Practical Applications of Stochastic Modelling

Editors: Jeremy Bradley and William Knottenbelt

PASM 2004

The Royal Society
London

4th September 2004

Preface

Recent years have seen not only significant theoretical advances in stochastic modelling and associated formalisms but also the increasing availability of high-powered computing facilities in academia and industry alike. The workshop on Practical Applications of Stochastic Modelling (PASM) was created out of a desire to harness these developments by encouraging the application of stochastic modelling formalisms to large-scale real-world systems.

We are, therefore, pleased to showcase in the very first workshop proceedings some excellent examples of how formalisms such as stochastic Petri nets, stochastic process algebras, queueing networks and SANs can be applied to diverse topical systems of practical interest such as peer-to-peer (P2P) file sharing networks, security protocols, electronic voting systems, parallel programming environments and RAID arrays.

It takes a great deal of effort and imagination to tackle the problems inherent in complex systems. We would like to thank everyone who submitted a paper – accepted or not – for taking on this challenge and for helping to contribute to a vibrant workshop.

We would also like to thank the programme committee who put in a stalwart effort to ensure a high quality refereeing process (with every submission receiving at least four reviews):

Marco Bernardo
Mario Bravetti
Gianfranco Ciardo
Tod Courtney
David Daly
Dan Deavours
Tony Field
Stephen Gilmore
Roberto Gorrieri
Boudewijn Haverkort
Graham Horton
Pieter Kritzinger
Gethin Norman
Mohamed Ould-Khaoua
Dave Parker
Nigel Thomas

Jeremy Bradley, William Knottenbelt
Program Co-chairs

Contents

Fluid Stochastic Petri Nets for Computing Transfer Time Distributions in Peer-to-Peer File Sharing Applications

R. Gaeta, M. Gribaudo, D. Manini, M. Sereno [1,2]

Dipartimento di Informatica
Università di Torino
Corso Svizzera, 185, 1049 Torino, Italia

Abstract

This paper presents an application of the Fluid Stochastic Petri Net (FSPN) formalism for the analysis of the transfer time distribution in peer-to-peer (P2P) file sharing applications.

The transfer of the resource follows a successful search; the transfer time is mainly dominated by network characteristics, application characteristics, resource characteristics, and user behavior.

The proposed analytical modeling technique accounts all these aspects and provides an estimation of the transfer time distribution after the search for a given resource has been performed.

Some numerical results are presented to prove the flexibility and the potential of the proposed technique.

Keywords: Peer-to-Peer Systems, File-Sharing Applications, Fluid Stochastic Petri Nets.

1 Introduction

File sharing applications are generating an increasing fraction of the traffic on the Internet. These applications are based on the peer-to-peer (P2P) paradigm where components of the application can act both as client and as server in requesting and providing a service. Several P2P file sharing applications have been developed so far, e.g., Napster, Gnutella, Freenet, Kazaa, etc.

[1] This work is partially supported by the Italian minister of Research (MIUR), within the project *Perf* (FIRB).

[2] Email: {rossano,marcog,manini,matteo}@di.unito.it

This is the workshop version of the paper. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

The peer acting as a *server*, accepts and forwards incoming queries for the search of resources (files), provides response to search queries, and serves requests when selected by clients. The peer acting as a *client*, alternates between the search of a resource and the transfer of the resource (downloading) from a server. Following a successful search, P2P applications provide the client a list of peers holding a copy of the requested resource; depending on the particular P2P application, additional information describing the peers is included such as bandwidth between the server and its Internet Service Provider (ISP), number of clients that are using this server, and other information the client may use for its server selection policy.

Both the search and the transfer phase may be time consuming. The search time is mainly influenced by architectural characteristics of the particular P2P application such as signaling, routing, searching protocols. By contrast, the transfer time of a resource is mainly dominated by network characteristics (last and first mile bandwidth [3], latency along the path connecting peers), application characteristics (the maximum number of allowed concurrent downloads and uploads), resource characteristics (the number of peers holding a copy of the requested resource as a function of its popularity, the size of the resource to be downloaded, the competing load on the selected peer during download), and user behavior (the selection criteria a user implements when multiple peers hold a copy of the requested resource).
In this paper we develop an analytic modeling technique for the analysis of P2P file sharing systems with the aim of providing Quality of Service (QoS) user-perceived measures related to the transfer phase for a given resource; in particular, we provide a method for the estimation of transfer time distribution. This analysis is both general (since it might be applied to different P2P file sharing architectures) and flexible enough to be adapted to the analysis of other P2P applications, e.g., streaming content distribution, information management for vehicular traffic.

We develop a hybrid modeling technique based on the combined use of Fluid Stochastic Petri Nets (FSPN) [12, 24] and combinatorial analysis. The joint use of different modeling paradigms allows us to capture several features that dominate the resource transfer time as well as to obtain an efficient model solution. Parameters of the models we develop are obtained from measured data on P2P applications presented in the literature.

The outline of the paper is the following: in Section 2 we briefly summarize the related work on the subject of analytical models of P2P systems including a description of measure based work we exploit to match parameters of the models we develop. Section 3 summarizes the peculiarities of the FSPN formalism. Section 4 discusses the main issues dominating the transfer time for a resource that are captured by our analytical modeling technique that is illustrated in detail in Section 5. Preliminary results of the model analysis are

[3] The bandwidth of channel connecting peers to ISPs.

presented in Section 6 while Section 7 draws conclusions and outlines several developments of the current work.

2 Background and Related Works

The P2P paradigm has recently captured scientific and academic researchers attention since the explosive growth of the number of file sharing application users that generate a large fraction of the nowadays Internet traffic. P2P-based applications pose challenging research problems related to reliability, scalability, resource organization, indexing, dimensioning, discovery and coordination in decentralized architectures.

Several measurement-based evaluations of P2P-based applications have been presented in the literature. The work in [19] is one of the first quantitative evaluations of P2P systems behavior and it is based on a "crawler" to extract the topology of Gnutella application level network and concludes that, although Gnutella is not a pure power-law network, its configuration has the benefits and drawbacks of a power-law structure, e.g., resilience to random peer failures.

In [22] a systematic characterization of P2P traffic and its impact on the underlying network is performed; a novel approach for conducting large scale measurements of P2P traffic for collecting data from multiple routers in a large ISP is presented. Three popular P2P systems are analyzed: FastTrack, Gnutella, and DirectConnect. The results reveal significant skew in the distribution of traffic across IP addresses, subnets, and autonomous systems. In [21] a study based on measurement supplies a precise characterization of end-user hosts that participate in two popular P2P file sharing system like Napster and Gnutella. This characterization takes into account the bottleneck bandwidth between these hosts, Internet at large, IP-level latencies to send packets to these hosts, how often hosts connect and disconnect from the system, how many files hosts share and download, and correlation between these aspects. The results of this study show that there is significant heterogeneity in peers bandwidth, availability, transfer rates, and peer behaviors.

The work in [10] presents a trace-based analysis of workload in the modern P2P multimedia file-sharing Kazaa. The results show that P2P file-sharing workloads are driven by considerably different processes than the classical Web applications. It is also demonstrated that there is significant locality in Kazaa workload, and therefore substantial opportunity for caching to reduce bandwidth consumption.

Model-based performance evaluation of P2P-based application is still in its infancy since at the moment there are few analytical models considering P2P networks. In most cases, analysis is carried out through extensive discrete event simulations.

An analytical model based on age-dependent branching processes is developed

in [25] where the service capacity of P2P file sharing applications is analyzed by considering a transient regime to characterize the ability of such systems to handle bursty traffic.
A mathematical model is deployed in [7] to explore and illustrate fundamental performances issues of P2P file-sharing systems. This model is applied in three different type of architecture (centralized indexing, distributed indexing with flooded queries, and distributed indexing with hashing directed queries), and it is used for analyzing important aspects regarding performance like system scaling, freeloaders, file popularity and availability.
In [13] a random-graph based model is introduced for studying the evolution of P2P communities such as Gnutella or Freenet; this model is used for analyzing basic properties such as reachability from a given node in the network. Another simulation approach is presented in [15] where a tool is developed that can simulate P2P networks on top of representative Internet topologies. In [11] a framework for an extensible and scalable P2P simulation environment that can be built on top of existing packet-level network simulators is developed.
In [14] the authors model the request-response process of file-sharing networks and obtain analytical expressions for three performance parameters (delay, jitter and loss probability) as a function of the number of resources indexes in the network and a time-out value for the queries.
The work in [3] is one of the few examples of the use of fluid models to analyze P2P-based applications. A fluid model for the performance analysis of the Squirrel cooperative cache system is proposed and studied. To cope with the large number of users that join and leave the cache system randomly, the request streams of the individual nodes are approximated by a fluid flow. The resulting stochastic fluid model turns out to be mathematically tractable, and provides a simple and low-complexity procedure for computing the hit probability.

3 The Fluid Stochastic Petri Net Formalism

Stochastic fluid models are a class of analytic models that have recently drawn the attention of many researchers for the performance evaluation of complex communication systems (see for instance [16]). Fluid variables may be used to approximate discrete variables to tackle the state space explosion problem that typically occurs when analyzing discrete-state based models.
In the simplest first-order fluid models the fluid flow rate is constant and a system of first-order differential equations is written to characterize the model [1]. This simple first-order model is extended to consider fluid-dependent flow rates in [6]. The efficient transient analysis method for first-order fluid models with constant rates based on power expansion methods is proposed in [23].

Fluid Stochastic Petri Nets (FSPN) were introduced in [24, 12] mainly with the aim of providing a feasible approximation to discrete state systems

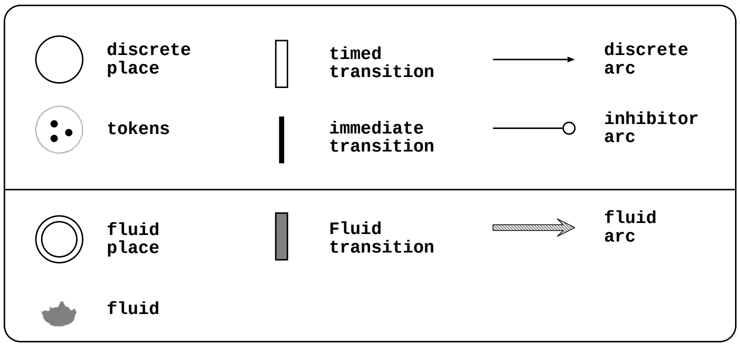


Fig. 1. The primitives of the FSPN formalism

in which the number of objects to be considered (customers, packets, tasks, workpieces etc.) tends to become too large to be treated with the usual discrete state approach common to Stochastic Petri Nets (SPN).
The basis of the model is to define two disjoint sets of places: the discrete places that carry a non-negative integer number of tokens and continuous places characterized by a real non-negative number. Continuous places can be physically interpreted as reservoirs, and the associated real number represents the fluid level in the reservoir. This is the reason for the name of FSPN assigned to this model. Fluid is generated by fluid transitions and flows along special fluid arcs. The flow rate can be a function of the token distribution in the discrete places and of the fluid level in the continuous places. Graphical representation of all the primitives of the formalism is depicted in Figure 1.
FSPNs can be solved in various ways. In [2, 8] simulative solutions for the FSPN formalism have been proposed. Analytical solutions instead compute the performance indices of the FSPN model by creating an appropriate Continuous Time Markov Chain (CTMC), on a mixed discrete and continuous state space. In this case the equations that must be solved to obtain either the steady state or the transient solution of the underlying stochastic process are partial differential equations. The solution of such systems of equations is not an easy task, and various numerical techniques, such as semi-discretization [24], have been proposed in the literature.

4 Modeling P2P Networks

We propose an analytic modeling technique for the estimation of the transfer time distribution, i.e., the phase starting after a successful search phase.

In general, for the resource transfer phase all P2P applications provide a list of peers holding a copy of the requested resource. In the following we denote the peer that requests the resource as the *client* and the peers holding a copy of the requested resource as the *servers*. For each server the P2P applications also provide additional information such as the bandwidth between the server and its ISP, the number of clients that are using this server, and other information that help the client to choose the server to download

Bandwidths	%
14.4 Kbps	4%
28.8 Kbps	1%
33.6 Kbps	1%
56 Kbps	23%
64 Kbps	3%
128 Kbps	3%
DSL	14%
Cable	44%
T1	5%
T3	2%

Table 1
Distribution of the user bandwidths (from [21])

the resource.

In the following we discuss the parameters that influence the transfer phase duration.

- *Network Characteristics*: the rate at which files can be downloaded from a server depends on the bottleneck bandwidth between the client and the chosen server, the available bandwidth, and the latency along the path connecting the two peers.

  The connection bandwidth between peers and ISPs has a clear impact on the transfer phase. In [21] measure-based results for the Napster and Gnutella P2P applications show that there is a significant amount of heterogeneity in bandwidth, latency, and other characteristics that vary several orders of magnitude across the peers of the system. Table 1 reports the percentages of the user bandwidths presented in [21].

  Our investigations are not related to a specific P2P application but they can be applied to different P2P file sharing systems. Nevertheless, we use results derived for Napster and Gnutella for setting some parameters in our models. This use of the results presented in [21] might be considered an improper extension but the bottleneck bandwidth results derived for Napster and Gnutella are quite similar even if the two P2P applications are based on different architectures. It is important to point out that the measures reported in Table 1, that we use as basis for our experiments, were presented in 2002 and these measures represent a reasonable "picture" of the last-mile connections at that time. The trend is towards high-speed bandwidth connections and then the results presented in this paper should be considered as a sort of worst-case analysis.

- *Application Characteristics*: after the client chooses a server (or more servers in case of parallel downloads) the resource bandwidth allocated to this (these) transfer(s) may change during the transfer phase. These bandwidth fluctuations are mainly due to the variation of the load on the chosen server and also depend on the maximum number of concurrent uploads/downloads. In general, P2P applications implement sharing bandwidth policies among the different clients that download resource(s): in some cases the server equally shares the available download bandwidth among the clients, in other cases the sharing policy depends on some parameters that account for the participation level of the client.

The speed at which the client downloads the requested file also depends on the possibility that the file can be downloaded in pieces or "chunks" from several different servers (for instance, Kazaa allows this possibility).

- *Resource characteristics*: the size of the resource to be downloaded has an obvious impact on the distribution of the transfer time. Measure-based analysis of P2P applications (see for instance [10, 20]) shows that that there is a substantial difference in typical resource size between P2P and WWW traffic. The measures presented in these papers show three prominent regions: *small* resources, typically MP3 files, that are less than 10 MBytes, *medium-size* resources, 10 to 100 MBytes that correspond to small-medium video files, and *large-size* resources over 100 MBytes, that correspond to large video files.

  The influence of the resource popularity on the transfer rate is quite clear. If the client is looking for a very popular resource, then the probability that a copy of this resource is held by a server with high speed connection bandwidth is higher than the case of a search for a "rare" resource. On the other hand the probability that a server holding a very popular resource is overloaded (because there are many clients that require its resource(s)) increases with the resource popularity.

- *User Behavior*: This issue is important when multiple servers hold a copy of the requested resource; in this case the user has to select the actual server from which downloading the resource. The choice could be based on the server with the fastest connection, on the server with the lowest load, or it could be a random selection.

5 The Modeling Technique

The modeling technique we develop aims to compute the distribution of the time required to download a file when the client (that we denote as *the tagged client*) behave as follows: after the search of a given resource (file), the client selects a server to transfer the resource and leaves the system as soon as the download terminates. The modeling technique does not take into account the search phase explicitly and assumes that the result of this search is the number of peers holding a copy of the resource.

5.1 Modeling Assumptions and System Parameters

The modeling technique we develop takes into account several system parameters that dominate the transfer time among those described in Section 4: network characteristics (last and first mile bandwidth), application characteristics (the maximum number of allowed concurrent downloads and uploads), resource characteristics (the number of peers holding a copy of the requested resource as a function of its popularity, the size of the resource to be downloaded), and the user behavior (the selection criteria a user implements when multiple peers hold a copy of the requested resource).

Furthermore the modeling technique has been developed under the following assumptions:

- The popularity of a resource is the number of peers holding a copy that we denote as N. We assume that all the N servers not only hold a copy of the resource but also are available for download, i.e., a request for download is neither queued nor refused.
- We do not consider peer availability issues, that is, we assume that the server is available for all the duration of the resource download time, and that the session ends only when client has completely downloaded the resource.
- The underlying IP network is not congested, i.e., network transfer times are dominated by first mile and last mile characteristics of the peers. This assumption could be dropped by adding slight modifications to the models we developed to account for variability in transmission times along the path from client to server.
- The offered bandwidth of a server is equal to its first mile bandwidth, i.e., during the upload of the resource the server does not perform downloading. On the other hand, a client, during the transfer phase, dedicates all its bandwidth to the download of the resource, i.e., during the transfer phase the client does not allow uploads.
- The download of a resource is not organized in parallel downloads of smaller chunks from different servers.
- A server does not discriminate among clients, i.e., it equally shares its offered bandwidth.
- We assume that some system parameters are function of the peer bandwidth. In particular, we assume the maximum number of concurrent uploads allowed by the server is defined as a function of its bandwidth. Furthermore, the average number of uploads arriving to a peer is assumed to be a function of its bandwidth as well. This assumption is based on the available measures published in [19, 21] that characterizes this system parameter as function of the peer bandwidth.

5.2 Definitions and Notation

This section introduces the notation we use. We use uppercase for both random variables and functions, values using lowercase, and sets using calligraphic style. In particular:

- $\mathcal{B} = \{14.4 \text{ Kbps},\ 28.8 \text{ Kbps},\ 33.6 \text{ Kbps},\ 56 \text{ Kbps},\ 64 \text{ Kbps},\ 128 \text{ Kbps},\ \text{DSL},\ \text{Cable},\ \text{T1},\ \text{T3}\}$ denotes the set of all possible available bandwidths connecting peers to their ISP;

- N denotes the popularity of given resource, i.e., the number of peers holding a copy of the resource; the range of N is the set of positive natural numbers;
- SB is a random variable denoting the bandwidth of the peer selected by the tagged client to download the requested resource; in Section 5.5 we propose a technique to compute its conditional distribution on N;
- CB denotes the tagged client bandwidth. The range of both SB and CB is the discrete set $\mathcal{B}$;
- S denotes the resource size (in Bytes);
- $K : \mathcal{B} \to I\!N$ denotes the maximum number of concurrent downloads allowed by the server;
- LT denotes the average number of requests of uploads arriving to a server, $W : \mathcal{B} \to [0, 1]$ denotes a bandwidth dependent weight, $L : \mathcal{B} \to I\!R$ denotes the average number of requests of uploads as function of the peer bandwidth, i.e., $L(b) = LT \cdot W(b)$.

5.3 The FSPN Model

The model we describe in this section refers to particular instancies of the variables CB, SB, and S that we denote as cb sb, and s respectively. The basis of our approach is the development of a FSPN model representing a server serving the request of the tagged client; the FSPN model captures the time evolution of the activity of the tagged client from the instant it starts its download until completion. It also represents the concurrent downloads interference by other client whose effect is to introduce fluctuations in the available bandwidth for the tagged client.

The server is represented by the FSPN model depicted in Figure 2. Timed transition request_arrival models the arrival of a request and its rate is equal to $L(sb)$. The sub-net composed by places CHOICE, STAGE_1, STAGE_2, END_SERVICE, immediate transitions choose_1, choose_2, terminate_service, and timed transitions service_1, service_2 models a two stage hyper-exponential service and for each tangible (discrete) state $\boldsymbol{m}_j$ we denote as I_j the sum of the tokens in places STAGE_1 and STAGE_2 that represents the requests that interfere with the tagged client service. Place AVAILABLE represents the number of requests that can be accommodate by the server. This model has a single P-semiflow that covers all the (discrete) place. The sum of tokens in these places is equal to $K(sb) - 1$. Fluid place TRANSFERRED represents the bytes transferred by the tagged client. Fluid transition transfer models the file transfer; its flow rate f is a function of the number of clients that are in service, i.e.,

$$f(I_j) = \min\left(\frac{sb}{I_j + 1}, cb\right) \quad \text{for each tangible (discrete) state } \boldsymbol{m}_j. \qquad (1)$$

The min function takes into account that the transfer rate is limited by the

lowest bandwidth; in this way the actual flow rate depends on the client bandwidth and on the available bandwidth of the server (which in turn, depends on the instantaneous number of peers using the server). Note that in the definition of f the tagged client is considered by adding one to the number of clients in the system ($I_j + 1$ in Equation (1)). In this way we consider only the first and the last mile bandwidths, neglecting the underlying network.
The choice of this particular modeling of the server activities stems from available measures [10] that provide statistics on the distribution of session times for downloads. In particular, this study highlights that session times exhibit high variability (because of the large heterogeneity in resource sizes); the hyper-exponential service time with infinite server policy is therefore the simplest choice to account for these phenomena.

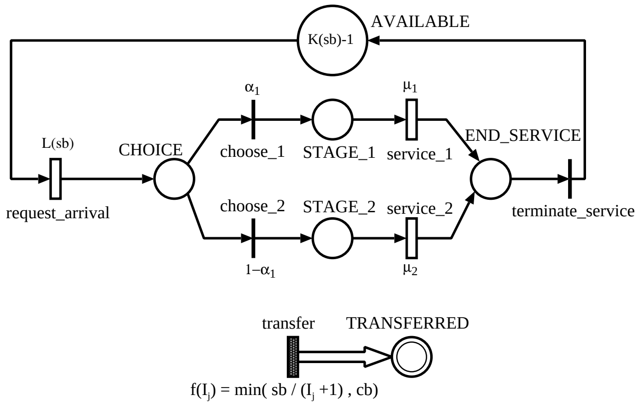


Fig. 2. FSPN model representation of a server for the computation of the transfer time distribution $F_t(t|sb, cb, s, \boldsymbol{\pi}_0)$.

Since we assume that the service rate of transition request_arrival and the initial marking of place AVAILABLE depend on server bandwidth sb, and the flow rate function f (Equation (1)) depends on both sb and the client bandwidth cb, the transient solution of the FSPN model can be considered function of these two input parameters.

