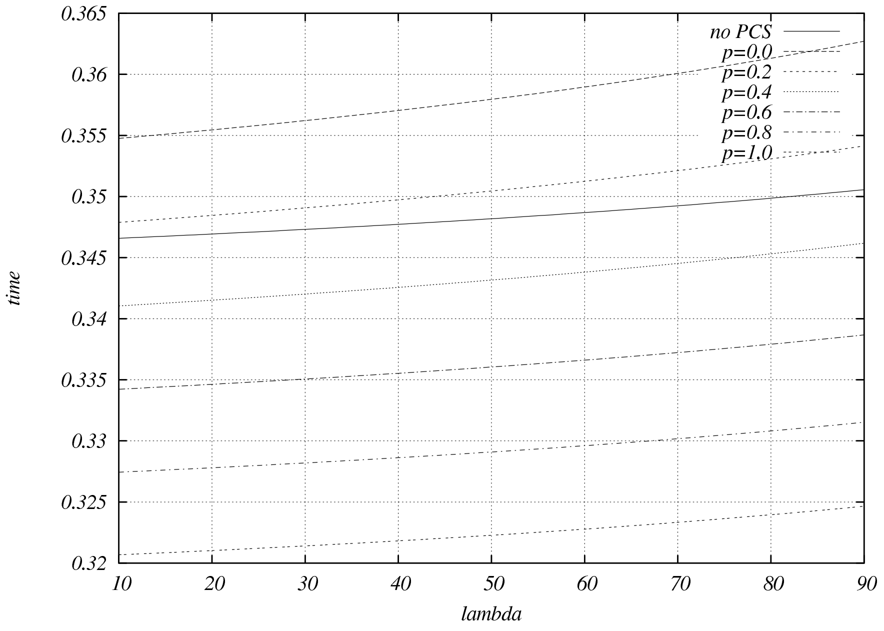

Fig. 4. Response Times With and Without PCS (Modified Analytical Model)

3 The Analytical Model Corrected

In this section we tell the "story" how could we find the errors in [5] with the help of PRISM. Variables with prime, like T', represent the corrected equations, variables with-

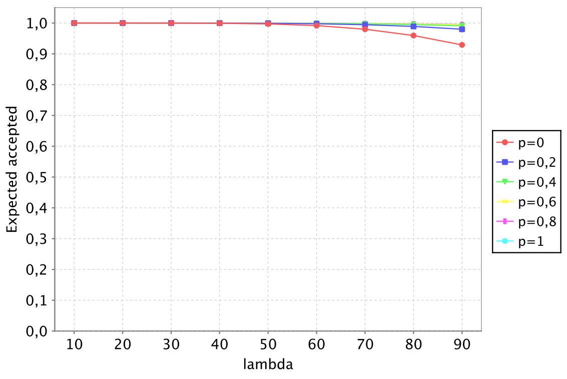

Fig. 5. Estimated Acceptance Ratio

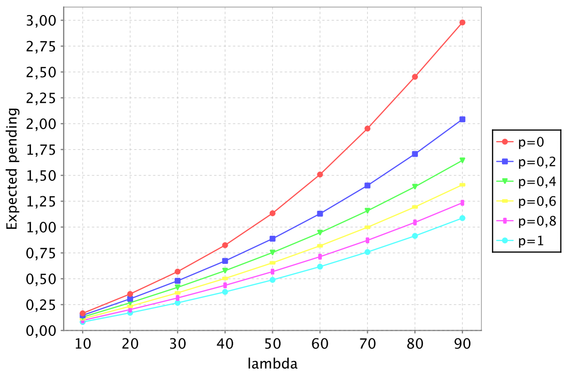

Fig. 6. Number of Pending Requests (N)