The FSPN represented in Figure 2 is analyzed using the techniques described in [9, 12]. These techniques consider the discrete and the continuous part of the model separately. In particular the underlying Markov chain describing the discrete component of the model is obtained from the FSPN. Since the transition rates of the timed transitions that compose the model are constant, this underlying Markov chain can be characterized by a single constant matrix $\boldsymbol{Q}$. The fluid interaction is taken into account in a diagonal matrix $\boldsymbol{R}$ whose elements represent the actual flow rate in each discrete state, i.e., the value of function f defined in Equation (1) computed in each discrete state $\boldsymbol{m}_j$. Note that in this particular example the underlying stochastic process is a *Markov reward model* since only positive flow rates are present. For such kind of models, specialized and efficient techniques (see for example [4, 5, 17, 18]) exist. Nevertheless we choose to use the more general technique presented in

[12] because of its simplicity, and to allow possible future extensions that may require features not included in standard reward models, e.g., fluid dependent rates. In particular, we are planning to use fluid rates that depend on the continuous part to represent policies that account the percentage of the resource already downloaded.

If we denote as $\pi_j(\tau, x)$ the probability density of having x unit of fluid at time τ in discrete state $\boldsymbol{m}_j$, i.e., the probability that x bytes of a resource have been downloaded at time τ in state $\boldsymbol{m}_j$, then according to the results presented in [9], we can write:

$$\frac{\partial \boldsymbol{\pi}(\tau, x)}{\partial \tau} + \frac{\partial \boldsymbol{\pi}(\tau, x)}{\partial x}\boldsymbol{R} = \boldsymbol{\pi}\boldsymbol{Q}, \tag{2}$$

where $\boldsymbol{\pi}(\tau, x)$ is a vector whose components correspond to $\pi_j(\tau, x)$. The file transfer time is conditioned by the number of competing peers on the server when the transfer starts. In order to consider this effect we modified the model: we set the initial marking of the FSPN illustrated in Figure 2 as a model parameter, denoted as $\boldsymbol{\pi}_0$. The parameter $\boldsymbol{\pi}_0$ is a probability distribution on the state space of the FSPN (the component of vector $\boldsymbol{\pi}_0$ that corresponds to marking $\boldsymbol{m}_j$ is denoted as $\pi_0(\boldsymbol{m}_j)$).
Since $f(I_j) > 0$ for any state $\boldsymbol{m}_j$, and since the fluid place is unbounded, we do not need any boundary condition while the initial condition is:

$$\boldsymbol{\pi}(0, x) = \delta(x)\boldsymbol{\pi}_0,$$

where $\delta(x)$ is a Dirac delta. When considering the complete server model, the matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$, and the vector $\boldsymbol{\pi}_0$ depend on the model parameters. In particular, matrix $\boldsymbol{Q}$ and the initial probability vector $\boldsymbol{\pi}_0$ depend only on the selected server bandwidth sb, while matrix $\boldsymbol{R}$ depends on both the server and the client bandwidth.

With these assumptions we can denote the solution of Equation (2) as $\boldsymbol{\pi}(\tau, x, sb, cb, \boldsymbol{\pi}_0)$ for a given combination of parameters sb, cb and $\boldsymbol{\pi}_0$. The probability of having downloaded s bytes in less than t seconds is equal to the probability of having downloaded at least s bytes at time $\tau = t$, can be computed as:

$$F_t(t|sb, cb, s, \boldsymbol{\pi}_0) = \left. \int_s^\infty \bar{\pi}(\tau, x, sb, cb, \boldsymbol{\pi}_0) dx \right|_{\tau=t}, \tag{3}$$

where $\bar{\pi}(\tau, x, sb, cb, \boldsymbol{\pi}_0) = \boldsymbol{\pi}(\tau, x, sb, cb, \boldsymbol{\pi}_0)\mathbf{1}$, that is, $\bar{\pi}(\tau, x, sb, cb, \boldsymbol{\pi}_0)$ is the probability density of the fluid level regardless of the discrete state, and $\mathbf{1}$ is a vector of ones with a number of components equal to the number of discrete states of the model.

5.4 *Removing the dependency on the initial load*

The probability $F_t(t|sb, cb, s, \boldsymbol{\pi}_0)$ depends on the initial state of the server model (i.e., $\boldsymbol{\pi}_0$) when the tagged client starts its download. This dependency is crucial because the initial state can have a significant impact on the overall download time distribution, especially when considering short files. For instance, consider the time required to download a 112 KByte JPEG image from a DSL server, using a DSL connection. Figure 3 represents the distribution of the transfer time for different numbers of competing requests on the server when the tagged client starts the file transfer (the used model parameters are summarized in Table 3). In the case of long files the effect of the initial load is less evident since the discrete model reaches the steady state.
It is easily noted that the mean downloading time when no other peers are interfering with the file transfer is more than five times shorter than when there are 4 other peers accessing the server.

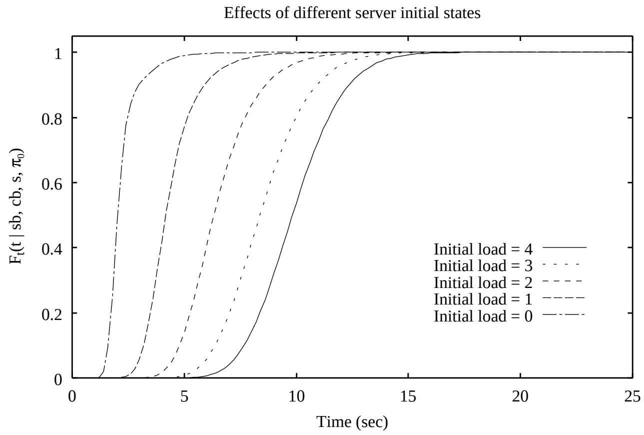


Fig. 3. Distribution of the transfer time as a function of the number of competing clients when the tagged client starts the file transfer.

To obtain the distribution of the transfer time removing the dependency on the initial state when the tagged client starts to download, we must correctly characterize the probability vector $\boldsymbol{\pi}_0$ for server whose bandwidth is sb. For the computation of $\boldsymbol{\pi}_0$ we consider the FSPN of Figure 2, without the fluid place where this time we do not distinguish the tagged client from the other clients. We compute the steady state distribution of this model and use it to determine $\boldsymbol{\pi}_0$.

We denote by $\hat{\boldsymbol{\pi}}$ the stationary distribution vector of the modified version of FSPN of Figure 2 given that the bandwidth of the server is equal to sb. When considering the complete model, the tagged client can be accepted only if there is at least a token in place AVAILABLE. We use this assumption to compute $\boldsymbol{\pi}_0$ by normalizing $\hat{\boldsymbol{\pi}}$, excluding the cases where the server would reject the tagged client request (that is, neglecting the states $\boldsymbol{m}_j$ such that $I_j = K(sb)$). In this manner the component of probability vector $\boldsymbol{\pi}_0$ that corresponds to the state $\boldsymbol{m}_j$ (denoted $\pi_0(\boldsymbol{m}_j)$) can be computed as

$$\pi_0(\boldsymbol{m}_j) = \frac{\hat{\pi}(\boldsymbol{m}_j)}{\displaystyle\sum_{\mathbf{m}_h : I_h < K(sb)} \hat{\pi}(\boldsymbol{m}_h)}. \tag{4}$$

Note that this derivation has been possible thanks to the assumption that the tagged client can only choose a server with an available position.

5.5 Modeling the server selection policy

When multiple servers hold a copy of the requested resource, the tagged client has to select the actual server from which downloading the resource. The available bandwidth of the selected server plays an important role in the distribution of the transfer time, especially when the tagged client has a fast connection. To show how the performance can be affected by the server selection, we consider the behavior of different file downloads versus the selected server bandwidth. We perform two experiments by considering two different values for the tagged client bandwidths: modem 56 Kbps and DSL. For each value, we compute the distribution of the file transfer time for different values of the server bandwidths: 33 Kbps, 56 Kbps, DSL, Cable and T3.
Figure 4 shows the results of this investigation for a resource size equal to 4 MBytes (the used model parameters are summarized in Table 3). In particular, when the client has a 56 Kbps bandwidth the distribution of the file transfer time is heavily conditioned by the bottleneck of the client connection. Furthermore, when the server uses a faster connection (DSL, Cable or T3) the performance cannot improve because the client bandwidth limits the speed at which data are transferred (the curves for DSL, Cable, and T3 are superposed). When the server uses the same (or lower) bandwidth (56 Kbps), the performance gets worse because the server becomes the bottleneck due to the simultaneous activity of other peers that use its bandwidth. This effect is amplified when the server has a 33 Kbps bandwidth.

In the case of a DSL bandwidth for the client, the performance are always influenced by the server bandwidth except for the T3 case.
If the server has the Cable or DSL bandwidth instead, the role of bottleneck depends on the probability of having other peers that are downloading simultaneously from the same server. A high number of concurrent downloads implies that most of the bandwidth is used, making the probability that the server becomes the bottleneck get higher.
In the case of lower bandwidths (56 Kbps or 33 Kbps) the server is always the bottleneck and the performance depends on its bandwidth.
We assume that after the search phase, the client selects the server with the highest bandwidth. We consider that the bandwidth of the peers holding a resource are i.i.d. random variables. For the distribution of the bandwidths we consider the one reported from Napster's users and described in Table 1.

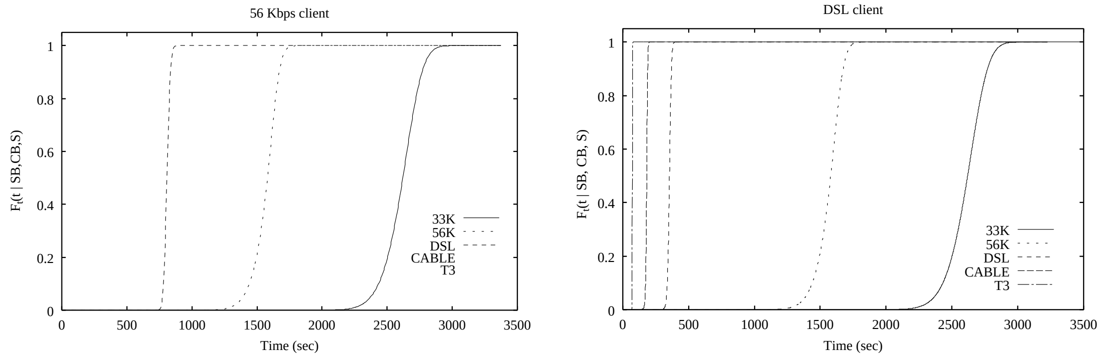


Fig. 4. The effect of server bandwidth on the distribution of the transfer time (4 MBytes resource size) for a modem 56 Kbps client (left graph) and for a DSL client (right graph).

According to this distribution, we define a combinatorial manipulation that models the selection of the peer with the higher bandwidth. Our model assumes that when the resource is available on only one server, the probability $P(SB = sb)$ is exactly the one reported in Table 1. When the number of available resources is greater than one, the probability to select a peer with higher bandwidth grows according to the bandwidth distribution of the peers that are present in the system. The greater the number of available resources, the greater the probability to select a peer with the highest bandwidth. When the number of available resources is large enough, the probability of finding the resource on the highest bandwidth peers tends to 1. The probability of finding a resource on a peer with a given bandwidth, using the fastest-connection policy, can be computed using the following recurrence relation:

$$
\begin{aligned}
P(SB = sb|N = n) = {} & P(SB \leq sb|N = n-1)P(SB = sb) + \\
& + P(SB = sb|N = n-1)P(SB < sb),
\end{aligned} \tag{5}
$$

where

$$
P(SB \leq sb|N = n-1) = \begin{cases} \displaystyle\sum_{\hat{b} \leq sb} P(SB = \hat{b}|N = n-1) & \text{if } n > 1, \\ 1 & \text{if } n = 1, \end{cases}
$$

$$
P(SB = sb|N = n-1) = 0 \qquad \text{if } n = 1,
$$

and

$$
P(SB < sb) = \sum_{\hat{b} < sb} P(SB = \hat{b}).
$$

The meaning of this formula is the following: the probability of selecting a peer with bandwidth $SB = sb$ given that there are n available resources is equal to the probability of having $n - 1$ peers with bandwidth less than or equal to sb and to find a peer with bandwidth equal to sb, or it is equal to the probability of finding a peer with bandwidth less than sb but to have a peer with bandwidth sb in the previous $n - 1$ resources.

5.6 The complete modeling technique

We can then summarize the steps required to compute the distribution of the transfer time using the methodology we propose:

- compute the probability distribution of random variable sb, i.e., $P(SB|N)$, as discussed in Section 5.5;
- compute the probability vector $\boldsymbol{\pi}_0$ as described in Section 5.4. Since $\boldsymbol{\pi}_0$ depends on the server bandwidth sb, to point out this dependency we write $\boldsymbol{\pi}_0(sb)$.
- compute $F_t(t|sb, cb, s, \boldsymbol{\pi}_0(sb))$ as described in Section 5.3.

We can then compute the cumulative distribution for the transfer time as

$$F(t|N = n, CB = cb, S = s) = \tag{6}$$
$$= \sum_{sb \in \mathcal{B}} F_t(t|SB = sb, CB = cb, S = s, \boldsymbol{\pi}_0(SB = sb))P(SB = sb|N = n).$$

6 Numerical Experiments

In this section we present some numerical results that illustrate the potentials of the proposed methodology. Extensive validation of the model results for our modeling technique shares the same difficulty of previous works on analytical models for P2P systems [7, 13]. Nevertheless, we performed (not shown) simple validations by comparing the model results in selected cases where theoretical results are known or can be exactly computed. In particular, we compared the model results with the ideal case where there is no competition for the server bandwidth and the transfer is only conditioned by the minimum bandwidth between server and clients; in these cases we found a perfect agreement between the model predictions and the theoretical results.

All the experiments have been performed by using a Pentium IV (2.4 Ghz) computer, with 1.5 GB of RAM, under Linux OS. In all the cases the model solution required few minutes (ranging from 10 to 60).

In the following we present numerical experiments derived for two different scenarios. These scenarios have two different distributions of the user bandwidths. For the scenario n. 1 we assume that the user bandwidth distribution is the one presented in Table 1. In this case, the user bandwidth distribution is derived from measure-based results presented in [21]. These results have been presented in 2002 and then they can be considered a reasonable “picture” of the last-mile connections at that time. For the scenario n. 2 we assume that the user bandwidth distribution is the one presented in Table 2. This scenario aims to investigate the increasing of number of peers with high-speed bandwidth connections.

For both scenarios Table 3 summarizes the model parameters that we derived from existing results [10, 19, 21] and that we use to perform the

Bandwidths	%
56 Kbps	6%
DSL	23%
Cable	64%
T1	5%
T3	2%

Table 2
Distribution of the user bandwidths for the scenario n. 2

Bandwidths	K(sb)	W(sb)
14.4 Kbps	1	0.21
28.8 Kbps	1	0.21
33.6 Kbps	2	0.21
56 Kbps	2	0.21
64 Kbps	3	0.21
128 Kbps	4	0.72
DSL	5	0.72
Cable	6	0.72
T1	6	0.07
T3	7	0.07

Service parameters	
μ_1	0.001
μ_2	0.1
α_1	0.6
α_2	0.4
Arrival rate	
LT	7

Table 3
Model parameters used for the numerical experiments

cb	n	Scenario n. 1		Scenario n. 2	
		50^{th} quantile	90^{th} quantile	50^{th} quantile	90^{th} quantile
56 Kbps	1	13m 48s	24m 13s	13m 43s	14m 10s
DSL	1	3m 03s	24m 2s	1m 42s	1m 56s
56 Kbps	30	13m 41s	14m 11s	13m 41s	14m 11s
DSL	30	1m 30s	3m 1s	1m 30s	3m 1s
56 Kbps	90	13m 41s	14m 11s	13m 41s	14m 11s
DSL	90	1m 12s	2m 20s	1m 12s	2m 20s

Table 4
Model results for the two scenarios, for different client bandwidths and different values of popularity

analysis presented in this section and in Sections 5.3 and 5.5.

We considered a 4 MBytes resource size which is a typical size for a MP3 file; we consider three different values for the popularity of the requested resource (1, 30, and 90 servers are available to provide the resource) and then we compute the distribution of the transfer time for two different client bandwidths: modem 56 Kbps and DSL.

The case of $n = 1$ considers a not-widespread resource; in this case, the client performs a random choice when selecting a server using the probability distributions presented in Tables 1 and 2 for the scenario n. 1 and scenario n. 2, respectively. For $n = 30$ and $n = 90$ the client selects a server according to the distribution obtained by using the recurrence Equation (5).
Table 4 reports the values of the 50^{th} and the 90^{th} quantiles computed from the model results. It is interesting to note that the transfer time distribution for high popularity resources is insensitive to the bandwidth distribution; this

phenomenon is easily explained by considering that for $n = 30$ and $n = 90$ the probability of selecting a server whose bandwidth is larger than 56Kbps or DSL is almost equal to 1. This implies that almost surely the transfer time distribution depends only on the bandwidth of the client.
On the other hand, the transfer time distribution for uncommon resources ($n = 1$) heavily depends on the bandwidth distribution; it is easily noted from Table 4 and Figure 5 that the scenario where the fraction of large bandwidth users is increased (scenario n.2) provides an impressive reduction of the value of the 90^{th} quantile for both a 56Kbps and DSL clients (this is because the probability of selecting a server whose bandwidth is larger than or equal to 56Kbps or DSL is almost equal to 1).
For a DSL client, an almost 50% reduction in the value of the 50^{th} quantile is also observed. This phenomenon can be explained by noting that improvements for a DSL client are more remarkable since the proability that this type of client selects a lower bandwidth server is markedly decreased in scenario n.2.

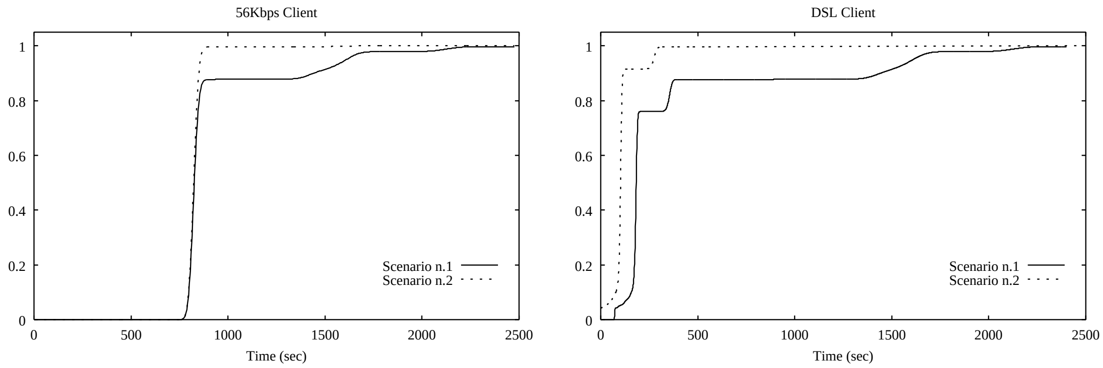


Fig. 5. Cumulative distributions of the transfer time for $n = 1$ in the two different scenarios (cb=56Kbps left plot, cb=DSL right plot)

7 Conclusions and Further Developments

In this paper we propose an analytical modeling technique for the evaluation of transfer times in file sharing P2P applications. The technique we present allows us to provide QoS user-perceived measures related to the transfer phase for a particular resource, i.e., the cumulative distribution of the transfer time.

Although the technique has been developed under several simplifying assumptions the results we derive for the analyzed scenarios highlight interesting issues that involve the relations among the parameters governing the duration of the transfer phase.

Several different extensions are currently underway. These extensions and improvements can be classified into three different categories:

(i) extensions to remove some of the simplifying assumptions in order to obtain a modeling technique more adherent to the studied systems;

(ii) developments of solution techniques to improve the efficiency and/or the accuracy of the numerical solution of the FSPN model;

(iii) modeling of more complex P2P file sharing issues as well as extensions of our technique to address additional P2P-based applications besides the classical file sharing.

For the first category we are currently working

- to obtain a more sophisticated representation of issues related to the popularity of a given resource. In fact, in this paper we consider the number of peers holding a copy of the requested resource as an input parameter n that can be arbitrarily varied to perform a what-if analysis of the transfer times distribution.
- to capture peer availability issues: in particular, we will try to model peers (servers as well as clients) that exhibit transient behaviors.
- to represent more detailed peer behavior: in this case we are trying to model servers that may also behave as a client during the transfer phase. In this manner we can have a more realistic representation of the bandwidth sharing among downloads and uploads. The same effort is devoted to the representation of the client behavior;
- to account for P2P applications that allow parallel downloads of smaller resource chunks from different servers;
- to model of selection strategies different from the criterion of choosing the server having the largest bandwidth connections, e.g., selection criteria based on the load of the server.

To account some of the previous issues, we can use a modified version of the FSPN model. In particular, peer availability issues could be captured by means of a sub-net that represents an "on-off" behavior of the server. In the off state the server is not available for serving requests. We can also modify the FSPN to account more detailed peer behavior, for instance by adding to the FSPN depicted in Figure 2 a sub-net that is similar to the discrete part of the model we can model peer that behaves both as server and client. In this case the rate of fluid transition that models the speed at which the tagged client downloads the requested resource has to be modified to account that the server bandwidth is used for both downloads and uploads.

In the second category of extensions, fall all the improvements related to the transient solution of the FSPN model. In particular, we are investigating the use of different transient solution methods, e.g., [23].

The last category of extensions includes the use of the modeling technique to derive an optimization model that can be employed for designing and evaluating strategies to incentivise cooperation in P2P file sharing systems.

References

[1] D Anick, D. Mitra, and M. M. Sondhi. Stochastic Theory of a Data-Handling System. *Bell Sys. Thech. J*, 61(8):1871–1894, Oct 1982.

[2] G. Ciardo, D. M. Nicol, and K. S. Trivedi. Discrete-event Simulation of Fluid Stochastic Petri Nets. *IEEE Transactions on Software Engineering*, 2(25):207–217, 1999.

[3] F. Clevenot and P. Nain. A Simple Model for the Analysis of SQUIRREL. In *Proceedings of INFOCOM 2004*, Hong Kong, Mar 2004.

[4] E. De Souza e Silva and R. Gail. An algorithm to calculate transient distributions of cumulative rate and impulse based reward. *Commun. in Statist. – Stochastic Models*, 14(3):509–536, 1998.

[5] L. Donatiello and V. Grassi. On evaluating the cumulative performance distribution of fault-tolerant computer systems. *IEEE Transactions on Computers*, 1991.

[6] A. I. Elwalid and D. Mitra. Statistical Multiplexing with Loss Priorities in Rate-Based Congestion Control of High-Speed Networks. *IEEE Transaction on Communications*, 42(11):2989–3002, November 1994.

[7] Z. Ge, D. R. Figueiredo, S. Jaiswal, J. Kurose, and D. Towsley. Modeling Peer-Peer File Sharing System. In *Proceedings of INFOCOM 2003*, San Francisco, USA, Apr 2003.

[8] M. Gribaudo and M. Sereno. Simulation of Fluid Stochastic Petri Nets. In *8th Int. Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 2000)*, San Francisco CA, USA, 2000.

[9] M. Gribaudo, M. Sereno, A. Bobbio, and A. Horvath. Fluid Stochastic Petri Nets augmented with Flush-out arcs: Modelling and Analysis. *Discrete Event Dynamic Systems*, 11(1 & 2), 2001.

[10] P. K. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan. Measurement, Modeling, and Analysis of a Peer-to-Peer File-Sharing Workload. In *In Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP-19)*, Bolton Landing, NY, USA, Oct 2003.

[11] Q. He, M. Ammar, G. Riley, H. Raj, and R. Fujimoto. Mapping Peer Behavior to Packet-level Details: A Framework for Packet-level Simulation of Peer-to-Peer Systems. In *11th Int. Sym. on Modeling, Analysis and Simulation of Computer Tel. System (MASCOTS 2003)*, Orlando, Florida, USA, Oct 2003.

[12] G. Horton, V. G. Kulkarni, D. M. Nicol, and K. S. Trivedi. Fluid stochastic Petri Nets: Theory, Application, and Solution Techniques. *European Journal of Operations Research*, 105(1):184–201, Feb 1998.

[13] K. Kant. An Analytic Model for Peer to Peer File Sharing Networks. In *Proceedings International Communications Conference*, Anchorage, AL, USA, May 2003.

[14] A. Kapur, N. G. Harold, I. Marcus, R. Brooks, and S. Rai. Performance and Design of P2P Networks for Efficient File Sharing. In *Industrial Engineering Research Conference*, Portland, Oregon, USA, 2003.

[15] S. Merugu, S. Srinivasan, and E. Zegura. p-sim: A Simulator for Peer-to-Peer Networks. In *In Proceeding of 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications System (MASCOTS 2003)*, Orlando, Florida, USA, Oct 2003.

[16] V. Misra, W. B. Gong, and Towsley. Fluid-based Analysis of a Network of AQM Routers Supporting TCP Flows with an application to RED. In *Proc. SIGCOMM '2000*, Stockholm, Sweden, 2000.

[17] H. Nabli and B. Sericola. Performability analysis: a new algorithm. *IEEE Transactions on Computers*, 45:491–494, 1996.

[18] S. Rácz and M. Telek. Numerical analysis of large Markovian reward models. Performance Evaluation, 36&37:95–114, 1999.

[19] M. Ripeanu. Peer-to-Peer Architecture Case Study: Gnutella Network. In *In proceedings of IEEE 1st International Conference on Peer-to-peer Computing (P2P2001)*, Linkoping Sweden, Aug 2001.

[20] S. Saroiu, P. K. Gummadi, R. J. Dunn, S. D. Gribble, and H. M. Levy. An Analysis of Internet Content Delivery Systems. In *In Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI 2002)*, Boston, MA, USA, Dec 2002.

[21] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A Mesaurement Study of Peer-to-Peer File Sharing System. In *In Proceedings Of Multimedia Computing and Networking (MMCN) 2002*, Jan 2002.

[22] S. Sen and J. Wang. Analyzing peer-to-peer traffic across large networks. In *In Proceedings of the Second SIGCOMM Internet Measurement Workshop (IMW 2002)*, Marseille, France, Nov 2002.

[23] B. Sericola. Transient Analysis of Stochastic Fluid Models. *Performance Evaluation*, 32(4):245–263, 1998.

[24] K. Trivedi and V. Kulkarni. FSPNs: Fluid Stochastic Petri nets. In *Application and Theory of Petri Nets 1993, Proc.* 14^{th} *Intern. Conference*, LNCS, Chicago, USA, June 1993. Springer Verlag.

[25] X. Yang and G. de Veciana. Service Capacity of Peer to Peer Networks. In *Proceedings of INFOCOM 2004*, Hong Kong, Mar 2004.

Calibration of a Queueing Model of RAID Systems

Peter G. Harrison [a,1] Soraya Zertal [b,2]

[a] *Imperial College London, South Kensington Campus, London SW7 2AZ, UK*

[b] *PRiSM, Université de Versailles, 45, Av. des Etats-Unis, 78000 Versailles, France*

Abstract

A recent queueing-based modelling methodology of RAID systems compared the mean disk access times of the two most common variants, RAID0-1 and RAID5, as well as a multi-RAID system in which they coexist. Accesses to multiple disks occur concurrently for each logical (user) request and complete only when every disk involved has completed. The models therefore needed to estimate the mean value of the maximum of the individual disk response times, each of which is modelled by the waiting time of an $M/G/1$ queue. This mean-max value was approximated in terms of the second moment of queueing time which in turn required the third moment of disk service time, itself a function of seek time, rotational latency and block transfer time. To achieve consistently good agreement with an event-driven simulator of the physical hardware and system software requires careful calibration of the resulting model's parameters and validation of its assumptions. This calibration and validation process involves detailed analysis of sub-models to reveal the restrictions necessary on the domain of real-world operating parameters that facilitate a viable predictive model. The process yields significant insight into several of the abstract subsystems involved that may be utilised in a range of practical modelling studies; for example, the effect of approximating a bank of parallel queues with synchronised arrivals by a bank of identical, independent queues. The final comparison against the hardware simulator shows excellent agreement, far surpassing that of the original model.

Keywords: queueing model, RAID system, mean response time

[1] Email: pgh@doc.ic.ac.uk

[2] Email: zertal@prism.uvsq.fr

This is the workshop version of the paper. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

1 Introduction

A recent model of RAID (Redundant Array of Independent Disks) systems [7, 11] proposed by the authors of this paper was based on the simple idea of a collection of $M/G/1$ queues, one for each disk in the array. The main modelling issues were therefore how to choose the service time distribution at each disk (corresponding to the 'G') and how to represent the synchronisation between the disks, caused by the mirroring and/or striping schemes used. The former has already been considered widely. We used disk hardware specifications, such as disk rotation speed and head lateral movement speed and acceleration, together with given distributions of disk I-O request sizes, assumed to be estimated by profiling. In fact, for testing purposes, we assumed requests to be composed of fixed numbers of blocks. Regarding the second issue, synchronisation amongst disk accesses and hence correlation of parallel response times, certain simplifying assumptions were made. Accesses to multiple disks occur concurrently and complete only when every disk involved has completed – a 'fork-join' architecture. We assumed first, that access time is the maximum of the response times in the set of *independent* $M/G/1$ queues representing each disk accessed. Secondly, we assumed that the mean value of this maximum is accurately approximated by a new 'mean-max' formula derived in [7]; this is indeed exact when all service times are exponential random variables. The accuracy of the mean-max formula was investigated in [7] and found to be good except when the variance of service times is low – not the usual scenario. A secondary issue is the assumption of Poisson arrivals. This assumption has been found to be quite robust in many modelling studies, often because the aggregate arrival process is well approximated by a superposition of independent, sparse renewal processes, a large number of which can be shown to approach a Poisson process asymptotically.

This paper addresses the parameterisation, assumptions and approximations used in this RAID model and identifies distributional restrictions on the random variables used to parameterise it necessary to attain reliable predictions. In the next section, the RAID system and queueing model are briefly overviewed, the reader being referred to the previous publications for the details. In section 3, the numerical results of the model are compared with simulation and sensitive areas, where agreement deteriorates, are identified. The possible sources of error, listed above, are then each investigated independently in section 4.

2 Overview of the RAID system and model

A RAID storage system consists of a disk system manager and a collection (array) of independent disks. The disk system manager is a software component; it receives requests from the, typically many, users of the system. These requests are considered *logical* because they are completely independent of the physical configuration of the storage system. Requests arrive from the different users at various rates λ'_i, $1 \leq i \leq Cl$. The disk system manager subdivides the data into blocks called stripe units and distributes them across the collection of disks. Consequently, for each logical request, it generates a number of physical requests and sends them to the associated disks. Each disk i of the array receives requests at rate λ_i as shown in figure 1, $1 \leq i \leq N$. Finally, the disk system manager waits for the (physical) responses from each requested disk to construct the (logical) response to each logical request, which it then sends to the corresponding user.

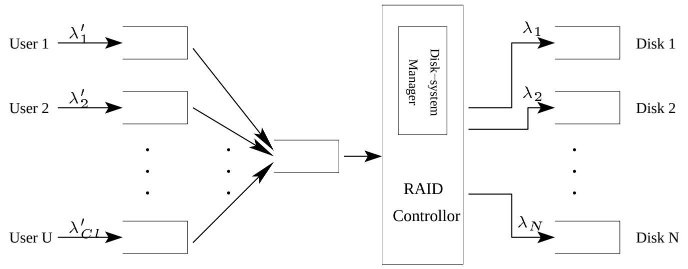


Fig. 1. Requests flow in a RAID storage system

The request subdivision and distribution process is performed according to the data-placement/redundancy pattern over the disks. In fact, there are various RAID levels [3] corresponding to these patterns [4, 5], but we are interested in the two most common and useful ones: RAID0-1 and RAID5.

2.1 RAID levels

In the RAID0-1 level, both shadowing (full redundancy) and striping are used. The disk collection is divided into two groups: native disks and mirror disks, which are both subdivided into stripe units. All data is duplicated and distributed on both the native disks and the mirror disks. A read physical request is sent to the native or to the mirror disk while a write physical request is sent to both of them in order to maintain the native and mirror data coherency. In the RAID5 level, block striping and parity based redundancy are used to improve performance in the sense of the rate of processing of logical requests at low cost. The redundancy units are spread across the disks in a cyclic manner. Thus, the redundancy disk is for every stripe [4], which enhances the writes' parallelism.

2.2 The RAID queueing model

As already noted in the introduction, the entire RAID model is based on the $M/G/1$ queue with various extensions to account for the fork-join nature of the

[3] A RAID level is characterised by a specific data/redundancy placement scheme.
[4] A stripe is a collection of native data blocks (stripe units) stored on a subset of the disks and the redundancy block stored on another disk.

parallel disk accesses corresponding to a logical request. The response time of each physical request, to an individual disk, is composed of four components: the time spent waiting to start service in the disk queue (Q), the seek time (S), the rotational latency (R) and the transfer time which we separate into two components, T and t, corresponding to transfer from the disk's buffer (via a bus) and the physical rotation of the disk respectively. The service time of the server in the disk's $M/G/1$ queue is the sum of the last three of these components and estimated from the hardware parameters, the particular model chosen for seek time and the physical block-size distribution, this depending on the workload profile and the RAID variant. The queueing component is calculated as an output of the $M/G/1$ model. The arrival rate to the server is computed from the logical request arrival rate, a pure workload parameter, and the RAID variant, assuming uniform access to the disks.

The access time of a logical request is then defined as the maximum of all its physical request access times; we require the expected value of this quantity. It is estimated by the mean-max formula of [7], under the approximating assumption that the physical request access times are independent, as follows.

The expected value of the maximum of n *independent, non-negative random variables with means* $\mathbf{m} = (m_1, \ldots, m_n)$*,* $\boldsymbol{\alpha} = (m_1^{-1}, \ldots, m_n^{-1})$ *and second moments* $\mathbf{M} = (M_1, \ldots, M_n)$ *is approximated by the function* $I(n, \boldsymbol{\alpha}, \mathbf{M})$ *defined by the recurrence*

$$I(k, \boldsymbol{\alpha}, \mathbf{M}) = \frac{1}{k} \sum_{i=1}^{k} I(k-1, \boldsymbol{\alpha}_{\setminus i}, \mathbf{M}_{\setminus i}) + \alpha_i M_i L_{k-1}(\boldsymbol{\alpha}_{\setminus i}, \alpha_i)/2 \tag{7}$$

$$I(1, \alpha_1, M_1) = 1/\alpha_1$$

for $k = 2, \ldots, n$*, where* $\boldsymbol{\alpha}_{\setminus i} = (\alpha_1, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_n)$*,* $\mathbf{M}_{\setminus i}$ *similarly, and* $L_{k-1}(\boldsymbol{\alpha}_{\setminus i}, s)$ *is the Laplace transform of the probability density function of the maximum of* $k-1$ exponential *random variables with parameters* $\boldsymbol{\alpha}_{\setminus i}$*.*

It is shown that this result is exact if all the random variables are exponential. In the special case that all the parameters are equal, say $\alpha_i = \alpha$ and $M_i = M$ for $1 \leq i \leq n$, we have $L_{k-1}(\boldsymbol{\alpha}_{\setminus i}, \alpha_i) = 1/k$ and so

$$I(k, \boldsymbol{\alpha}, \mathbf{M}) = I(k-1, \boldsymbol{\alpha}, \mathbf{M}) + \frac{M\alpha}{2k}$$

and hence

$$I(k, \boldsymbol{\alpha}, \mathbf{M}) = 1/\alpha + (M\alpha/2) \sum_{i=2}^{k} 1/i$$

For each type of access (read or write), RAID variant and several ranges of request sizes, the number of participating disks is computed, along with the parameters of their $M/G/1$ models, in the numerical examples considered in the next section. The intricate details appear in [7, 11]. The number of participating disks determines the number, n above, of random variables maximised over and then averaged.

3 Results

In order to validate our model and assess its accuracy, we developed a detailed event-driven simulator. This simulator is written in C and is composed of three main parts. The first part is a logical request generator, which uses standard random number generation functions to produce inter-request arrival times with arbitrary probability distributions. The second part is a logical to physical mapping, which contains all the physical request generation functions. This part deals with the different access modes and rates of the physical requests corresponding to the redundancy (RAID level) associated with their requested storage area. The third part is the simulation engine, which schedules the execution of physical requests on (operational abstractions of) the disks and manages synchronisation. We obtained the hardware parameters from a library, which we separated from the execution routines in order to enhance the flexibility and the scalability of the simulator.

We generated workloads with different mean logical request sizes (measured in blocks of 4KB each), using sizes of 1, 4 and 8 blocks to represent minimum and small- to-medium requests. It would also be interesting to use bigger sizes (going up to 250 blocks) to represent medium-to-large requests. In fact, the upper bound is 1MB for the large requests observed in image applications. We discarded such big requests because applications manipulating them don't use the RAID levels considered here but RAID3 instead. Concerning the balance between reads and writes in the workload, we generated model inputs with three ratios : 25% of reads for write oriented workloads, 75% of reads for read oriented workloads and 100% of reads for exclusively read workloads. For the results presented in this paper, we used an array of 16 disks. The characteristics of the disks we used are : number of cylinders C=1200; full rotation time $R_{MAX} = 16.7$ ms; number of blocks per track (bpt) = 12; acceleration time $a = 3$ms; seek factor $b = 0.5$ and one block transfer time $T = 1.34$ms. We chose this parameterisation in order to compare our results with those in [3]. Any modifications needed for testing more modern disks are straightforward, and more advanced architectures, e.g. with variable sector sizes according to cylinder, can be handled with an adapted model. Notice that we simulate the physical operation of a real RAID system, not the queueing model abstraction considered in the previous section. All service times are taken from the operational characteristics of the system, which are modelled explicitly in the simulation and aggregated in the analytical model. To validate our analytical model, we first assumed external Poisson arrivals of the logical requests and then validated this assumption by considering non-exponential inter-arrival times in section 4.

Simulations were run for a warm-up period of 300000 logical requests to allow the system to reach a stable state. They were then run for a further 700000 logical requests during which the measurements concerning response time were gathered. The confidence bands are quite narrow but omitted here. However, the regions where there is good agreement and bad between the simulation and the analytical model are apparent. Figures 2 to 7 compare the mean response time predicted by the analytical model with that obtained by simulation for minimum, small-to-medium request sizes under RAID0-1 and RAID5 redundancy levels and for decreasing ratios (1, 0.75 and 0.25) of read to write disk accesses.

For read-only accesses, the model and the simulation response times show excellent agreement in figures 2 to 7 over the whole range of request sizes considered and for both RAID0-1 and RAID5.

RAID0-1 is superior at the small request sizes 1 and 4 on figures 2 and 4 compared to figures 3 and 5, where the extra complexity of the RAID5 redundancy and striping-based scheme leads to a penalty rather than a benefit. Comparing with figures 6 and 7, we can see how the system behaves in small vs. medium request size environments. We deduce that the workload thresholds (above which performance degrades very rapidly to unacceptable levels) decrease considerably with the increase in the mean request size; by a factor of about 10 here for RAID5 and 2 for RAID0-1. Thus, RAID5 is penalised more than RAID0-1 on small requests, suggesting a 'cross-over point' at a higher request size, below which RAID0-1 is the better scheme and above which RAID5 gains increasing superiority.

As the proportion of write accesses increases in figures 2 to 7, the agreement remains good but deteriorates at high loads (i.e. high logical request arrival rates), especially for RAID5. This is the region in which the approximating assumptions are put to the test more stringently, the queueing component of the response time dominating. We examine their relative effects quantitatively in the next section but note here that we can still predict the onset of excessive loading (the 'threshold' referred to above) accurately.

The last two pairs of figures, figures 8 and 9, show the effect of the choice of the ratio between RAID partition sizes on the whole storage system's performance, again in terms of response time. The complementary partition choices (75% of RAID0-1 and 25% of RAID5 in figure 8 against 25% of RAID0-1 and 75% of RAID5 in figure 9) show how response time is dominated by the larger partition. Note that the given fraction of the storage space (the partition size) implies an equal fraction of incoming requests to this partition. That is, the partition workload is proportional to its size.

In figure 9, 75% of the workload is allocated to RAID5, where, because of the small request size (one block), regrouping policies like full/large writes are inefficient. As a result, these costly writes (each one generates 4 disk accesses) lead to a high response time compared to that obtained in the mixed system of figure 8, where the writes are penalized less because only 25% are allocated to RAID5.

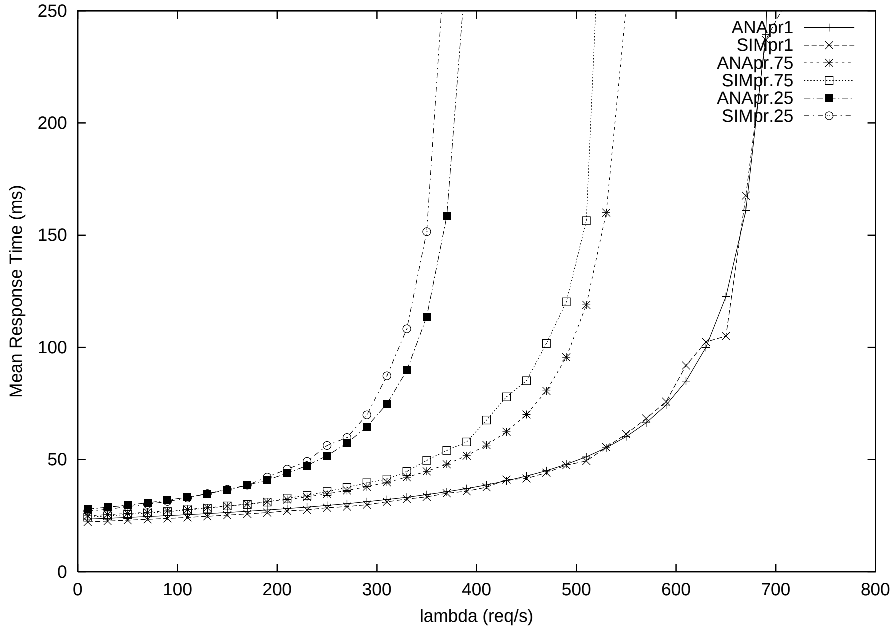


Fig. 2. RAID1 B=1

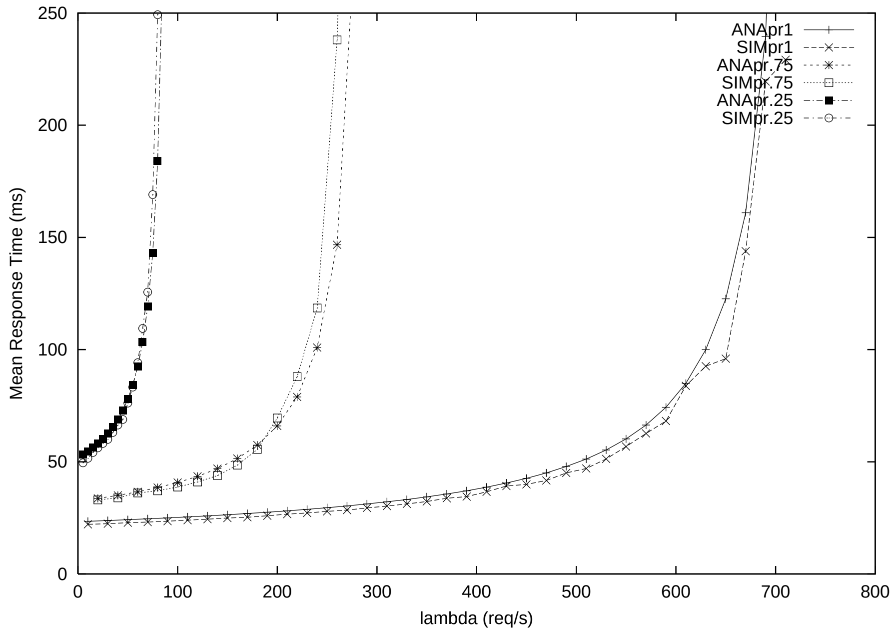


Fig. 3. RAID5 B=1

4 Sources of approximation

The most obvious candidate source of inaccuracy in our model, which we consider first, is the mean-max approximation of section 2.2, which is only exact