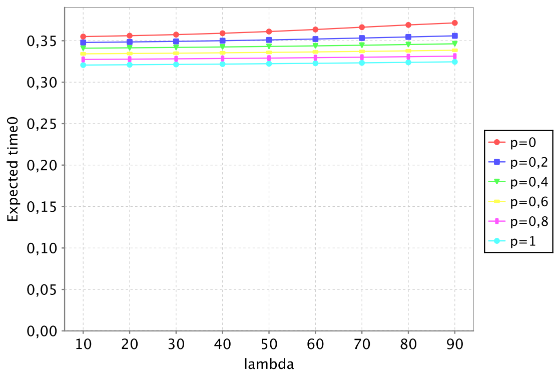

Fig. 7. Estimated Response Time $N/\lambda + \frac{F}{N_c} + (1-p)\frac{F}{N_s}$

out prime, like T, represent the original equations in [5], and variables with asterisks, like T^*, represent the original equations in [11].

3.1 The Model without PCS

It is claimed in [5] that the equation for T

$$T = \frac{1}{\frac{1}{I_s} - \lambda} + \frac{1}{\frac{1}{\frac{F}{B_s}[Y_s + \frac{B_s}{R_s}]} - \lambda/q} + \frac{F}{N_s} + \frac{F}{N_c}$$

represents the special case reported in [11], where T^* is given as

$$T^* = \frac{F}{N_c} + \frac{I_s}{1 - \lambda I_s} + \frac{F}{N_s - \lambda F} + \frac{F(B_s + R_s Y_s)}{B_s R_s - \lambda F(B_s + R_s Y_s)}$$

But this is actually *not* the case. In [5], the only term where the server bandwidth N_s plays a role is

$$\frac{F}{N_s}$$

which indicates the time for the transfer of the file over the server network. In [11], instead the term

$$\frac{F}{N_s - \lambda F}$$

is used which can be transformed to

$$\frac{1}{\frac{N_s}{F} - \lambda}$$

which indicates the time that a request spends in a queue with arrival rate λ and departure rate $\frac{N_s}{F}$. In other words, while [11] did not treat the client network as a queue, it nevertheless treated the server network as such. However, in [5], neither the client network nor the server network are treated as queues; they are just used to give additional time constants for file transfers.

The system without PCS can be modeled by the following PRISM implementation:

```
module jobs
  [accept] true -> lambda : true ;
endmodule
module S_I
  waiting: [0..IA] init 0;
  [accept] waiting < IA -> 1 : (waiting' = waiting+1) ;
  [forward] waiting > 0 -> (1/Is) : (waiting' = waiting-1) ;
endmodule
module S_R
  irwaiting: [0..IR] init 0;
  [forward] irwaiting < IR -> 1 : (irwaiting' = irwaiting+1) ;
  [done] (irwaiting > 0) & (q > 0) -> 1/(Ys+Bs/Rs)*q :
    (irwaiting' = irwaiting-1) ;
endmodule
```