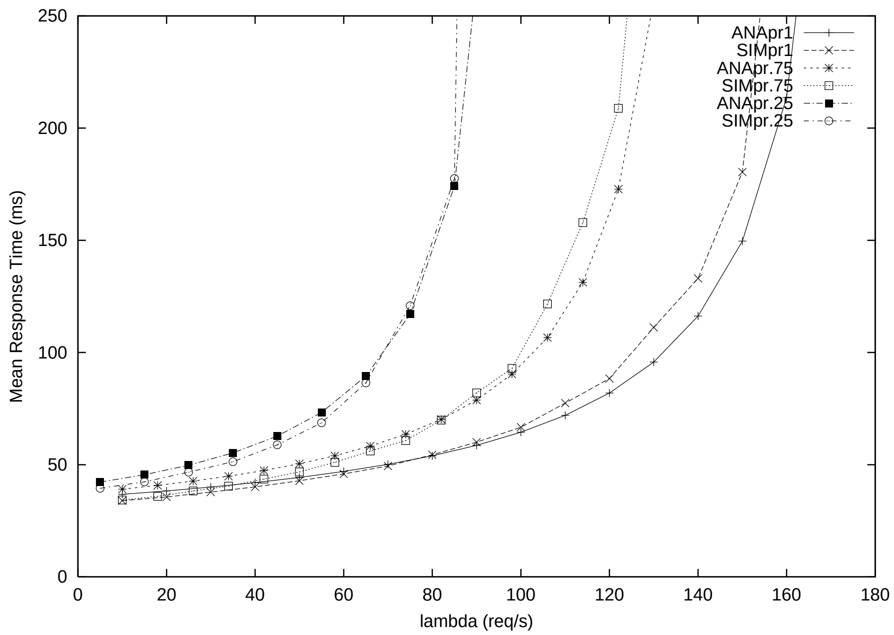


Fig. 4. RAID1 B=4

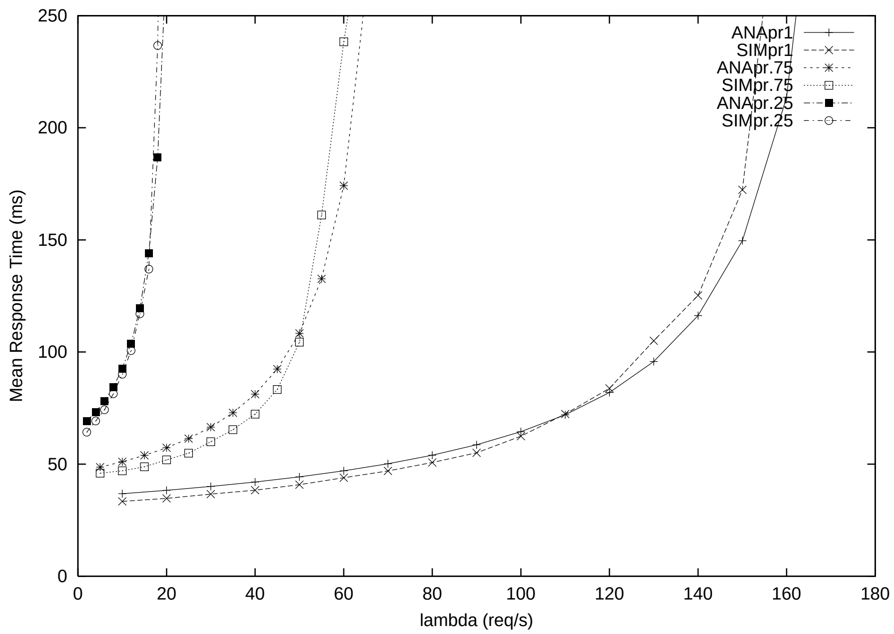


Fig. 5. RAID5 B=4

for parallel exponential delays. However, there are other potential causes of inaccuracies, which we also address in this section: inaccurate approximation for the moments of response time at a *single disk*, which we considered to be an $M/G/1$ queue with service times given by particular formulae for seek time and rotational latency, and dependence between these response times. We also investigate the robustness of the assumption of Poisson arrivals by comparing our results with simulations having non-Poisson arrivals.

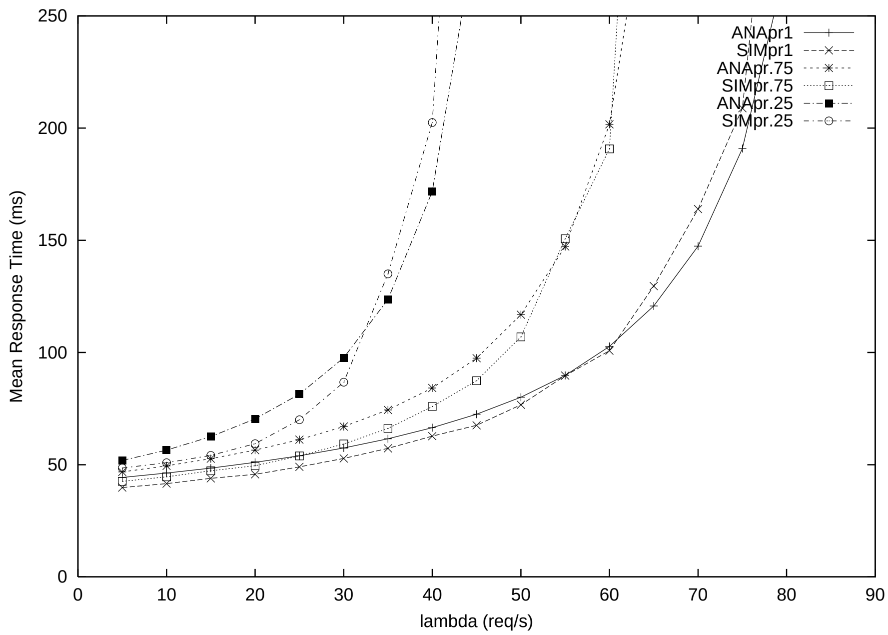


Fig. 6. RAID1 B=8

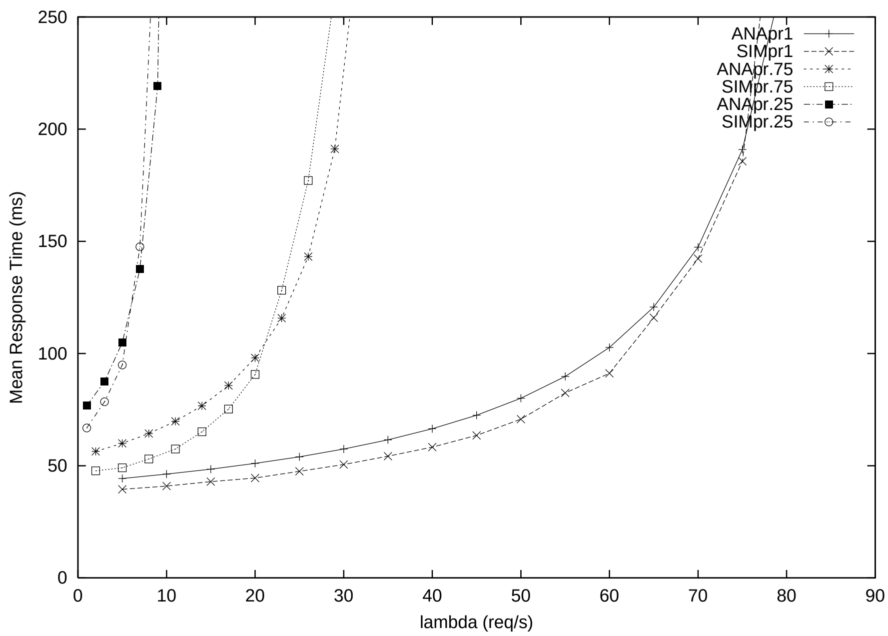


Fig. 7. RAID5 B=8

4.1 Accuracy of the mean-max approximation

To assess its accuracy, we compared the mean-max formula of section 2.2 against simulations of the maxima of a number N independent identically distributed random variables of two types: Erlang and Pareto. The simulations were run 100,000 times, giving 98% confidence bands of the order 0.01.

Each test distribution was standardised to have unit mean value so that

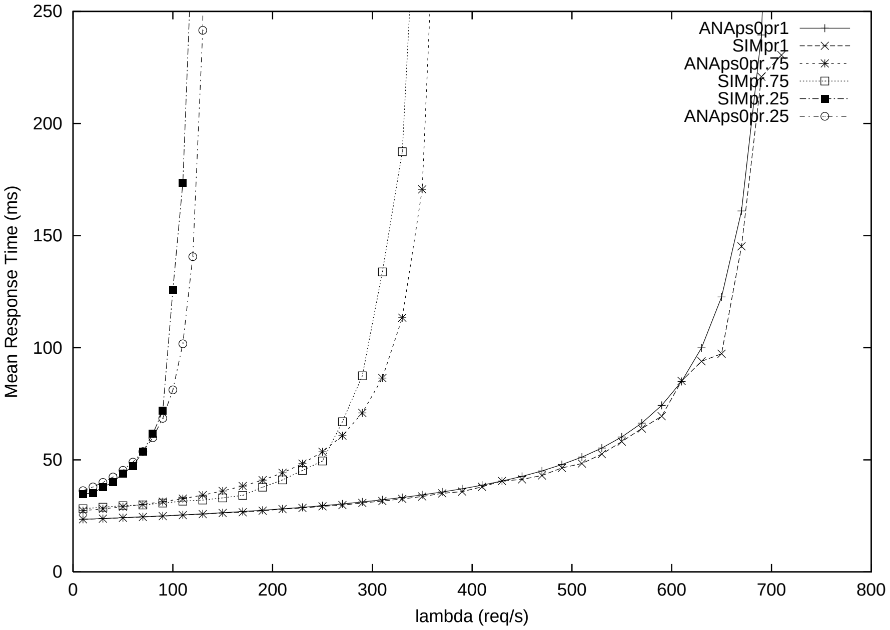


Fig. 8. RAID0-1 = 75% RAID5 = 25%

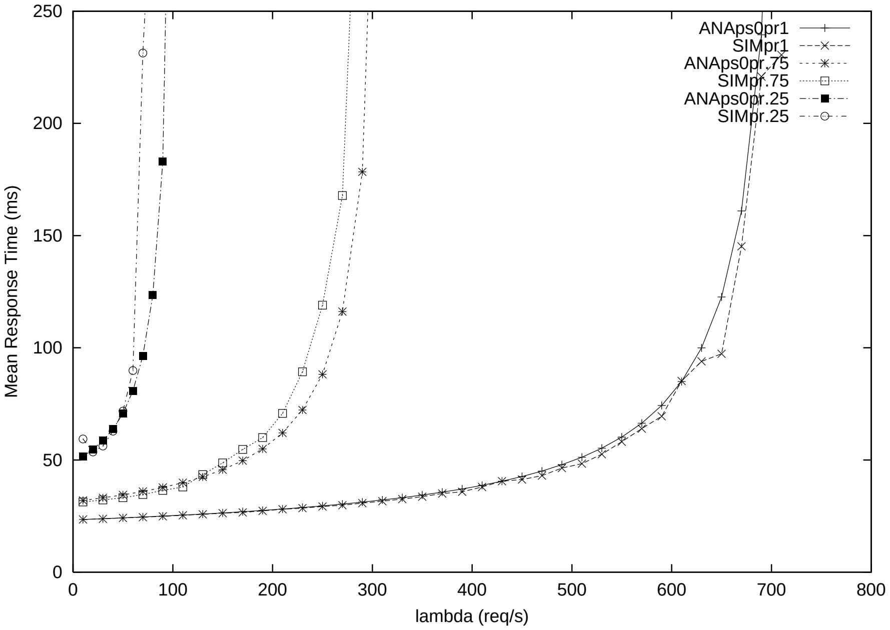


Fig. 9. RAID0-1 = 25% RAID5 = 75%

the approximate mean-maximum is determined solely by the second moment. Notice that, even when the variance is zero, the second moment is the square of the mean, *viz.* 1. Consequently, the approximation's estimate will always diverge as the number of parallel random variables maximized increases. Thus, for N deterministic random variables, here each equal to 1 with probability 1, the exact mean-maximum is 1 whereas the approximation diverges to infinity with N. Thus, the approximation is not appropriate for small variances.

This is illustrated in table 1 where the approximation is tested for Erlang-2, Erlang-3 and Erlang-4 distributions. The mean of a k-phase, Erlang-k distribution with parameter λ is k/λ and so we choose $\lambda = k$. The variance is therefore $k/\lambda^2 = 1/k$ which tends to zero as $k \to \infty$. Thus the approximation deteriorates at larger k, as we see from the table 1. The second moment of the k-phase Erlang is $1 + 1/k$ and we see a 36% error for 16 parallel Erlang-4 random variables. Each of these has variance 0.25 and so we see poor agreement at moderately small variances for more than 8 parallel random variables – all overestimates as expected. However, for up to 4 in parallel, the accuracy is quite acceptable; this happens in reads from mirrored disks and RAID accesses with small numbers of blocks. Also included in each row of the table is the mean of the maximum of N parallel exponential random variables, each with unit parameter. As can be seen in section 2.2, this is just the Nth harmonic number and it can be seen that it overestimates seriously; more than double the error in its best case of 16 parallel Erlang-4 distributions.

N	Exp-1	Erlang-2			Erlang-3			Erlang-4		
		Mod	Sim	% err	Mod	Sim	% err	Mod	Sim	% err
1	1.000	1.000	1.003	-0.334	1.000	0.999	0.062	1.000	0.999	0.060
2	1.500	1.375	1.373	0.135	1.313	1.271	3.281	1.281	1.195	7.207
4	2.083	1.813	1.772	2.265	1.677	1.546	8.448	1.609	1.380	16.64
8	2.718	2.288	2.182	4.881	2.074	1.806	14.84	1.966	1.555	26.43
16	3.381	2.786	2.588	7.648	2.488	2.061	20.74	2.339	1.716	36.30

Table 1
Comparison with Erlang (low-variance)

However, in practice, waiting times in queues tend not to have very low variance – it would be perhaps easier to predict if they did. Consequently, we tested the accuracy of the approximation near the opposite extreme, against high variance, heavy-tailed Pareto distributions. Again these were chosen to have unit mean and zero distribution function at the origin. The form of the distributions chosen is $F_P(x) = 1 - \alpha(x+\gamma)^{-\beta}$, where $\beta > 2$ for the first two moments to be finite. In order to pass though the origin and have unit mean, we require $\alpha = \gamma^\beta$ and $\gamma = \beta - 1$. This gives a second moment $M_2 = 2 + 2/(\beta - 2)$, which we use to parameterise the approximation. We call a Pareto distribution with these properties Pareto-β and compare our approximation with simulation for the mean-maximum of Pareto-4 and Pareto-5 random variables; see table 2.

It can be seen that the agreement is much better here than for the low variance cases. In fact the approximation is at its worst for moderately small numbers in parallel (N), improving as N reaches 16. As expected, the approximation improves as the parameter β increases, giving a lower variance closer to that of the exponential, 1. The exponential mean-maximum values are

N	Exp-1	Pareto-4			Pareto-5		
		Mod	Sim	% err	Mod	Sim	% err
1	1.000	1.000	1.004	-0.381	1.000	0.994	0.614
2	1.500	1.750	1.579	10.82	1.667	1.567	6.350
4	2.083	2.625	2.327	12.81	2.444	2.269	7.744
8	2.718	3.577	3.261	9.698	3.290	3.129	5.173
16	3.381	4.571	4.394	4.027	4.174	4.153	0.512

Table 2
Comparison with Pareto (high-variance)

repeated in this table and show underestimates, again as expected since the Pareto second moments are greater than that of an exponential random variable with mean 1, *viz.* 2. Recall too that, when these waiting times are exponential, the recurrence is exact. Indeed, if the waiting times are phase-type, the mean of their maximum can also be computed exactly, the maximum also being phase-type. This calculation has exponential complexity in N but an efficient polynomial approximation was obtained in [1]. This could be used in cases where ours is too inaccurate, for example low variance Erlang distributions, a special case of phase-type.

We conclude that only for coefficients of variation (ratio of standard deviation to mean) much less than one is the approximation likely to be poor. Fortunately this is the least likely scenario, file access times being notoriously variable, sometimes even having heavy tailed distributions.

4.2 Moment estimation at individual disks

We now assess the error introduced by approximating the delay experienced by a physical request at a single disk by the response time in an $M/G/1$ queue. This is a relatively simple response time to calculate since it excludes any additional delays waiting for the synchronisation with parallel requests to complete a 'join' operation. However, it is crucial as a component of the set of delays maximised and is itself given by the service time of the $M/G/1$ queue. We therefore first consider this.

4.2.1 Moments of service time

Service time, X, is defined as the sum of seek time, S, rotational latency, R, and transfer time, $K(T+t)$, where K is the number of blocks transferred. We write $Y = S+R$ and $X = Y+K(T+t)$ to be consistent with previous notation. There is no problem with the precision of the transfer time component since T and t are known constants and K is a control parameter in our experiments.

We therefore compared the random variables S, R and their sum Y in the analytical and simulation models. These are obviously independent of the arrival rate of logical requests, λ, and of other workload characteristics such as request size. We use the first three moments of these quantities in our model and so compared their analytically computed values (see [7, 11]) with those estimated from simulation runs. This was done by simply dividing the sum of the ith powers of the simulated quantity by (one less than) the number of times the simulator generated it, to estimate the ith moment, $i = 1, 2, 3$. To isolate the higher order effects and make a consistent comparison, we used *central* ith moments, raised to the power i^{-1}, for $i > 1$; this gives the standard deviation for $i = 2$. We obtained the following results, shown in Table 3:

	Model	Simulation
Mean (1)	20.58	20.54
Std Dev (2)	6.15	6.11
Cube root of 3rd central	8.45	2.32

Table 3
Comparison of moments of service time component $Y = S + R$

	Seek time S		Latency R	
	Model	Simulation	Model	Simulation
Mean (1)	12.23	12.22	8.35	8.32
Std Dev (2)	3.82	3.85	4.82	4.74
Cube root of 3rd central	8.45	2.10	0.02	0.71

Table 4
Comparison of moments of seek and latency times

We see that the agreement is good for the mean and the standard deviation. The difference noticed for the cube root of the 3^{rd} central moment come from the third moment summerized in the following table :

Latency R			Seek time S			Service time Y		
Model	Simulation	%err	Model	Simulation	%err	Model	Simulation	%err
1142.67	1139.53	0.274	1765.26	2363.69	-33.9	10457.7	10970.9	-4.9

Table 5
Comparison of 3^{rd} moment for latency, seek and service times

The question now becomes, how well will response times match at higher loads when they may be composed of many service time random variables?

4.2.2 Moments of queueing time at a single disk

To be able to use our approximate mean-max formula of section 2.2, we require the rate α, the reciprocal of $\overline{Q} + \overline{Y}$, and the corresponding second moment $M_2 = \overline{\overline{Q}} + 2\overline{Q}\ \overline{Y} + \overline{\overline{Y}}$. This requires the first two moments of the queueing time Q, which is given by the first three moments of the service time. The formulae used for the moments of Q require assumptions about the operation of the physical disks (relating to seek times and rotational latency) and rely on standard properties of the $M/G/1$ queue. We therefore next plotted graphs of the mean queueing time $\overline{Q}$ and the standard deviation of queueing time ($\sqrt{\overline{\overline{Q}} - \overline{Q}^2}$) against the external arrival rate of logical requests λ at a single disk, for various request sizes and RAID levels. Notice that at very small λ, the effects of queueing are negligible and we are only assessing the accuracy of our assumptions regarding the operation of the disks. This queueing effect becomes important as λ and the request size B increase leading to a disk congestion as shown on figure 10.

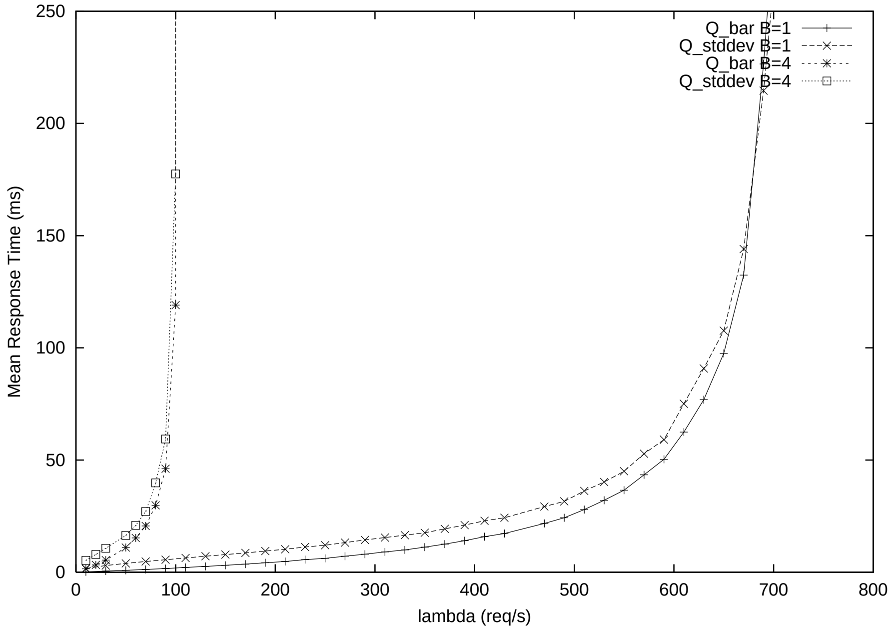


Fig. 10. Queueing time Moments (RAID0-1, pr=1)

As expected, we did observe that the imprecision in the calculation of the moments of service time magnifies in the queueing time moments, increasing the error in our mean response time calculation. In fact, the discrepancy in the queueing time moments increases more sharply at about the same loading as in the corresponding mean response time graphs. This is particularly so in the case of the second moment of queueing time.

4.3 Dependence of parallel queues

The next concern is the dependence between the queues, caused by the synchronised arrivals of the physical requests spawned by a logical request. We cannot assume that the collection of response times, of which we estimate the mean of the maximum, is independent. For example, if service times are constant and arrivals are synchronised, i.e. always occur simultaneously at each of a set of disks, every disk will behave identically and so all response times will be the same in the maximized set. Thus the maximum response time will be that of a single disk no matter how many disks we have in the RAID system. The same applies for any service time distribution if the service time of each of the synchronised parallel requests is the same. Our mean-max approximation, however, will diverge (logarithmically) as the number of disks increases, giving an infinite error! This is essentially the low-variance situation we considered in section 4.1, but here it implies that we cannot ignore dependencies between arrivals, even though we would be unlikely to encounter such extreme circumstances in practice because of asynchronous positioning of disk heads and the interleaving of logical requests requiring only subsets of the RAID array, even one disk.

An analytic assessment would appear to be out of the question, requiring the joint distribution of queue lengths at an arbitrary number of queues for a start, then a multidimensional analysis of response times using either supplementary variables or possibly finding some embedded Markov chain, as in a single $M/G/1$ queue. Consequently we again use simulation. The following experiments were conducted for various service time distributions G, including exponential, Erlang, deterministic (constant) and Pareto (with parameters taken from the set used in section 4.1, all with unit mean):

- For $N = 2, 4, 8, 16$, calculate the mean-max of the response times of the N *independent* $M/G/1$ queues (exactly as in our model), each with arrival rate λ (for various λ);
- For the same set of values of N, calculate the mean-max of the response times of the same N *fully synchronised* $M/G/1$ queues – in other words, there is a single Poisson arrival process with rate λ which generates an arrival to *every* queue at each arrival instant.

These two scenarios represent the extremes of observable behaviour. In practice, not all disks will be involved for every request, leading to asynchronous behaviour that one would expect to be better approximated by assuming independence.

From the tables 6 and 7 we observe that there is not a great difference between the scenarios except for the higher-order Erlang and deterministic (unit response time) cases, which have small variance (zero in the latter case). This is consistent with our observations in section 4.1.

N	Exp		Erlang-2		Erlang-4	
	Ind	Dep	Ind	Dep	Ind	Dep
2	1.62	1.62	1.47	1.46	1.35	1.34
4	2.26	2.23	1.90	1.87	1.65	1.62
8	2.95	2.89	2.34	1.29	1.95	1.89
16	3.66	3.58	2.80	2.71	2.25	2.15

Table 6
Comparison with Erlang

N	Pareto-4		Pareto-5		Constant	
	Ind	Dep	Ind	Dep	Ind	Dep
2	1.77	1.77	1.73	1.72	1.08	1.04
4	2.61	2.62	2.53	2.51	1.15	1.04
8	3.73	3.68	3.51	3.46	1.28	1.04
16	5.06	4.96	4.66	4.57	1.47	1.04

Table 7
Comparison with Pareto and Constant

4.4 Non-Poisson arrivals

For our final test, we relaxed the Poisson arrival requirement, by simply using alternate arrival processes in the simulation. These were parameterised so that we could plot analogous graphs to those of section 3, using the same set of average arrival rates λ. We used the following interarrival time distributions, each with mean interarrival time $1/\lambda$: Erlang-$n(n, \lambda)$ for $n = 2, 4$; generalised exponential $\mathrm{GE}(p, p\lambda) = 1 - pe^{-p\lambda t}$ for $p = 0.5$ and the 2-phase Interrupted Poisson Process $\mathrm{IPP}(Q, 2\lambda)$ with generator matrix $Q = \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix}$ modulating the two phases in which the arrival rates are 0 and 2λ – again giving average arrival rate λ. The GE distribution gives Poisson arrivals of *batches* with geometric size ($p = 1$ gives unit batches, as in a Poisson process, smaller p gives larger batches) and the IPP, gives correlated traffic.

We found mean response time to be fairly insensitive to the particular distribution of inter-arrival time, only to its mean value (i.e. only to the 'arrival rate'). We can see on figure 11 that the GE and IPP curves are slightly on top, which is predictable because of the bursty/batch characteristic of such distributions. However, the difference represents 2.29% and 7.33% (for GE and IPP respectively) of the response time with poisson arrival distribution at high arrival rates. In fact the Poisson arrival assumption has often been found to be robust, especially for modelling external, user-generated, logical requests because such external traffic is usually composed of a number of low intensity streams that behave independently. The superposition of such sparse streams can be shown to approximate a Poisson process under quite mild assumptions.

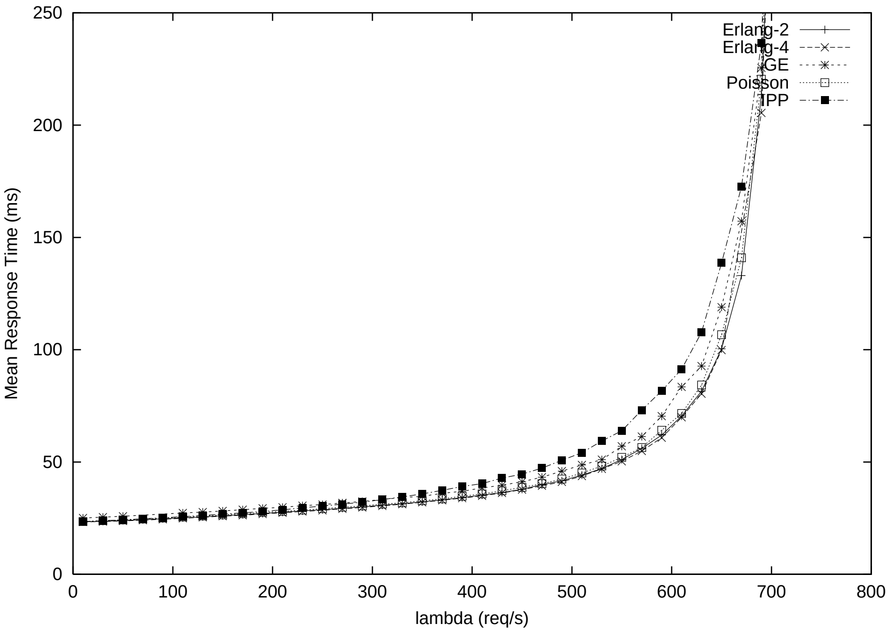


Fig. 11. Poisson Vs non-poisson arrival distributions

5 Conclusion

We have systematically constructed analytical models of two RAID storage systems, RAID0-1, RAID5 and a multi-RAID where both of RAID0-1 and RAID5 cœxist, based on detailed sub-models of their constituent (hardware and software) parts, and validated them against explicit simulations of their detailed operation. Such models are not new and our contribution is the methodological way in which we have developed the model by fine-tuning those parts that were not giving an adequate representation – whilst at the same time, keeping the model simple and efficient.

Analytical results were compared with simulation at a very fine level of abstraction and showed very good agreement at low-medium loads for a range of request sizes and read-write access ratios. In addition, the model predicted the onset of saturation well, i.e. the level of loading above which response time grows rapidly to unacceptable levels whereupon poor quality of service ensues. Apart from this overall comparison, specific assumptions of the analytical model were carefully checked and the most serious causes of inaccuracy were identified. We considered

- the accuracy of the mean-max approximation, central to the model;
- the precision in our estimates of seek time and rotational latency, based on standard sub-models, together with their compounded effect on queueing time;
- the effect of synchronisation between parallel fork-join queues.

We concluded that the main causes of inaccuracy were the first and third of these, the effect only being serious when disk service times were fairly consistent and so relatively predictable, i.e. having small variance. This is rarely the case with today's disk access patterns.

We also checked the robustness of our assumption that external requests arrive as a Poisson stream, finding that mean response time is sensitive primarily to just the arrival rate rather than to the particular distribution of inter-arrival time.

In the calculation of the mean of the maximum of an independent set of random variables, in general, the rate and second moment parameters α_i, M_i of section 2.2 are distinct. In this study we assumed equal parameters, giving a simple non-recursive result, but it requires a controlled experiment to ensure that all the workload parameters are the same at every disk. In fact, the disk-selection probability for a physical request is particularly sensitive to workload variations and choice of RAID level, influencing the arrival rate at each disk. An optimisation of the general calculation, in particular when the parameters fall into classes corresponding to subsets of identical disks, is the subject of work in progress. It is also important to further consider the extent of the error in the mean-maximum calculation. For low variances especially, a comparison against exact results in the phase-type case could be carried out, using the method of [1], cf. Indeed, in small models, the phase-type method itself might be used.

Finally, we are extending the study to a dynamic and heterogeneous storage system, dealing with the layout schemes and reconfiguration necessary for a RAID scheme that adapts to its varying offered workload. This work also includes the representation of much larger request sizes. We will then be able to evaluate the overheads of the related data migration and communications.

References

[1] Bohnenkamp, H. and B. Haverkort, *The mean value of the maximum*, in: *Proc. PAPM/PROBMIV 2002, Lecture Notes in Computer Science 2399, Springer-Verlag, pp. 37–56*, 2002.

[2] Chen, S., "Design, Modeling and evaluation of high performance, ph.D. thesis," ph.D. thesis, University of Massachusetts, (USA) (1992).

[3] Chen, S. and D. Towsley, *A performance evaluation of RAID architecture*, in:

IEEE Transactions on computers, 1997.

[4] D. A. Patterson, G. G. and R. H. Katz, *A case for redundant arrays of inexpensive disks (RAID)*, in: *Proceedings of SIGMOD Conference*, 1988.

[5] D. A. Patterson, G. G., P. M. Chen and R. H. Katz, *Introduction to redundant arrays of inexpensive disks (RAID)*, in: *IEEE COMPCON*, 1989.

[6] Gravey, A., *A simple construction of an upper bound for the mean of the maximum of n identically distributed random variables*, Journal on Applied Probability (1985), pp. 844–65.

[7] Harrison, P. and S. Zertal, *Queueing models with maxima of service times*, in: *Proceedings of TOOLS Conference*, 2003.

[8] Lee, E. and R. H. Katz, *An analytic perfomance model of disk arrays and its application*, in: *Technical Report UCB/CSD 92/660*, 1991.

[9] The RAID Advisory board, "The RAIDBOOK: A source Book for RAID Technology," Lino Lakes MN Publisher, 1993.

[10] Zertal, S., "Dynamic redundancy mechanisms for storage customisation on multi disks storage systems," ph.D. thesis, University of Versailles, France (2000).

[11] Zertal, S. and P. Harrison, *Multi-level RAID storage system modelling*, in: *Proceedings of 2003 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, 2003.

Performance models for *master/slave* parallel programs [1]

Lucas Baldo [2] Leonardo Brenner [3] Luiz Gustavo Fernandes [4]
Paulo Fernandes [5] Afonso Sales [6]

Faculdade de Informática
Pontifícia Universidade Católica do Rio Grande do Sul
Porto Alegre, Brazil

Abstract

This paper proposes the use of Stochastic Automata Networks (SAN) to develop models that can be efficiently applied to a large class of parallel implementations: *master/slave* (m/s) programs. We focus our technique in the description of the communication between master and slave nodes considering two standard behaviors: *synchronous* and *asynchronous* interactions. Although the SAN models may help the pre-analysis of implementations, the main contribution of this paper is to point out advantages and problems of the proposed modeling technique.

Keywords: performance evaluation, analytical models, stochastic automata networks, parallel programming, propagation algorithm.

1 Introduction

Performance evaluation is, or at least should be, a primary concern of parallel program developers. Unfortunately, most of this concern does not appear early enough in the development process. Execution trace techniques and visualization tools [23, 13, 22] offer important information about performance of an existing parallel implementation, but if you have a trace of execution much of the implementation effort was already done. Benchmarks [2, 10] contribute to generic, and yet useful, information about an execution platform.

[1] This work was developed in collaboration with HP Brazil R&D.

[2] Email: `lbaldo@inf.pucrs.br`
[3] Email: `lbrenner@inf.pucrs.br`
[4] Email: `gustavo@inf.pucrs.br`
[5] Email: `paulof@inf.pucrs.br` (corresponding author - the order of authors is merely alphabetical)
[6] Email: `asales@inf.pucrs.br`

This is the workshop version of the paper. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: `www.elsevier.nl/locate/entcs`

However, in the absence of a formal approach to model the interaction of a designed parallel program, it can be very hard to correctly estimate the behavior of its implementation before starting the programming phase.

To cope with this problem, we propose the use of modeling formalisms to describe and analyse the designed implementation and its platform. We believe that such modeling may furnish good estimation on the implementation behavior in quite early stages of the parallel program development. The main difficulty in doing so is the (usually great) complexity in developing performance models. In fact, the development of an accurate model demands as much understanding of parallel programs as performance issues. Thus, it would be very interesting to relieve the parallel programmers from this performance analysis skills burden.

Although far from an automated generator of performance models [21], we propose some modeling techniques that can be efficiently applied to a large class of parallel implementations: the *master/slave* (m/s) programs, *e.g.*, *bag of tasks* solutions [1]. We focus our technique in the description of the communication between master and slave nodes considering two standard behaviors: *synchronous* and *asynchronous* interaction. Our experience in the matter suggests that the adequate modeling of the communication is the key issue to develop models for such parallel programs. This paper contribution is based on the real case study of a classical parallel application to, hopefully, describe in the same manner any m/s parallel program implementation.

The next section informally presents the *Stochastic Automata Networks* (SAN) formalism [17] with a minimum of details. Section 3 briefly describes the concept of m/s programs and the possible kinds of interaction between master and slave nodes, as well as it presents the generic m/s models using the SAN formalism. To illustrate our approach, Section 4 presents a practical case example: the implementation choices for a parallel version of the Propagation Algorithm. Section 5 shows the SAN models describing the possible implementations of the Propagation Algorithm. Section 6 presents the analysis of performance indices for the presented models. Finally, the conclusion summarizes our contribution and suggests future works.

2 Stochastic Automata Networks

The SAN formalism was proposed by Plateau [16] and its basic idea is to represent a whole system by a collection of subsystems with an independent behavior (*local transitions*) and occasional interdependencies (*functional rates* and *synchronizing events*). The framework proposed by Plateau defines a modular way to describe continuous and discrete-time Markovian models [17]. However, only continuous-time SAN will be considered in this paper, although discrete-time SAN can also be employed without any loss of generality.

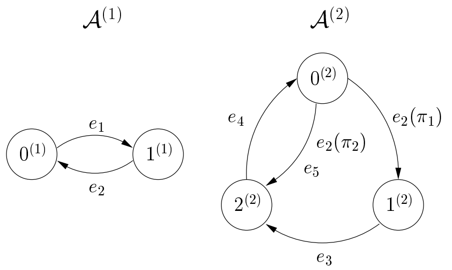
The SAN formalism describes a complete system as a collection of subsystems that interact with each other. Each subsystem is described as a stochastic automaton, *i.e.*, an automaton in which the transitions are labeled with probabilistic and timing information. Hence, one can build a continuous-time stochastic process related to SAN, *i.e.*, the SAN formalism has exactly the same application scope as Markov Chain (MC) formalism [20, 6]. The state of a SAN model, called *global state*, it is defined by the cartesian product of the *local states* of all automata.

There are two types of events that change the global state of a model: *local events* and *synchronizing events*. Local events change the SAN global state passing from a global state to another that differs only by one local state. On the other hand, synchronizing events can change simultaneously more than one local state, *i.e.*, two or more automata can change their local states simultaneously. In other words, the occurrence of a synchronizing event forces all concerned automata to fire a transition corresponding to this event. Actually, local events can be viewed as a particular case of synchronizing events that concerns only one automaton.

Each event is represented by an *identifier* and a *rate* of occurrence, which describes how often a given event will occur. Each transition may be fired as result of the occurrence of any number of events. In general, non-determinism among possible different events is dealt according to Markovian behavior, *i.e.*, any of the events may occur and their occurrence rates define how often each one of them will occur. However, from a given local state, if the occurrence of a given event can lead to more than one state, then an additional *routing probability* must be informed. The absence of routing probability is tolerated if only one transition can be fired by an event from a given local state.

The other possibility of interaction among automata is the use of functional rates. Any event occurrence rate may be expressed by a constant value (a positive real number) or a function of the state of other automata. In opposition to synchronizing events, functional rates are one-way interaction among automata, since it affects only the automaton where it appears.

Fig. 1 presents a SAN model with two automata, four local events, one synchronizing event, and one functional rate. In the context of this paper, we will use roman letters to identify the name of events and functions, and greek letters to describe constant values of rates and probabilities.



Type	Event	Rate
loc	e_1	f
syn	e_2	μ
loc	e_3	σ
loc	e_4	δ
loc	e_5	τ

$$f = \left[\left(st\ \mathcal{A}^{(2)} == 0^{(2)}\right) * \lambda\right] + \left[\left(st\ \mathcal{A}^{(2)} == 2^{(2)}\right) * \gamma\right]$$

Fig. 1. Example of a SAN model

In the model of Fig. 1, the rate of the event e_1 is not a constant rate, but a functional rate f described by the SAN notation[7] employed by the PEPS tools [4]. The functional rate f is defined as:

$$f = \begin{cases} \lambda \text{ if automaton } \mathcal{A}^{(2)} \text{ is in the state } 0^{(2)} \\ 0 \text{ if automaton } \mathcal{A}^{(2)} \text{ is in the state } 1^{(2)} \\ \gamma \text{ if automaton } \mathcal{A}^{(2)} \text{ is in the state } 2^{(2)} \end{cases}$$

The firing of the transition from $0^{(1)}$ to $1^{(1)}$ state occurs with rate λ if automaton $\mathcal{A}^{(2)}$ is in state $0^{(2)}$, or γ if automaton $\mathcal{A}^{(2)}$ is in state $2^{(2)}$. If automaton $\mathcal{A}^{(2)}$ is in state $1^{(2)}$, the transition from $0^{(1)}$ to $1^{(1)}$ state does not occur (rate equal to 0).

The use of functional expressions is not limited to event rates. In fact, routing probabilities also may be expressed as functions. The use of functions is a powerful primitive of the SAN formalism, since it allows to describe very complex behaviors in a very compact format. The computational costs to handle functional rates has decreased significantly with the developments of numerical solutions for the SAN models, *e.g.*, the algorithms for generalized tensor products [4]. Fig. 2 presents the equivalent MC model to the SAN model in Fig. 1.

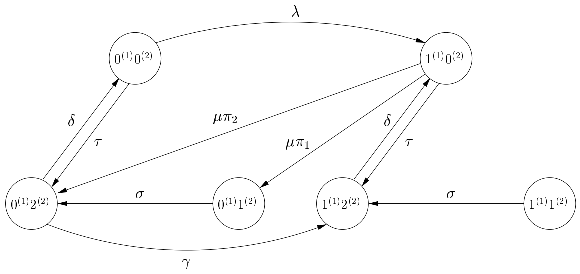


Fig. 2. Equivalent Markov Chain to the SAN model in Fig. 1

It is important to notice that only 5 of the 6 states in this MC model are reachable. In order to express the reachable global states of a SAN model, it is necessary to define a (*reachability*) function. For the model in Fig. 1, the reachability function must exclude the global state $1^{(1)}1^{(2)}$, thus:

$$Reachability = !\left[\left(st\ \mathcal{A}^{(1)} == 1^{(1)}\right) \&\& \left(st\ \mathcal{A}^{(2)} == 1^{(2)}\right)\right]$$

3 Master/Slave paradigm for parallel programs

In the m/s paradigm, one master node generates and allocates work, usually called *tasks*, to N slave nodes. Each slave node receives tasks, computes them,

[7] The interpretation of a function can be viewed as the evaluation of an expression of non-typed programming languages, *e.g.*, C language. Each comparison is evaluated to value 1 (*true*) and value 0 (*false*).

and sends the result back. Thus, master node is responsible for receiving and analyzing the computed results by slave nodes (*summation*).

This paradigm is generally suitable for shared-memory or message-passing platforms, since the interaction is naturally two-way. The main problem with m/s paradigm is that the master may become a *bottleneck*. This may happen if the tasks are too small or if there are too many slaves connected to one master. The choice of the amount of work in each task is called *granularity*. One of the main performance issues in parallel programming is the choice between many small tasks (*fine grain*) or few large tasks (*coarse grain*).

According to how the master distributes tasks and collects results from the slaves, we present two different kinds of m/s implementations: programs with *synchronous* or *asynchronous* interaction[8]. A program is said to have synchronous interaction when the tasks need to be performed in phases, *i.e.*, all tasks in each phase must finish before the distribution of next phase tasks. In opposition, the interaction between the master and the slaves is called *asynchronous* when the master distributes new tasks every time a slave finishes its computation. Such interaction may reduce waiting time considering a non-deterministic distribution of slave requests (tasks have an unknown computational cost). In this context, the master node usually implements a waiting buffer in order to deal with simultaneous arrivals of results from slaves.

3.1 Generic model for m/s parallel programs

Regardless of interaction type, the SAN generic models to m/s parallel programs are composed of an automaton to each node. *Master* automaton (see Fig. 3) is composed of three states:

- ITx : representing the initial transmission of tasks to slave nodes;
- Tx : representing the transmission of new tasks to slave nodes; and
- Rx : representing the reception of computed results from slave nodes.

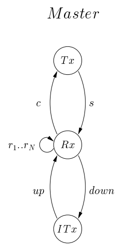


Fig. 3. Master automaton

Each automaton $Slave^{(i)}$ (see Fig. 4), where $i = 1 \dots N$, is also composed of three states:

- I : representing the slave node without tasks to process (idle);

[8] Readers must pay attention that we are *not* talking about synchronicity in the programming primitives level.

- Pr : representing the slave node processing a task; and
- Tx : representing the slave node transmitting results back to the master node.