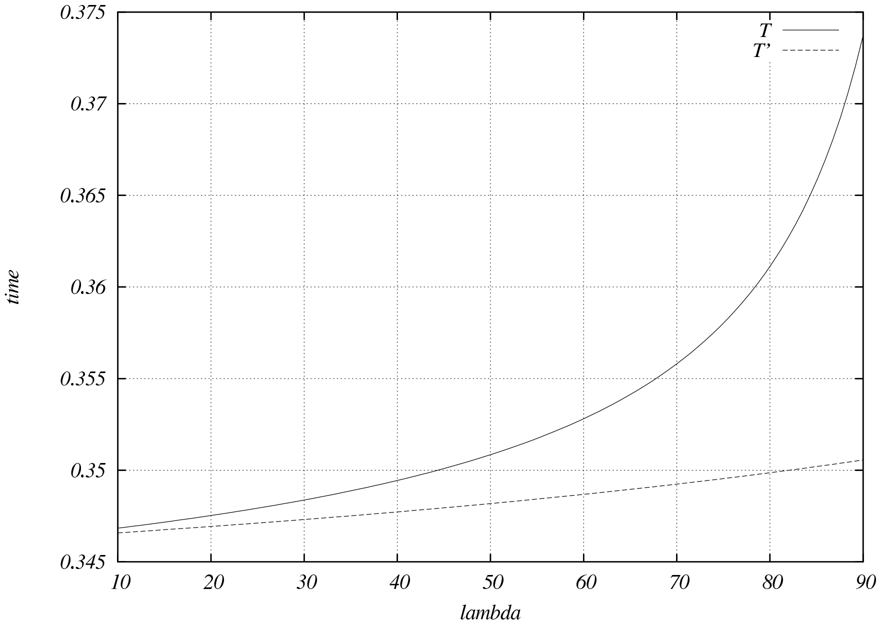

Fig. 8. Response Time Without PCS (Modified Analytical Model)

As it turns out, the numerical results produced by the analysis in PRISM do not accurately correspond to those depicted as "No PCS" in Figure 2, in particular for $\lambda \geq 50$. Actually the results are better described by the equation

$$T' = \frac{1}{\frac{1}{I_s} - \lambda} + \left(\frac{F}{B_s}\right) \frac{1}{\frac{1}{Y_s + \frac{B_s}{R_s}} - \lambda/q} + \frac{F}{N_s} + \frac{F}{N_c}$$

depicted in Figure 8 where the second term (modeling the "repetition loop" in the generation of the web server output) has been modified. Indeed, a closer inspection substantiates the correctness of this formulation: F/B_s represents the number of "iterations" of the corresponding queue which has arrival rate λ/q and departure rate $1/(Y_s + \frac{B_s}{R_s})$; this term now also equals the last term of the equation for T of [11]. (taking $q = \frac{B_s}{F}$).

Actually the same problem also affects the corresponding terms in the equation T_{xc}

$$\begin{aligned} T_{xc} = {} & \frac{1}{\frac{1}{I_{xc}} - \lambda} + p \left\{ \frac{1}{\frac{1}{\frac{F}{B_{xc}}[Y_{xc} + \frac{B_{xc}}{R_{xc}}]} - \lambda_1} + \frac{F}{N_c} \right\} \\ & + (1-p) \left\{ \frac{1}{\frac{1}{I_s} - \lambda_2} + \frac{1}{\frac{1}{\frac{F}{B_s}[Y_s + \frac{B_s}{R_s}]} - \lambda_2/q} + \frac{F}{N_s} + \frac{1}{\frac{1}{\frac{F}{B_{xc}}[Y_{xc} + \frac{B_{xc}}{R_{xc}}]} - \lambda_2} + \frac{F}{N_c} \right\} \end{aligned}$$

modeling repetition loops; the correct formulation apparently is:

$$
\begin{aligned}
T'_{xc} = {} & \frac{1}{\frac{1}{I_{xc}}-\lambda} + p\left\{\left(\frac{F}{B_{xc}}\right)\frac{1}{\frac{1}{Y_{xc}+\frac{B_{xc}}{R_{xc}}}-\lambda/p_{xc}} + \frac{F}{N_c}\right\} \\
& +(1-p)\left\{\frac{1}{\frac{1}{I_s}-\lambda_2} + \left(\frac{F}{B_s}\right)\frac{1}{\frac{1}{Y_s+\frac{B_s}{R_s}}-\lambda_2/q} + \frac{F}{N_s} + \left(\frac{F}{B_{xc}}\right)\frac{1}{\frac{1}{Y_{xc}+\frac{B_{xc}}{R_{xc}}}-\lambda/p_{xc}} + \frac{F}{N_c}\right\}
\end{aligned}
$$

where $p_{xc} = B_{xc}/F$ is the probability that the repetition loop is terminated (please note also the changes in the arrival rates of the corresponding terms). The corresponding numerical results are depicted in Figure 4, compare with the original results in Figure 2. However, here the difference plays only a minor role (for $p \geq 0.2$ only the third digit after the comma is affected).