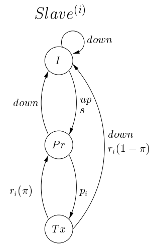


Fig. 4. Slaves automata

There are six distinct events in the generic model composed of those $1+N$ automata: *up*, s, p_i, r_i, c, and *down*. The program execution begins with event *up*. Event s corresponds to the sending of tasks to the slaves. Event p_i represents the processing stop of a task by the i^{th} slave. After this stop, event r_i indicates the sending of results from the i^{th} slave back to the master. If the i^{th} slave needs to continue the task execution, events p_i and $r_i(\pi)$ will occur successively until the ending of the task execution. On the other hand, when there is no more points to be evaluated, the slave transmits a pack through the occurrence of event $r_i(1-\pi)$ and returns to idle state. The numerical value of π must be estimated according to information known by the parallel programmer. After receiving a result from one of the slave nodes, the master node performs the summation represented by event c. When all needed work were done, event *down* represents the ending of the parallel application and the returning to the initial situation. Note that event *down* may happen even if the slaves are in Pr or Tx states.

According to specific characteristics of each modeled application, some additional features must be added to this generic model. For example, in asynchronous models, event s must be split in N events s_i ($i = 1..N$), each one representing the sending of a task to the i^{th} slave. Synchronous models have only one event s, because all slaves receive tasks at the same time. Also, the passage of results from slaves to master is usually performed using a buffer. In that case an additional automaton may be included and the events r_i and c will not directly synchronize master with slaves, but slaves with buffer, and buffer with master respectively (see asynchronous model in Section 5.1).

The numerical value for the rate of each event will depend on a variety of characteristics, *e.g.*, problem size, granularity, nodes processing power, communication speed, *etc.* This mapping of real case information to numerical values, called *parameterization*, is a crucial point in the model development. Unfortunately, such mapping seems quite dependent on the particularity of each implementation. In Section 5, we provide two practical cases of parameterization, one considering a synchronous implementation, and the other considering the analogous asynchronous version of the same application.

4 Case study: The Propagation Algorithm

Image-based interpolation is a method to create smooth and realistic virtual views between two original view points. One of the major computational cost phases of this method is the one related to the construction of a dense matching map between the original scenes [5]. In this section, we briefly discuss some relevant details about the implementation of a region growing algorithm called Propagation [14]. Later, a parallel version for this algorithm is also briefly presented.

4.1 Basic Propagation Algorithm

The Propagation Algorithm is based on a classic region growing method for image segmentation [15] which uses pixel homogeneity. However instead of using pixel homogeneity property, a similar measure based on the matches correlation score is adopted [14]. This propagation strategy could also be justified because the seed pairs are composed by points of interest, which are the local maxima of the textureness. Thus, these matches neighbors are also strongly textured, which allows good propagation even though they are not local maxima.

Points of interest [12, 19] are naturally good seed point candidates because they represent the points of the image which have the highest textureness. These points are detected in each separated image. Next, they are matched using the ZNCC (zero-mean normalized cross correlation) measure. At the end of this phase, a set of seed pairs is ready to be used to bootstrap a region growing type algorithm, which propagates the matches in the neighborhood of seed points from the most textured pixels to the less textured ones.

The neighborhood $N_5(a, A)$ of pixels a and A is defined as being all pixels within the 5x5 window centered at these two points (one window per image). For each neighboring pixel in the first image, a list of match candidates is constructed. This list consists of all pixels of a 3x3 window in the corresponding neighborhood in the second image. The complete definition of the neighborhood $\mathcal{N}(a,\ A)$ of pixel match $(a,\ A)$ is given by:

$$N(a, A) = \{(b, B), b \in \mathcal{N}_5(a), B \in \mathcal{N}_5(A), (B - A) - (b - a) \in \{-1, 0, 1\}^2\}.$$

The input of the algorithm is the set *Seed* that contains the current seed pairs. This set is implemented by a heap data structure for a faster selection of the best pair. The output is an injective displacement mapping *Map* that contains all good matches found by the Propagation Algorithm. Let $s(\mathbf{x})$ be an estimation of the luminance roughness for the pixel $\mathbf{x}$, which is used to stop propagation into insufficiently textured areas. And let t be a constant value that represents the lower luminance threshold accepted on a textured area.

Briefly, all initial seed pairs are starting points of concurrent propagations. At each step, a match $(a,\ A)$ with the best ZNCC score is removed from the

Algorithm 1. The Propagation Algorithm

```
while Seed ≠ ∅ do
   pull the best match (a, A) from Seed
   Local ← ∅
   {store in Local new candidate matches}
   for all (x, y) ∈ 𝒩(a, A) do
      if ((x, ∗), (∗, y) ∉ Map) and (s(x) > t, s(y) > t) and (ZNCC(c, d) > 0.5)
      then
         Local ← (x, y)
      end if
   end for
   {store in Seed and Map good candidate matches}
   while Local ≠ ∅ do
      pull the best match (x, y) from Local
      if (x, ∗), (∗, y) ∉ Map then
         Map ← (x, y), Seed ← (x, y)
      end if
   end while
end while
```

current set of seed pairs. Then, the algorithm looks for new matches in its match neighborhood. When it finds one, it is added to the current seed pairs set and also to the set of accepted matches which is under construction.

An example of the sequential propagation program execution can be observed on Fig. 5. The squared regions in both images show the extension of the matched regions obtained from the seed matches.

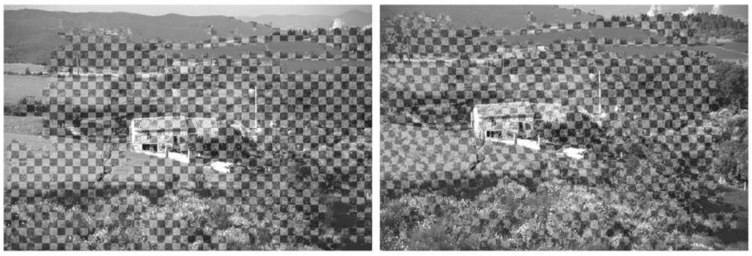


Fig. 5. Propagation example using the House pair

4.2 Parallel implementation

Parallel implementation for the Propagation Algorithm discussed on this section was developed in order to allow the use of this new algorithm on real situations. Thus, it was necessary to achieve better performances without using parallel programming models oriented to very expensive (but not frequently used) machines. Useful parallel versions for this algorithm should run distributed over several processors connected by a fast network. Therefore, the natural choice was a cluster with a message passing programming model.

As presented in the previous section, the Propagation Algorithm advances by comparing neighbors pixels through out the source images. From some seed pairs, it can form large matching regions on both images. In fact, a single seed pair can start a propagation that grows through a large region over the images. This freedom of evolution guarantees the algorithm to achieve good results in terms of matched surfaces. Another characteristic is that the algorithm is based on global best-first strategy to choose the next seed pair that will start a new propagation, which also have a direct effect on the final match quality. These two characteristics are hard to deal with if one wants to propose a parallel distributed version of the algorithm without loosing quality at the final match. The best-first strategy implementation is based on a global knowledge of the seed pairs set, which is not appropriated to a non-shared memory context. In addition, the freedom of evolution through out the images assumes that the algorithm knows the entire surface of the images. This can create a situation in which several processors are propagating over the same regions at the same time creating a redundancy of computation (Fig. 6).

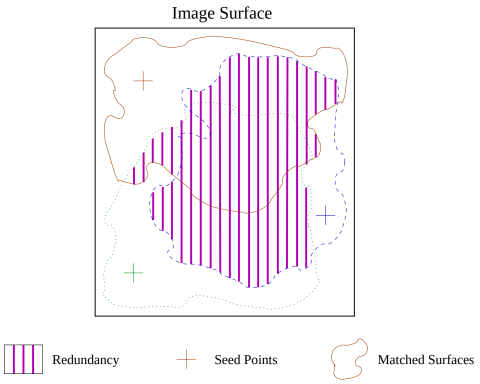


Fig. 6. Redundancy problem

Besides, it is not possible to know in advance how many new matches a seed pair will generate. Thus, from a parallel point of view, the Propagation Algorithm is an irregular and dynamic problem which exhibits unpredictable load fluctuations. Therefore, it requires the use of some load balancing policy in order to achieve a more efficient parallel solution.

The solution proposed in [9] is based on a master-slave paradigm. One processor (master) will be responsible for distributing the work and centralizing the final results. The others (slaves) will be running the propagation algorithm each one using a subset of the seed pairs and knowing a pair of corresponding slices over the images (coordinates of target slice). The master distributes the seed pairs over the nodes considering their location over the slices (see Fig. 7). This procedure replaces the global best-first strategy by several local best-first ones. Each local seed pairs subset is still implemented as a heap which is ordered by the pair ZNCC score. This strategy minimizes the problem of loosing quality at the final match.

Once the problem with the global best-first strategy is solved, it still

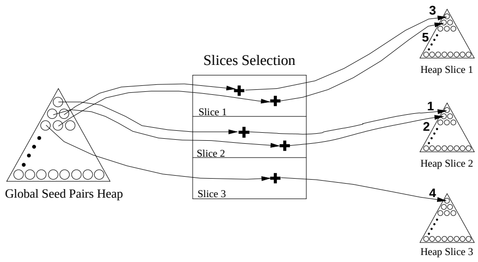


Fig. 7. Seed pairs heap distribution over the slices

remains the problem with the algorithm limitation of evolution over the images. As said before, each node can propagate just over the surface of its associated slice in order to avoid computation redundancy. But, forbidding the evolution out of the associated slice generates two kinds of losses. First, some matches are not done because they are just at the border of one slice and one of its points is placed outside. Second, some regions in one slice may not be reached by any propagation started by a seed pair located inside of its surface, but instead they could be reached by a propagation started at a neighbor slice.

Such limitation is partially solved by a technique called flexible slices [9]. This technique allows the Propagation Algorithm to expand through the surface of its neighbor slices in a controlled way. As shown on Fig. 8, each processor works over its own associated slice, but it also knows its neighbor slices and it has the permission to propagate over them. But still, it is not interesting to leave the Propagation Algorithm free to compute its neighbors entire surface. This may cause the computation of too many repeated matches. To avoid that, each processor has the permission to compute just over a percentage of its neighbors surface. This percentage, called *overlap*, is related to the number of slices. A large number of slices implies in thinner slices. In this case, it is acceptable to allow a processor to advance over a large percentage of its neighbors surfaces. On the other hand, a small number of slices implies in larger slices. Here, the algorithm must not propagate too much over the neighbors' surface.

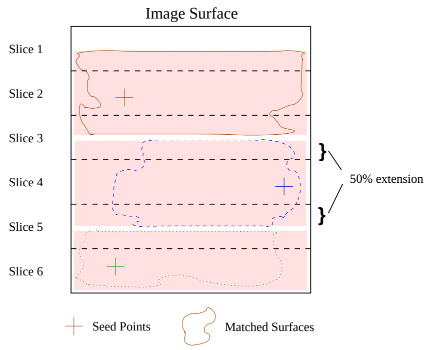


Fig. 8. Flexible slices approach

The last problem to deal with in the parallelization of the Propagation

Algorithm is the load balancing. If the source images are divided into more slices than the number of nodes available, the load balancing strategy adopted is:

(i) the master divides the set of seed pairs into subsets based on their location over the slices;

(ii) each slave receives one slice with its associated subset;

(iii) each slave computes its own subset of seed pairs;

(iv) when there is no more seed pairs to compute, the slave sends a signal to the master;

(v) if there are available slices remaining, the master send it to an available slave;

Actually, the master has a queue of slices, organized by their position over the images. To choose which slice will be sent to an available slave, the master simply gets the first slice of this queue. In fact, the master sends a new seed pairs subset (coordinates of the slice) to the slave.

Finally, it is important to mention that the master must receive all matches generated by the slaves and it must filter unavoidable duplicated ones. To send these final matches to master, each slave has a communication buffer which is filled progressively as the Propagation Algorithm advances. A buffer, when it is full, is sent to master and the slave immediately returns to its execution. All slaves do the same procedure, in a way that forces the master to have a receiving queue. When a slave reaches the end of its seed pairs subset, it sends an incomplete buffer to the master. When the master receives an incomplete buffer, it knows that the sender has finished its work and sends a new slice (seed pairs subset) back to it (if there are available subsets).

5 Modeling examples

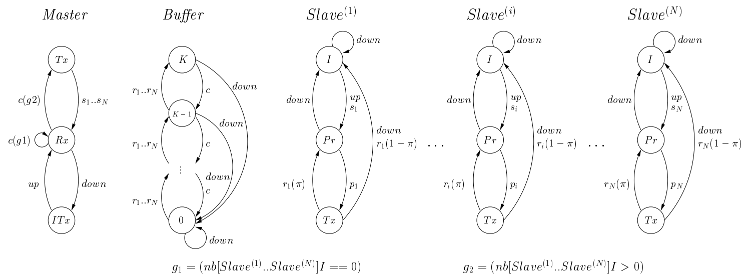
This section proposes two distinct models to two quite different approaches to parallelisation of the Propagation Algorithm presented previously.

Our analysis of the sequential Propagation Algorithm indicates that a m/s implementation would be more efficient using asynchronous interaction between the master and the slaves, *i.e.*, the master does not distribute and collect results from the slaves in phases. In this section, we use SAN to model both synchronous and asynchronous m/s implementations in order to confirm (in Section 6) if our pre-implementation analysis of the Propagation Algorithm is corrected.

5.1 Asynchronous implementation model

Fig. 9 presents the SAN asynchronous model which contains one *Master* automaton, one *Buffer* automaton, and N $Slave^{(i)}$ $(i = 1..N)$ automata.

Master automaton is responsible for the distribution of slices (*tasks*) to



Type	Event	Rate
syn	*up*	λ
syn	*down*	μ
syn	c	σ
syn	s_i	δ
syn	r_i	α
loc	p_i	γ

Fig. 9. SAN asynchronous model for the Propagation Algorithm

slaves and analysis of final matches (*results*) evaluated by them. This automaton has three states (ITx, Tx and Rx) which mean respectively: the initial transmission of all seed pairs to the slaves; the transmission of new slices to the slaves; and the reception of final matches evaluated by the slaves. Synchronizing event *up* sends the initial slices to all slaves, starting a new program execution. On the other hand, synchronizing event *down* ends one execution of an application. The occurrence of this event indicates that all automata must change their actual state for the initial one.

Synchronizing event s_i represents the sending of a new slice to the i^{th} slave. Automaton *Master* consumes final matches evaluated by the slaves through synchronizing event c, consuming results from a repository (automaton *Buffer*). Since this SAN model has asynchronous characteristics, the results generated by the slave nodes are stored on this repository. Therefore, master node does not need to synchronize the reception of results sent by slaves.

When synchronizing event c occurs, the master node may have two behaviors. In the first one, indicated by functional probability [9] g_1, master stays on receiving and analyzing results. In the second one, functional probability g_2 indicates that a slave node is in idle state, so master must send him a new slice.

Finally, automaton $Slave^{(i)}$ represents the i^{th} slave and it has three states: I (*idle*), Pr (*processing*) and Tx (*transmission*). $Slave^{(i)}$ finishes a final match through the occurrence of local event p_i. Synchronizing event r_i represents the reception of final matches from $Slave^{(i)}$ by *Buffer*. The slave transmits a pack (final matches) and returns to Pr state with a probability π, when there still exists points to be evaluated. On the other hand, when there is no more points to be evaluated, the slave transmits a pack with probability $1-\pi$ and returns to idle state. The numerical values of π and the rates of all events must be estimated according to information known by the parallel programmer. In the next section, we present how to map this information into model parameters.

[9] Using the SAN notation employed by the PEPS tools [4], the command $nb[Slave^{(1)}..Slave^{(N)}]I$ computes how many automata $Slave^{(i)}$ ($i = 1..N$) are in I (idle) state.

5.1.1 Assigning parameters

This section shows how to assign numerical values to the event rates and probabilities. Some parameters are given by the developer (input values of the model), whereas other are evaluated using those input values. We will describe the choice of those values considering the processing of the image in Fig. 5, and the characteristics of an homogeneous COW (Cluster of Workstations) with Pentium III 1.0 GHz and Myrinet network. Obviously, other environments or other workloads would have different input parameters.

Some variables were created to help the parameter definition. Let BL be the slaves buffer length, which was fixed with corresponding value of $64KB$ size ($\simeq$ 5.500 points). The percentage of slices extension of one slave over its neighborhoods (PS) was also fixed with value 0.5. Another fixed value is the number of points (NP) on the input image: 230.000 points [10]. Obviously, the quality of the prediction is quite dependant on the accuracy of such input parameters. Finally, number of slices (NS) describes the number of tasks in which the problem will be split (granularity).

From those input values, we can calculate the real number of points (RNP) to be evaluated (NP plus redundancy):

$$RNP = \Big[2 * (1 + PS) + (NS - 2) * (1 + 2PS)\Big] * \frac{NP}{NS} \tag{8}$$

Hence, it was easy to estimate the average number of buffers (NB) for each slice:

$$NB = \left\lceil \frac{RNP}{BL * NS} \right\rceil \tag{9}$$

Probability π of the event r_i (automaton $Slave^{(i)}$) is given by:

$$\pi = \frac{NB - 1}{NB} \tag{10}$$

Rates for events r_i, p_i, s_i and c were obtained with some statistical measurements performed through some sample programs over the target architecture. Rate of synchronizing events s_i (δ) and r_i (α) correspond, respectively, to send a new task (TT) to a slave and send back a result pack (TP) containing final matches, given by:

$$\delta = \frac{1}{TT} \;\; and \;\; \alpha = \frac{1}{TP} \tag{11}$$

In a similar way, rate of events p_i (γ) and c (σ) correspond to the time spent on computing final matches (CM) and evaluating results (ER) respectively. Those event rates are defined by:

$$\gamma = \frac{1}{CM} \;\; and \;\; \sigma = \frac{1}{ER} \tag{12}$$

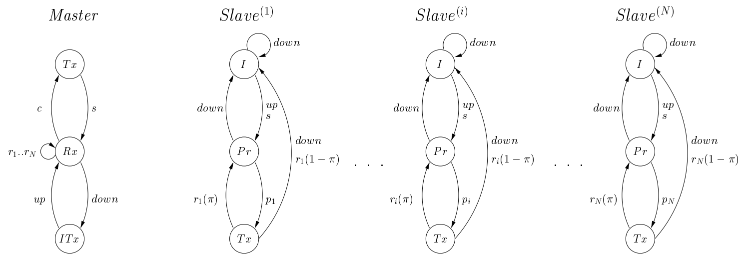
[10] This number of points was taken from the real image presented in Fig. 5.

Synchronizing event *down* (μ) has an insignificant time, so an arbitrary high rate ($\mu = 1000$) was adopted. Finally, synchronizing event *up* is associated with the time spent on sending tasks to slaves, similar as event s_i. Thus, its rate (λ) can be obtained by:

$$\lambda = \frac{1}{TT * N} \text{ , } where\ N = number\ of\ slaves \tag{13}$$

5.2 Synchronous implementation model

Fig. 10 presents the SAN synchronous model for the parallel implementation of the Propagation Algorithm. SAN synchronous model is quite similar to SAN asynchronous version. However, it is not necessary to consider a repository buffer (automaton *Buffer*), since the master node forces all slaves to finish their tasks before it starts a new task distribution.



Type	Event	Rate
syn	*up*	λ
syn	*down*	μ
loc	c	f_c
syn	s	δ
syn	r_i	α
loc	p_i	γ

$f_c = (nb[Slave^{(1)}..Slave^{(N)}]I == N) * \sigma$

Fig. 10. SAN synchronous model for the Propagation Algorithm

Since there is no buffer to store the slave results, which characterizes asynchronous model, master node is now responsible for reception of all packs evaluated by the slaves. Hence, events $r_1..r_N$ (automaton *Master*) describe this new behavior.

Additionally, synchronous model has all tasks in each phase finishing before the distribution of next phase tasks. This behavior is indicated by local event c, which has a functional rate (f_c) indicating that a new task distribution only must start when all slaves are in idle state. Synchronizing event s describes a new task distribution, *i.e.*, the starting of a new phase.

5.2.1 Assigning parameters

All events, which do not have any change from asynchronous to synchronous model, preserve their rates (*up*, *down*, s, and p_i). For the other ones, minimal changes have been done. Event c is now a local event and it has a functional rate (function f_c indicated in Fig. 10), expressing the waiting for synchronized distribution of new phase tasks.

Events $s_1..s_N$ were replaced for just one synchronizing event s. Since event s is related to a *broadcast* of new tasks, and inasmuch the time for a broadcast is similar to an *unicast*, its rate was preserved.

As the master node centralizes the receiving of all pack results, two or more slaves can send their results to master at the same time. Hence, the sending time of a slave declines drastically. Thus, rate of events $r_1..r_N$ (α) are now defined by:

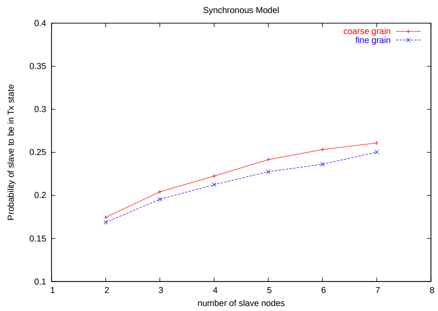
$$\alpha = \frac{1}{TP * N} \tag{14}$$

6 Performance indices

In order to validate the strategy of the implementation chosen and modeled using SAN, some models were made altering some parameters, *e.g.*, automaton *Buffer* size, number of slices (tasks), and number of slaves. With the results of those models, three kind of analysis could be done: synchronous *vs.* asynchronous; small buffer *vs.* large buffer; and fine grain *vs.* coarse grain. Despite of having no doubt about which approach (synchronous or asynchronous) fits better, the two models were solved and their theoretical results were compared. Finally, the size of automaton *Buffer* and the granularity were analyzed.

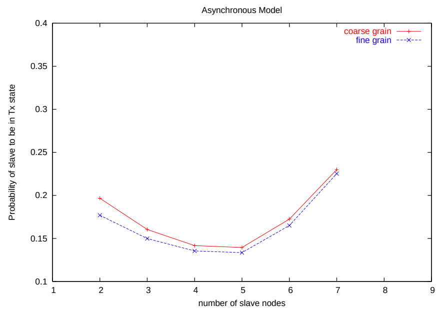
6.1 Synchronous vs. Asynchronous

Fig. 11 presents a comparison between synchronous and asynchronous models, observing the probability of a slave to be in Tx state. First analysis about synchronous model shows the probability of a slave to be (or trying to be) transmitting gets higher while the number of slaves grows. This analysis indicates that slaves stay more time transmitting, which is a coherent result, whereas master node centralizes all the receives. The behavior in asynchronous model is quite different. Up to five slaves, increasing the number of slaves do not result in an increasing Tx probability. This indicates an advantage for asynchronous implementation, since slaves can effectively work more time in parallel. We believe that it happens due to a smaller loss in transmission of result packs.



Number of Slaves	2	3	4	5	6	7
Coarse Grain	0.1746	0.2042	0.2225	0.2416	0.2533	0.2609
Fine Grain	0.1687	0.1954	0.2125	0.2275	0.2362	0.2503

(a)

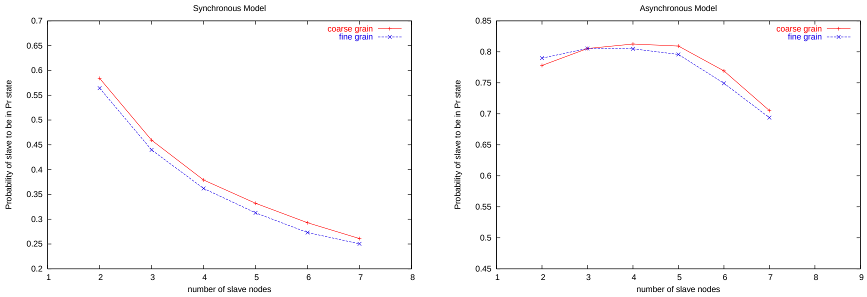


Number of Slaves	2	3	4	5	6	7
Coarse Grain	0.1967	0.1604	0.1417	0.1395	0.1725	0.2300
Fine Grain	0.1769	0.1499	0.1355	0.1336	0.1651	0.2253

(b)

Fig. 11. Probability of slaves to be in Tx state

This analysis can be confirmed by looking at the probability of a slave to be in Pr state. As it can be seen in Fig. 12 (b), the probability of Pr state (computing tasks) decreases as the number of slaves grows in asynchronous implementation.



Number of Slaves	2	3	4	5	6	7
Coarse Grain	0.5840	0.4594	0.3790	0.3322	0.2929	0.2609
Fine Grain	0.5643	0.4398	0.3620	0.3128	0.2731	0.2503

(a)

Number of Slaves	2	3	4	5	6	7
Coarse Grain	0.7780	0.8053	0.8126	0.8093	0.7692	0.7052
Fine Grain	0.7898	0.8054	0.8049	0.7958	0.7492	0.6938

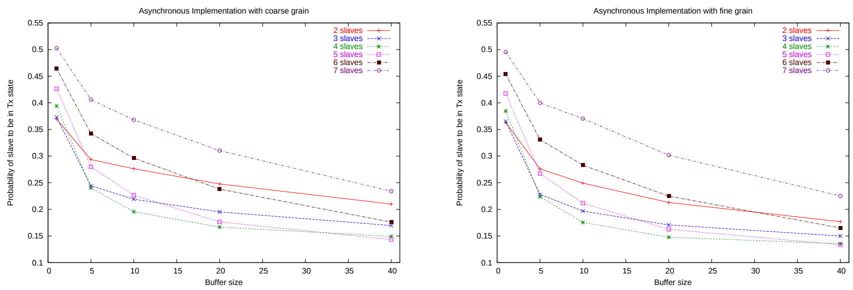
(b)

Fig. 12. Probability of slaves to be in Pr state

Additionally, performance gains once again keep increasing up to five slaves. This analysis do not indicate that we have a worse execution time with more than five slaves, but at this point the speed up starts to decline. The performance evaluation based on the SAN models indicates better results for asynchronous implementation of the Propagation Algorithm. Therefore, the next experiments will only consider asynchronous implementations.

6.2 Small buffer vs. Large buffer

For asynchronous model, an important parameter is the size of automaton *Buffer* (master node waiting queue). Fig. 13 shows curves for asynchronous model with fine and coarse grain.



Buffer size	2 Slaves	3 Slaves	4 Slaves	5 Slaves	6 Slaves	7 Slaves
1	0.3696	0.3729	0.3940	0.4264	0.4643	0.5025
5	0.2935	0.2443	0.2404	0.2799	0.3423	0.4061
10	0.2763	0.2190	0.1956	0.2266	0.2963	0.3680
20	0.2477	0.1953	0.1668	0.1765	0.2380	0.3103
40	0.2099	0.1696	0.1492	0.1434	0.1761	0.2341

(a)

Buffer size	2 Slaves	3 Slaves	4 Slaves	5 Slaves	6 Slaves	7 Slaves
1	0.3630	0.3652	0.3846	0.4176	0.4542	0.4954
5	0.2760	0.2280	0.2237	0.2672	0.3308	0.3999
10	0.2493	0.1969	0.1754	0.2116	0.2830	0.3703
20	0.2129	0.1708	0.1478	0.1626	0.2249	0.3017
40	0.1769	0.1499	0.1355	0.1336	0.1651	0.2253

(b)

Fig. 13. Buffer size for asynchronous model

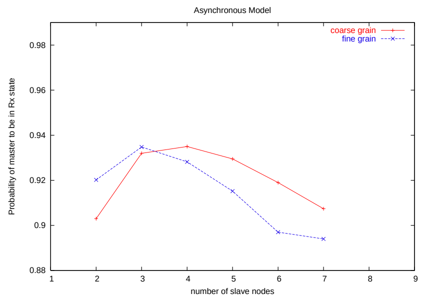
In both cases, a small buffer (less than 10 positions) shows slaves remaining too much time in Tx state (waiting for buffer space) and, consequently, less time calculating. On the other hand, increasing the buffer size indefinitely does not result in a better performance. The best buffer size depends directly on the number of slaves used. A 20-sized buffer seems to be enough using 2, 3, and 4 slaves. However, using more slaves, a larger buffer may be required, *e.g.*, a 40-sized buffer seems the most appropriated for 7 slaves. Therefore, for the next experiments we will consider a 40-sized buffer.

As can be seen in Fig. 13, probabilities presented for 1-sized buffer are almost the same as for synchronous model. This is a coherent result since synchronous model could be approximated by an asynchronous model with 1-sized buffer.

6.3 Fine grain vs. Coarse grain

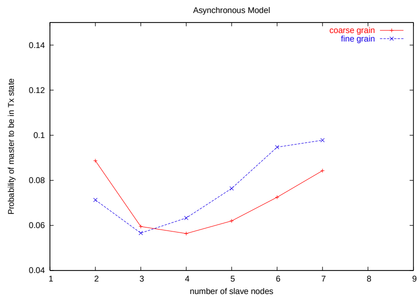
In parallel applications, the granularity is one of the most important features. In our study, granularity is represented by the number of slices (NS) in which the input image is partitioned. We use two different grains defined through the number of slices allowed per node: coarse grain (2 slices per node) and fine grain (3 slices per node).

Looking at Fig. 11 (b) and 12 (b), it is not possible to point out which granularity is better. Using fine grain, slaves spent less time calculating (Pr state) and less time transmitting (Tx state). On the other hand, using coarse grain, the situation is the inverse.



Number of Slaves	2	3	4	5	6	7
Coarse Grain	0.9030	0.9320	0.9350	0.9295	0.9190	0.9074
Fine Grain	0.9202	0.9348	0.9282	0.9152	0.8970	0.8940

(a)



Number of Slaves	2	3	4	5	6	7
Coarse Grain	0.0887	0.0595	0.0564	0.0620	0.0725	0.0843
Fine Grain	0.0713	0.0566	0.0633	0.0764	0.0947	0.0978

(b)

Fig. 14. Granularity for asynchronous model

In fact, the main aspect to identify the best granularity is related to automaton *Master*. For parallel implementation of the Propagation Algorithm described in Section 4.2, it is clear that, if master spends too much time transmitting, it quickly becomes the bottleneck of the application. Based on this assertion, the best granularity would be the one where master remains most of its time in Rx state. Increasing the number of slave nodes (see Fig. 14 (a)), master node remains more time receiving information with coarse grain.

Such analysis can be confirmed in Fig. 14 (b) in which the probability of master to be in Tx state is presented. Therefore, from our asynchronous model, we can assume that using coarse grain will result in a better speed up for the parallel implementation.

7 Conclusion

The modeling of such parallel programs is greatly facilitated by SAN formalism primitives. However, any other structured formalism, *e.g.*, *Stochastic Petri Nets* [7], Stochastic Activity Networks [18], and *Process Algebra* [11] could be employed. Thus, according to the programmer previous experience, the use of another formalism could be considered without any loss of generality. Balbo, Donatelli and Franceschinis, for example, had use a quite similar approach using SPN formalism [3]. In fact, the choice of SAN as modeling formalism was somewhat driven by our own experience with previous modeling examples [8]. Our main point is to show that the modeling of parallel programs can be generalized according to a given structured formalism, which was *Stochastic Automata Networks* for this paper.

The use of an automaton to describe each processing node (being a master or slave) and the buffer automaton (for asynchronous models) or a single synchronizing event (for synchronous models) seems quite intuitive. Such simplicity is the greater advantage of our modeling technique.

The major drawback of this approach is the importance left to the parameterization phase, *i.e.*, to the choice of rates for the model events. Much of the accuracy of the model rely on the mapping of user known characteristics to the rate numerical values of the events. In fact, the modeling technique may only help the programmer by posing the questions of what event rates should be informed. It remains on the programmer shoulders the burden of taking all important parameters into account.

The great advantages in our technique are the normal benefits of having a formal model for the application. The indication of possible *bottlenecks* may help the programmer to pay more attention to those specific points during the implementation. Results as those presented in the previous section may clearly point out the better number of tasks according to the number of processing nodes. Such information about the best granularity may be useful to a scheduler, or a processing node provider, to automatically choose how many nodes in a cluster or grid should be allocated to a given application.

The obtained results for the models of the Propagation Algorithm were not yet confirmed by a direct comparison with an actual implementation. Nevertheless, our results match the expectations of the algorithm programmers. Thus, the comparison with actual implementations, and other modeling experiences, are the natural future works to pursue. Equally, we may foresee the development of an user friendly tool to facilitate the construction of models to programmers with few, or none, performance evaluation skills. Despite the lack of more extensive practical experiences, we believe that the use of SAN analytical models for parallel implementations seems an useful and economic viable helping theoretical tool to programmers.

References

[1] Andrews, G. R., "Foundations of Multithreaded, Parallel, and Distributed Programming," Wesley, University of Arizona, USA, 2000.

[2] Bailey, D. H., E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, D. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan and S. K. Weeratunga, *The NAS Parallel Benchmarks*, The International Journal of Supercomputer Applications **5** (1991), pp. 63–73.

[3] Balbo, G., S. Donatelli and G. Franceschinis, *Understanding parallel programs behavior through Petri nets models*, Journal of Parallel and Distributed Computing **15** (1992), pp. 171–187.

[4] Benoit, A., L. Brenner, P. Fernandes, B. Plateau and W. J. Stewart, *The PEPS Software Tool*, in: P. Kemper and W. H. Sanders, editors, *Computer Performance Evaluation/TOOLS 2003*, Lecture Notes in Computer Science **2794** (2003), pp. 98–115.

[5] Brenner, L., L. G. Fernandes, P. Fernandes and A. Sales, *Performance Analysis Issues for Parallel Implementations of Propagation Algorithm*, in: *Proceedings of the* 15^{th} *Symposium on Computer Architecture and High Performance Computing*, São Paulo, 2003, pp. 183–190.

[6] Brenner, L., P. Fernandes and A. Sales, *The Need for and the Advantages of Generalized Tensor Algebra for Kronecker Structured Representations*, International Journal of Simulation: Systems, Science & Technology **5** (2004), pp. 1–13.

[7] Ciardo, G. and K. S. Trivedi, *A Decomposition Approach for Stochastic Petri Nets Models*, in: *Proceedings of the* 4^{th} *International Workshop Petri Nets and Performance Models* (1991), pp. 74–83.

[8] Farina, A. G., P. Fernandes and F. M. Oliveira, *Representing software usage models with Stochastic Automata Networks*, in: *Proceedings of the* 14^{th} *International Conference on Software Engineering and Knowledge Engineering* (2002), pp. 401–407.

[9] Fernandes, L. G., "Parallélisation d'un Algorithme d'Appariement d'Images Quasi-dense," Ph.D. thesis, Institut National Polytechnique de Grenoble, France (2002).

[10] Getov, V., A. Hey, R. Hockney and I. Wolton, *The Genesis Benchmark Suite: Current State and Results*, in: *Proceedings of Workshop on Performance Evaluation of Parallel Systems – PEPS'93*, University of Warwick, Coventry, 1993.

[11] Gilmore, S., J. Hillston, L. Kloul and M. Ribaudo, *PEPA nets: a structured performance modelling formalism*, Performance Evaluation **54** (2003), pp. 79–104.

[12] Harris, C. and M. Stephens, *A Combined Corner and Edge Detector*, in: *Proceedings of the* 4^{th} *Alvey Vision Conference*, Manchester, UK, 1988, pp. 147–151.

[13] Kraemer, E. and J. T. Stasko, *The Visualization of Parallel Systems: An Overview*, Journal of Parallel and Distributed Computing **18** (1993), pp. 105–117.

[14] Lhuillier, M., "Modlisation pour la synthse d'images partir d'images," Thse de doctorat en informatique, Institut National Polytechnique de Grenoble, France (2000).

[15] Monga, O., *An Optimal Region Growing Algorithm for Image Segmentation*, International Journal of Pattern Recognition and Artificial Intelligence **1** (1987), pp. 351–375.

[16] Plateau, B., *On the stochastic structure of parallelism and synchronization models for distributed algorithms*, in: *Proceedings of the 1985 ACM SIGMETRICS conference on Measurements and Modeling of Computer Systems* (1985), pp. 147–154.

[17] Plateau, B. and K. Atif, *Stochastic Automata Networks for modelling parallel systems*, IEEE Transactions on Software Engineering **17** (1991), pp. 1093–1108.

[18] Sanders, W. H. and J. F. Meyer, *Reduced Base Model Construction Methods for Stochastic Activity Networks*, IEEE Journal on Selected Areas in Communications **9** (1991), pp. 25–36.

[19] Schimid, C., R. Mohr and C. Bauckhage, *Comparing and Evaluating Interest Points*, in: *Proceedings of the 6th International Conference on Computer Vision*, Bombay, India, 1998, pp. 230–235.

[20] Stewart, W. J., "Introduction to the numerical solution of Markov chains," Princeton University Press, 1994.

[21] van Gemund, A. J. C., *Compiling performance models from parallel programs*, in: *Proceedings of the 8th International Conference on Supercomputing* (1994), pp. 303–312.

[22] Waheed, A. and D. T. Rover, *Performance Visualization of Parallel Programs*, in: *IEEE Conference on Visualization*, San Jose, USA, 1993, pp. 174–181.

[23] Yan, J. C., S. Sarukkai and P. Mehra, *Performance measurement, visualization and modeling of parallel and distributed programs using the AIMS toolkit*, Software Practice and Experience **25** (1995), pp. 429–461.

Securing statically-verified communications protocols against timing attacks

Mikael Buchholtz [a,1] Stephen Gilmore [b,2] Jane Hillston [b,3] Flemming Nielson [a,4]

[a] *Informatics and Mathematical Modelling*
The Technical University of Denmark, Lyngby, Denmark

[b] *Laboratory for Foundations of Computer Science*
The University of Edinburgh, Edinburgh, Scotland

Abstract

We present a federated analysis of communication protocols which considers both security properties and timing. These are not entirely independent observations of a protocol; by using timing observations of an executing protocol it is possible to calculate encryption keys which were intended to be secret or to deduce derived information about the nature of the communication even in the presence of unbreakable encryption. Our analysis is based on expressing the protocol as a high-level model and deriving from this process calculus models analysable by the Imperial PEPA Compiler and the LySatool.

1 Introduction

Through the use of strong encryption and well-designed communications protocols it is possible to engineer secure systems which allow a sender and a receiver to exchange confidential messages supported by authentication guarantees which confirm the identity of the principals in the communication. A very high level of confidence in the correctness of protocols such as these can be obtained by the application of static analysis methods to detect unwanted control flows. Such an analysis can ensure the absence of information flows from the sender or recipient to an attacker who is trying to eavesdrop on their conversation or to interfere through inserting bogus messages or otherwise subverting meaningful information flow between the sender and the receiver.

[1] Email: mib@imm.dtu.dk
[2] Email: stg@inf.ed.ac.uk
[3] Email: jeh@inf.ed.ac.uk
[4] Email: nielson@imm.dtu.dk

This is the workshop version of the paper. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

Supported by strong and justified belief in the encryption techniques used and the communication protocol deployed it might be tempting to conclude that no information could be leaked from the communication and gained by the attacker. However, digital communications are susceptible to a particularly subtle form of attack which is not always well defended against. Those attacks in which information is leaked through an inference obtained from timing a secure interaction are known as *timing attacks*.

The effects of a successful timing attack can be catastrophic. A successful timing attack can reveal the encryption keys which secure protocols depend upon. Messages believed to be private can now be read by an attacker. Additionally, an attacker can forge digitally signed messages and thereby entirely subvert the message authentication process.

It was previously believed to be the case that timing attacks could only be applied to hardware security tokens such as smartcards where they had been shown to be able to expose the secret keys used for RSA decryption [20]. In the context of a networked computing application it was hoped that the unpredictable delays which are inherent to networks would introduce sufficient "noise" to make timing attacks impractical in such a setting. Unfortunately this hope was shown to be misplaced when Brumley and Boneh were able to mount a remote timing attack against OpenSSL [8]. By measuring the time which a server takes to respond to decryption queries Brumley and Boneh's attack client is able to extract the private key stored on the server. Their attack applies in networked environments but also in interprocess and virtual machine environments as well.

Many cryptographic libraries completely ignore the problem of timing attacks and have no defenses implemented to prevent it, so this is not merely a problem with one particular implementation of a secure protocol but instead is a general problem which is not widely understood or one which is being methodically fought using well-founded analysis methods.

In the context where developers have access to the source code of the implementation of their encryption routine then it might be possible to add an implementation of *RSA blinding*, which draws from a random number distribution and applies encryption to this in order to add a random delay to the time taken per encryption step. Somewhat problematically, this method requires developers to have the ability and expertise to implement this operation correctly. In practice, many application programmers are unfamiliar with the details of encryption routines, and there is genuine reason for concern that they could compromise the integrity of the encryption in the process of modifying its implementation.

Timing attacks can be mounted by timing the exchange of messages during a session between two parties. The parts of the communication which do not require user interaction have sufficient repeatability that timings of these interactions can be used to derive information. Any user interaction, even a single keystroke or button click, introduces sufficient random delay to mask any information which could have been obtained from the timing information. Indeed, measuring the time between user keystrokes is used as a source of unrepeatable, weakly correlated random numbers in seeding many cryptographic routines.

Where a secure session between two parties can be monitored by an eavesdropper then information about the time of parts of the interaction can be recorded. In the case where there is a difference in the time taken by a successful interaction and one which fails then one bit of information is leaked from the communication, namely whether this session ended in success or failure. One scenario in which this single bit of information might be of some value is if an eavesdropper is timing the exchange of messages during a session between a buyer and a seller. The longer interaction could correspond to buying shares (successful completion) and the shorter one could correspond to getting a quote for a share price, with no attendant purchase (the failure outcome). If they are unable to break the encryption which is used then the eavesdropper can infer that some shares have been purchased, but not how much, or by whom.

An alternative method of combating timing attacks against secure sessions such as the one described above—and one which does not depend on a deep understanding of how to securely modify sophisticated encryption routines—is to introduce delays in the faster of the important interactions in order to mask the difference between a slower and a faster interaction.

It might seem strange to deliberately insert delays into a protocol. Surely we should be concerned with making the protocol run at top speed and therefore we should speed up the execution of whichever of the success or failure interactions takes the longer time. That might be possible to achieve if we have some additional insights into how the longer-running interaction can be made more efficient (using better algorithms for encryption or authentication, or compressing the transmitted data using lossless compression, or some other means). While that might be the most desirable option in an ideal world in practice we may not have better authentication protocols. In addition, to achieve a faster encryption rate we might be tempted to use lower grade encryption and so such an approach would even give rise to the possibility that the security of the system is compromised by trying to reduce the time taken by the slower of the two interactions.

In contrast to the above we can always make the faster route slower by introducing delays. This requires the invention of no novel algorithms with untested security and performance behaviour and no utilisation of previously-untried authentication protocols. For this reason we frame the problem in the terms of padding out a protocol with delays but our method is equally well applicable to the case where we do have the insight or skill to speed up the slower of the success and failure paths to come in line with the other.

The modification brought about by adding delays to a protocol gives rise to two interrelated questions:

(i) is the protocol with the additional delays still a secure protocol or have the delays themselves somehow compromised its security; and

(ii) do the delays which are inserted actually achieve the effect of masking the difference between a successfully completed session and one which was unsuccessful?

In order to be able to address these questions we need to have a way to efficiently re-run previously-developed security and performance analyses of a protocol.

It would be possible to take an engineering approach to attempting to determine the answer to these questions by padding an implementation of the system with delays and profiling it over multiple success and failure runs. Repeated testing in this way could give us some confidence that padding with delays had achieved the desired effect but it is difficult to believe that we could achieve the same level of confidence as could be achieved using a well-founded formal approach.

To perform formal analysis of security protocols we need a formal framework in order to describe the protocols and to conduct analysis of these descriptions. Rather than developing an entirely new framework, we make use of already-developed modelling formalisms and analysis tools. We use the process calculi LySa [11] and PEPA [18] for security and performance analysis, respectively. The software tools which we use to process our models in these calculi are the LySatool [9] and the Imperial PEPA Compiler (IPC) [6, 5] with DNAmaca [19].