3.2 The Model with PCS

Also in the model with PCS, the server network is not modeled by a queue but just by an additive constant for the transfer of the file over the network. This fact is made clear by rewriting the equation for the average response time as

$$
\begin{aligned}
T'_{xc} = {} & \frac{1}{\frac{1}{I_{xc}}-\lambda} + p\left\{\left(\frac{F}{B_{xc}}\right)\frac{1}{\frac{1}{Y_{xc}+\frac{B_{xc}}{R_{xc}}}-\lambda/p_{xc}}\right\} \\
& +(1-p)\left\{\frac{1}{\frac{1}{I_s}-\lambda_2} + \left(\frac{F}{B_s}\right)\frac{1}{\frac{1}{Y_s+\frac{B_s}{R_s}}-\lambda_2/q} + \left(\frac{F}{B_{xc}}\right)\frac{1}{\frac{1}{Y_{xc}+\frac{B_{xc}}{R_{xc}}}-\lambda/p_{xc}}\right\} \\
& +\left\{\frac{F}{N_c} + (1-p)\frac{F}{N_s}\right\}
\end{aligned}
$$

Here each fraction of form $\frac{1}{\mu-\lambda}$ indicates an occurrence of a queue with arrival rate λ and departure rate μ. We can see clearly that neither the server bandwidth N_s nor the client bandwidth N_c play a role in such fractions.

Figure 1 is therefore highly misleading; neither the server network bandwidth nor the client network bandwidth are in the model actually represented by queues; thus the queues labelled as “server network bandwidth” and “client network bandwidth” should be removed (i.e. replaced by other visual elements indicating simple delays).

Furthermore, similar to the “branching” discussed in Section 2.2 of [2] , the “branching” in this picture should not start after the “server network” but directly after the “web server output”, because the repetition rate of requests is not bounded by the network bandwidth in the model. To be more detailed, the server network bandwidth N_s (determining the processing rate of S_R) only shows up in the term $\frac{F}{N_s}$,i.e., it is only used to contribute to the time for the transfer of the file over the server network. If indeed, as suggested by Figure 1, after the transfer of every block the server would with probability q request the transfer of another block, the maximum transfer rate of blocks ($N_s/B_{xc} \simeq 96$) should also impose a limit on the number of “repetition” requests.

However, on the other side actually a queue is missing (also from the description in the text); this is the one that models the repeated requests for blocks of size B_{xc} which are sent by the clients to the PCS (analogous to the repeated requests for blocks of size B_s sent by the client to the web server in the basic web server model); therefore the client indeed needs to be modeled by a queue (whose output is redirected with probability $1 - p_{xc}$ to its input), but because of the looping process, not because of the client bandwidth.

Furthermore, the dotted arrow pointing to the input of the PCS queue is actually wrong; the corresponding requests do not flow to the PCS queue (where, since the queue cannot distinguish its inputs, they might generate new requests for the web server) but directly to the client queue.

Summarizing, the actual queueing network modeled in [5] contains only four nodes in contrast to the five ones shown in Figure 1 (no queue for modeling the server bandwidth) and one of these queues does not model the "client network bandwidth" but the repetition of block requests (it could be labelled in the figure as "client output" because it plays for the repetition the same role as the queue labeled "web server output").

Figure 3 shows a revised picture that describes the model as outlined above.

4 Conclusions

The work described in this paper seems to justify the following conclusions:

- The informal models used in the literature for the performance analysis of computing systems are sometimes ambiguous. This may lead to misunderstandings of other researchers that build on top of prior work; e.g., [5] describes their results as to be based on the model presented in [11], but actually [11] models the server network by a delay element rather than by a queue which gives different results in the performance evaluation.
- The use of diagrams of queue networks is an insufficient substitute for a formal specification of a system model and a constant source of pitfalls. In [11], the diagram depicts a queue where the actual performance model uses a constant delay; likewise [5] depict queues for the server network but also use delays in their analysis. Furthermore, in all three papers there is an apparent confusion of the roles of the "loop-back" arrows which are shown in the diagrams in places that are misleading with respect to the role that they actually play in the analyzed models.
- The paper [5] has errors in the analytical model; these errors were only detected after trying to reproduce the results with the PRISM models. This demonstrates that performance evaluation results published in the literature cannot be blindly trusted without further validation.
- Most important, after correcting the diagrams to match the actually analyzed models, a question mark has to be put on the adequacy of the models with respect to real implementations. The papers [11, 5] model the client network bandwidth outside the "loop" for the repeated transfer of blocks from the web (respectively proxy cache) server to the client. While the informal descriptions seem to suggest that this is intended to model the underlying network protocol, i.e. presumedly TCP, the "sliding windows" implementation of TCP lets the client interact with the server to control the flow of packets; this interaction is not handled in the presented

performance models (because then the network delay must be an element of the interaction loop).

- The PRISM modeling language can be quite conveniently used to describe queueing networks by representing every network node as an automaton ("module") with explicit (qualitative and quantitative) descriptions of the interactions between automata. This forces us to be much more precise about the system model, which may first look like a nuisance, but shows its advantage when we want to argue about the adequacy of the model.
- The major limitation of a PRISM model is that it can be only used to model finitely bounded queues, while typical performance models use infinite queues. However, by careful experiments with increasing queue sizes one may determine appropriate bounds where the finite models do not significantly differ from the infinite models any more. Furthermore, since actual implementations typically use (for performance reasons) finite buffers anyway, such models more adequately describe the real-world situation; the work performed for the analysis may be therefore used to determine appropriate bounds for the implementations and reason about the expected losses of requests for these bounds.

References

[1] T. Berczes, G. Guta, G. Kusper, W. Schreiner, and J. Sztrik. Comparing the Performance Modeling Environment MOSEL and the Probabilistic Model Checker PRISM for Modeling and Analyzing Retrial Queueing Systems. Technical Report 07-17, Research Institute for Symbolic Computation (RISC), Johannes Kepler University, Linz, Austria, 2007.

[2] T. Berczes, G. Guta, G. Kusper, W. Schreiner, and J. Sztrik. Analyzing Web Server Performance Models with the Probabilistic Model Checker PRISM. Technical report no. 08-17 in RISC Report Series, Johannes Kepler University Linz, Austria, 2008.

[3] T. Berczes and J. Sztrik. Performance Modeling of Proxy Cache Servers. Journal of Universal Computer Science, 12(9):1139–1153, 2006.

[4] M. Bernardo and J. Hillston, editors. Formal Methods for Performance Evaluation. 7th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2007. Lecture Notes in Computer Science, volume 4486, 2007.

[5] I. Bose and H. K. Cheng. Performance Models of a Firm's Proxy Cache Server. Decision Support Systems, 29:47–57, 2000.

[6] R. B. Cooper. Introduction to Queueing Theory. North Holland, 2nd edition, 1981.

[7] U. Herzog. Formal Methods for Performance Evaluation. Lecture Notes in Computer Science, 2090:1–37, 2001.

[8] A. Hinton, M. Z. Kwiatkowska, G. Norman, and D. Parker. PRISM: A Tool for Automatic Verification of Probabilistic Systems. Lecture Notes in Computer Science, Tools and Algorithms for the Construction and Analysis of Systems, 3920:441–444, 2006.

[9] G. Norman, M. Z. Kwiatkowska, and D. Parker. Stochastic Model Checking. Lecture Notes in Computer Science, Formal Methods for Performance Evaluation, 4486:220–270, 2007.

[10] PRISM—Probabilistic Symbolic Model Checker. www.prismmodelchecker.org. 2008.

[11] L. P. Slothouber. A Model of Web Server Performance. Proceedings of the 5th International World Wide Web Conference, 1996.

[12] K. Wolter, editor. Formal Methods and Stochastic Models for Performance Evaluation. Fourth European Performance Engineering Workshop, EPEW 2007. Lecture Notes in Computer Science, volume 4748, 2007.