Structure of this paper:

We continue with an overview of the process calculi which we use in this paper, examining their similarities and differences. In Section 3 we describe our method of working with these calculi in order to achieve a federated analysis of security and performance. In Section 4 we present the case study of the paper, a secure mobile m-commerce application in which the transactions between the buyer and the seller are secured using the "Wide-Mouthed Frog" protocol. A discussion of related work follows in Section 5 and the conclusions of the paper are presented after that.

2 Overview of the calculi

The LySa and PEPA calculi are used for different analysis purposes. LySa models can be used to determine if a protocol is secure against malicious (or accidental) harm, guaranteeing trustworthy exchange of information even in the presence of the hardest attacker. PEPA models can be used to determine if a protocol with continue to perform under malicious (or accidental) overloading, guaranteeing timely exchange of information even in the presence of increasing network traffic.

Due to their different ends, LySa and PEPA offer the modeller different linguistic constructs. In this section we contrast the features of the PEPA and LySa process calculi through the use of a common example. The intention in this section is to give the reader sufficient familiarity with the notation to be comfortable with the remainder of the paper. If further details are needed they can be found in Appendix A and B.

We begin with a simple example which models a user printing documents. The user issues `lpr` print commands which send a document to the print daemon. The print daemon converts them to PostScript and spools them to a printer which prints them. In PEPA the rates at which these events happen are quantified. The producer and consumer roles in a communication are distinguished because the producer (generating the work) determines its rate and the consumer passively accepts this (without specifying the rate).

$$\begin{aligned}
User &\stackrel{\mathit{def}}{=} (lpr, r_l).User \\
Daemon &\stackrel{\mathit{def}}{=} (lpr, \top).(toPS, r_{PS}).(spool, r_s).Daemon \\
Printer &\stackrel{\mathit{def}}{=} (spool, \top).(print, r_p).Printer \\
System &\stackrel{\mathit{def}}{=} (User \underset{\{lpr\}}{\bowtie} Daemon) \underset{\{spool\}}{\bowtie} Printer
\end{aligned}$$

We present the LySa version of this example next. The LySa version looks very similar although there are a number of small differences in the presentation:

(i) LySa uses process replication (via the ! operator) instead of binding names to processes and implementing looping behaviour using recursion;

(ii) in PEPA co-operation is symmetric whereas in LySa inputs and outputs are syntactically distinguished, as in the π-calculus;

(iii) the parallel composition operator is not indexed in LySa, processes synchronise on all of their common names; and

(iv) there are no individual actions in LySa, so we take the step of introducing a process which models the PEPA environment and has the function of matching the individual activities to make an input/output pair.

To facilitate the comparison step-by-step we introduce syntactic abbreviations for process terms using $\equiv$ though technically this notation is not a part of LySa, which has nameless processes.

$$\begin{aligned}
User &\equiv\ !\langle lpr, r_l\rangle.\mathsf{0} \\
Daemon &\equiv\ !(lpr, r_l;\ ).\langle toPS, r_{PS}\rangle.\langle spool, r_s\rangle.\mathsf{0} \\
Printer &\equiv\ !(spool, r_s;\ ).\langle print, r_p\rangle.\mathsf{0} \\
Environment &\equiv\ !(toPS, r_{PS};\ ).\mathsf{0}\ |!(print, r_p;\ ).\mathsf{0} \\
System &\equiv User \mid Daemon \mid Printer \mid Environment
\end{aligned}$$

Co-operations in PEPA have been replaced by pattern matching in the LySa model. Patterns such as those used above require inputs and outputs to be identical in order to match. More specifically, the LySa input requires everything before the semi-colon to be component-wise identical to the values in an output for the synchronisation to take place. PEPA's actions are made up of an activity type and an activity rate. These are represented by tuples of size two in the LySa model.

We could observe that individual activities should not be seen by an attacker (because they do not involve activities which cross the network). If we did not care about the timing behaviour of the protocol we could remove all of the individual activities and therefore the environment as well to have a much simpler LySa model.

$$\begin{aligned}
\mathit{User} &\equiv\ !\langle \mathit{lpr}, r_l\rangle.\mathsf{0} \\
\mathit{Daemon} &\equiv\ !(\mathit{lpr}, r_l;\ ).\langle \mathit{spool}, r_s\rangle.\mathsf{0} \\
\mathit{Printer} &\equiv\ !(\mathit{spool}, r_s;\ ).\mathsf{0} \\
\mathit{System} &\equiv \mathit{User} \mid \mathit{Daemon} \mid \mathit{Printer}
\end{aligned}$$

However, in this paper we certainly do care about the timing behaviour of protocols and so we retain the individual activities of the PEPA model now and later.

In stochastic process algebras it is usual to abstract away the data values which would be passed from process to process. However, in an analysis of security and confidentiality it is crucial to know which messages are being passed between the principals in the interaction. For this reason we now progress to consider a notational extension of PEPA, PEPA with value passing and function symbols. In this version of our model of a user printing documents we supplement the action types with an annotation which gives the local name of the data which is being passed. Thus the user refers to his file as *myFile*, the daemon calls this the *userFile*, and the printer sees it after conversion to PostScript as the *spoolFile*.

$$\begin{aligned}
\mathit{User} &\stackrel{\mathit{def}}{=} (\mathit{lpr}(\mathit{myFile}), r_l).\mathit{User} \\
\mathit{Daemon} &\stackrel{\mathit{def}}{=} (\mathit{lpr}(\mathit{userFile}), \top). \\
&\qquad\qquad (\mathit{toPS}(\mathit{userFile}), r_{PS}).(\mathit{spool}(\mathit{userFile}), r_s).\mathit{Daemon} \\
\mathit{Printer} &\stackrel{\mathit{def}}{=} (\mathit{spool}(\mathit{spoolFile}), \top).(\mathit{print}(\mathit{spoolFile}), r_p).\mathit{Printer} \\
\mathit{System} &\stackrel{\mathit{def}}{=} (\mathit{User} \underset{\{lpr\}}{\bowtie} \mathit{Daemon}) \underset{\{spool\}}{\bowtie} \mathit{Printer}
\end{aligned}$$

In contrast to PEPA, LySa supports value passing in its syntax. Values are passed through input/output pairs where passing a value involves the exact *matching* of values to ensure that the correct inputs and outputs are being married and the *binding* of formal parameter names to actual parameter names. In the example below the match on the lpr, r_l activity gives rise to the binding $\{\, userFile \mapsto myFile \,\}$. The names used within a pattern in an input action prefix are syntactically separated from the variables used there by a semi-colon.

$$
\begin{aligned}
User &\equiv\ !\langle lpr, r_l, myFile\rangle.0\\
Daemon &\equiv\ !(lpr, r_l;\ userFile).\langle toPS, r_{PS}, userFile\rangle.\langle spool, r_s, userFile\rangle.0\\
Printer &\equiv\ !(spool, r_s;\ spoolFile).\langle print, r_p, spoolFile\rangle.0\\
Environment &\equiv\ !(toPS, r_{PS};\ PSFile).0 \mid\ !(print, r_p;\ printFile).0\\
System &\equiv User \mid Daemon \mid Printer \mid Environment
\end{aligned}
$$

This brief example has illustrated some differences between the two process calculi used here. We will discuss the encryption and decryption primitives in the LySa calculus in a later section.

3 Methodology

Distilled to their essence, process algebras record a causal ordering on the events in a discrete-event system formed as a composition of concurrently-active communicating processes. This view of the system allows the verification of properties of the model which are expressible as inter-dependencies between events. Richer analyses are facilitated by decorating the events from the model in a variety of ways. One type of decoration is an estimate of the duration of events; another is a record of the security classification of values. In their turn, these types of decoration lead us to *stochastic process algebras* and *calculi for security analysis*.

Different analyses are applicable to languages of different types. The analyses can be so different in their nature and their use that they deserve the separate labels and classifications of *static analysis* and *dynamic analysis*. We wish to apply both static analysis and dynamic analysis to a protocol and so we should ask which analysis should be done first. Static analysis is computationally cheaper than dynamic analysis so static analysis should be completed first. If the results from the static analysis process are encouraging then one would follow with dynamic analysis. To begin with dynamic analysis would be a less prudent course of action because the computationally more expensive results of the dynamic analysis of a model would have to be discarded if a subsequent static analysis found the model to be unsatisfactory.

3.1 Extracting LySa and PEPA models from UML diagrams

To drive the static and dynamic analyses of the protocol under study, we start with a root description of the protocol expressed as a UML project and from this we draw out a LySa model (which is used in the static, security analysis) and a PEPA model (which is used in the dynamic, performance analysis).

The UML project which is the root of the federated analysis of security and performance which we perform contains a number of diagrams of different types. Among them are included class diagrams, sequence diagrams, state diagrams and collaboration diagrams. In our case these contain additional annotations from which the formal objects of the LySa and PEPA models can be extracted. In the DEGAS project we have implemented software components called *extractors* which automate this process. The extraction process is discussed in prior publications on the ForLySa extractor [10] and the PEPA extractor [13] so we do not discuss it further here.

3.2 Relationship between the PEPA and LySa models

We learned in the previous section that the process calculus models which we use have a common root (they come from the same high-level model expressed as a UML project). We now discuss the relationship between these derived models in formal language terms when they are viewed as processes which generate labelled transition systems.

The LySa model represents an interaction with an unbounded number of copies of the principals at work. In contrast, the PEPA model represents an interaction with a bounded number of copies of the principals at work. The components are fixed in number and processes are not dynamically spawned in PEPA.

The reason for the difference between the two models is due to the different types of analysis which are to be performed, as follows.

- When analyzing for security, we want to be certain that we do not exclude attacks which are possible when many runs of the protocol are concurrently active. Hence, we model the protocol using process replication in LySa and the LySatool computes its best approximation to this using parameterised replication instantiated at a sufficiently large bound.
- When analyzing for performance, we can compute the measures of interest as long as we have at least one of the principals of each type in the model. Additional copies of the components would incur greater state-space generation and model solution costs without changing the measures which can be computed. Therefore, in the PEPA model, we work with the least number of principals necessary.

A consequence of this difference is that the strong relationships which might naively be expected to hold between the two models (such as bisimulation equivalence) do *not* hold. However, the weaker relationship of *trace subsumption* does hold: every observable sequence of transitions from the PEPA model is an observable trace of the LySa model (but not the other way around). Further, if additional copies of the principals are added to the PEPA model then the traces generated by the PEPA model continue to be subsumed by those of the LySa model. Thus, the observations of the LySa model represent the behaviour of the PEPA model in the limit, as more and more principals participate in the interaction described by the protocol.

4 Case study: secure m-commerce

We consider a small case study of a buyer and a seller communicating securely using the "Wide-Mouthed Frog" (WMF) protocol due to Michael Burrows and studied by Burrows, Abadi and Needham in their paper on the BAN logic for authentication analysis [12].

Before the protocol commences the buyer makes an initial contact with the seller to set up the connection. After the protocol finishes there is a final handshake which closes the connection. Both the initial contact and the final handshake can be seen by an attacker, and this is the basis of the timing attack.

The design of such a system might seem to be fundamentally flawed. Why is it not the case that all of the communication between the buyer and the seller is protected by a secure protocol? The reason for this is that the application which we consider is a secure m-commerce (mobile e-commerce) application where buyers and sellers spontaneously meet and establish a connection. In consequence, the business topology is not static and so virtual introductions are necessary. Small devices such as PDAs and handhelds are the essential execution platform for a mobile computing application but devices such as these are limited in their computing and storage capabilities and cannot support many simultaneous connections. This makes virtual goodbyes a necessary part of our application software model.

Our concern is whether or not an attacker can draw success/failure inferences from a timing analysis of an implementation of the secure m-commerce application, under the assumption that they can time a significantly large number of interactions between buyers and sellers, through playing the role of a buyer and compiling tables of timings of success and failure outcomes.

4.1 Static analysis

From the point of view of authentication analysis the salient points of the description of the WMF protocol used in a mobile e-commerce setting are that it aims to establish a secure session under a secret session key K_{AB} between a buyer A and a seller B, who are the two principals in the dialogue. A trusted server is used to establish the secure session and A has registered a master key with the server, K_{AS}. Similarly, B has registered K_{BS} with the server.

The protocol has three steps, which we describe informally first.

- Principal A sends a message to the server including the name of B and the new session key K_{AB}, encrypted under K_{AS}.

- The server decrypts this and sends the name of A and the new key K_{AB} to B, encrypted under K_{BS}.
- Principal A sends a *message* to B encrypted under K_{AB}.

In LySa this protocol can be modelled as three parallel processes representing each of the principals A, B, and S. In Figure 1, we model the source and destination addresses of each message as the first two values of the message. Notice that these addresses are sent in the clear, thus allowing an attacker to modify or fake addressing information at will.

Symmetric key encryption in LySa is written as $\{B, K_{AB}\}_{K_{AS}}$ denoting that the name of principal B and the session key is encrypted under the key which principal A shares with the server. The corresponding decryption of this message takes place at the server in the process P_S in a separate language construct. It checks whether the correct key has been used for encryption and additionally uses matching to ensure that the first value inside the encryption is B before binding the second value to the variable zk. The ν operator, familiar from the π-calculus, is used to generate new session keys and new message content. In addition to symmetric key encryption the LySatool also supports public/private key pair encryption, but that is not used in this protocol.

The LySa model of the Wide-Mouthed-Frog protocol has been annotated with information about the authentication properties that the protocol is intended to have. The static analysis conducted by the LySatool is able to tell whether these properties are satisfied when executing the protocol in parallel with a "standard" network attacker in the style of Dolev and Yao [14].

To specify these authentication properties we first introduce labels or *crypto-points*, such as $[\mathsf{at}\, a1\,]$ and $[\mathsf{at}\, s1\,]$, to denote the places where information is encrypted or decrypted. Second, at each encryption point we describe a set of destination crypto-points where this message is intended to be decrypted. In the case, such as above, where we know exactly the intended recipient of the message this set has only a single element. Third, at each decryption point we assert where we believe that this message was encrypted by giving a set of possible origin points.

The static analysis performed by the LySatool works by computing over-approximations to what a LySa process can do in *all executions* of the process executed in parallel with an arbitrary attacker. The analysis will compute information about which messages may be sent over the network and compute information about which values may become bound to the variables used in the model. The latter includes the variables which may occur within the attacker and thus the analysis will report whether a specific value will be kept out of reach of the attacker i.e. whether the value is confidential or not.

The LySa processes are additionally annotated with authentication properties specifying intended origin and destinations of messages. Violations of these properties will also be reported by the static analysis. Since the analysis computes over-approximations of the behaviour of a process it may also report too many errors. In [4] it is proven that, with respect to the formal semantics of LySa, the analysis *cannot report too few errors* and also illustrated that reporting too many errors does not pose a big problem in practice.

Technically, the static analysis is specified as a flow logic, which gives a logical description of how the behaviour of a process is represented in the analysis result. For example, the analysis result contains a relation κ which is used to keep track of the possible message tuples which may be transmitted over the network. To generate a formula that describes e.g. how an output of a LySa process affects the analysis result, we use a constraint generation function $\mathcal{G}$ of the following form.

$$\mathcal{G}(\langle a, b, c\rangle.P) = (a, b, c) \in \kappa \wedge \mathcal{G}(P)$$

This states that to have an analysis of the LySa process $\langle a, b, c\rangle.P$ it must hold that the tuple (a, b, c) is present in κ because it is possible that this tuple is sent in some run of the process. Furthermore, there must be an analysis of the process P, which will be executed after the output. A similar constraint generation takes place for the other language constructs of LySa.

The actual analysis result is computed by finding a relation κ (and other analysis components), which satisfies the generated formula. This is done by using standard tools for solving systems of logical constraints. To perform the analysis process the LySa model is encoded into a fragment of first order logic and the solution to the constraints can be found by the Succinct Solver [21]. This computation of the analysis result is guaranteed to terminate in a time bounded by a polynomial in the size of the LySa process. The full analysis of LySa may be found in [4] while a thorough introduction to flow logics may be found in [22].

The LySa model of the WMF protocol is presented in Figure 1. The

$$\begin{aligned}
P_A \equiv\ & !(\nu\, K_{AB})\, \langle A, S, A, \{B, K_{AB}\}_{K_{AS}}[\mathsf{at}\ a1\ \mathsf{dest}\ s1\,]\rangle. \\
& (\nu\, \mathit{message})\, \langle A, B, \{\mathit{message}\}_{K_{AB}}[\mathsf{at}\ a2\ \mathsf{dest}\ b2\,]\rangle.\mathsf{0} \\
P_B \equiv\ & !(S, B;\ x).\mathsf{decrypt}\ x\ \mathsf{as}\ \{A;\ xk\}_{K_{BS}}[\mathsf{at}\ b1\ \mathsf{orig}\ s2\,]\ \mathsf{in} \\
& (A, B;\ y).\mathsf{decrypt}\ y\ \mathsf{as}\ \{;\ ym\}_{xk}[\mathsf{at}\ b2\ \mathsf{orig}\ a2\,]\ \mathsf{in}\ \mathsf{0} \\
P_S \equiv\ & !(A, S, A;\ z).\mathsf{decrypt}\ z\ \mathsf{as}\ \{B;\ zk\}_{K_{AS}}[\mathsf{at}\ s1\ \mathsf{orig}\ a1\,]\ \mathsf{in} \\
& \langle S, B, \{A, zk\}_{K_{BS}}[\mathsf{at}\ s2\ \mathsf{dest}\ b1\,]\rangle.\mathsf{0} \\
\mathit{WMF}_{\mathit{LySa}} \equiv\ & P_A \mid P_B \mid P_S
\end{aligned}$$

Fig. 1. LySa model of the Wide-Mouthed Frog protocol

LySatool confirms that the message sent from A to B in the WMF protocol is confidential as is the session key K_{AB}, since neither can come into the possession of an attacker in any run of the protocol. Furthermore, the authentication properties do hold for this version of the protocol though small variants of the model are vulnerable to parallel session attacks. More information on these LySa encodings of the WMF protocol and its analysis can be found in [4].

4.2 Dynamic analysis

Figure 2 shows a PEPA model of the Wide-Mouthed Frog protocol. This records the causality information of the protocol (by requiring A to communicate with the server first, and subsequent dependencies) and quantifies the average cost in time of each of the exchanges through the use of a random variable to use as the parameter of a negative exponential distribution.

$$P_A \stackrel{def}{=} (as, r_{as}).(ab, r_{ab}).P_A$$

$$P_B \stackrel{def}{=} (sb, \top).(ab, \top).P_B$$

$$P_S \stackrel{def}{=} (as, \top).(sb, r_{sb}).P_S$$

$$WMF_{PEPA} \stackrel{def}{=} P_A \underset{\{as, ab\}}{\bowtie} (P_S \underset{\{sb\}}{\bowtie} P_B)$$

Fig. 2. PEPA model of the Wide-Mouthed Frog protocol

Through the use of the IPC/DNAmaca tool chain we are able to analyse a PEPA model such as that in Figure 2 via the calculation of steady-state, transient and passage-time distributions. The DNAmaca solver provides direct and iterative solution methods, Krylov subspace techniques, and decomposition-based methods for both Markovian and semi-Markov processes. All PEPA models generate Continuous-Time Markov Chains (CTMCs) so we are only using the Markovian analysis capabilities provided by DNAmaca. The version of DNAmaca which extracts passage-times from purely Markov systems (release version HYDRA) uses uniformization to calculate both passage-time densities and cumulative distributions.

The advantages of such an exact solution method over the approximation offered by discrete-event simulation include the greater efficiency of the solution and the fact that the results do not need to be interpreted against difficult-to-compute confidence intervals which estimate a bound on the errors in simulation results. The methods implemented by the DNAmaca tool have been shown to scale to very complex models which generate state-spaces in excess of 15 million states [7].

Here we are using the IPC/DNAmaca tool chain to compute the difference between the passage-time distributions for successful and failing runs of the protocol. Our measure of a successful obfuscation of the difference between a success run and a failure run will be if the relative difference between their observed completion rates is less than a chosen small tolerance. The observed completion rates are plotted as a cumulative density function (CDF) showing how the probability of completion increases as time progresses. The CDF of the model before and after padding with delays are computed using IPC/DNAmaca.

Figure 3 shows the determination of a suitable value for the delay to be inserted into the execution of the protocol. The smooth bezier curve in the figure illustrates the protocol in a success run acting as the reference point for our experimentation. We wish to move the other plots as close to this reference curve as is required by varying the amount of delay which is inserted into the protocol.

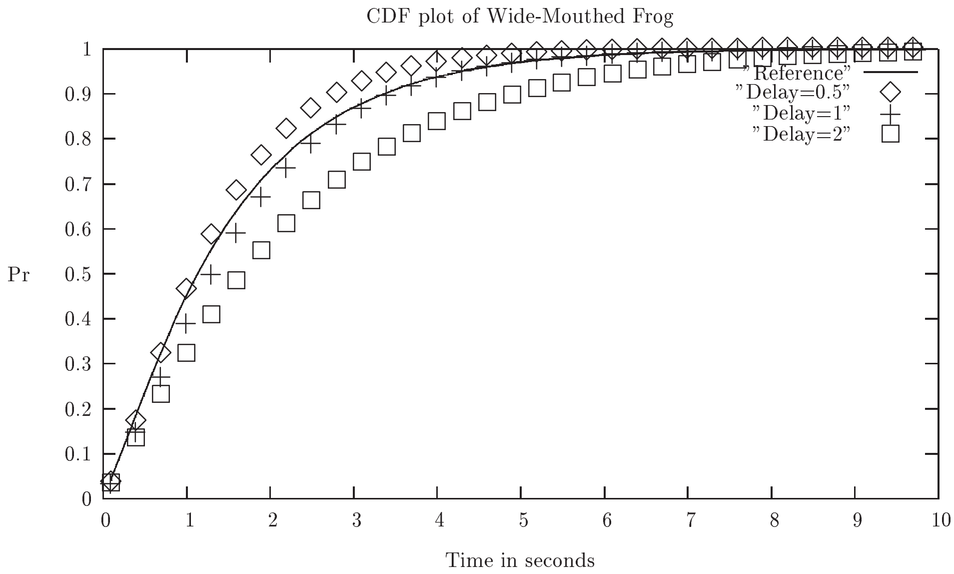


Fig. 3. CDF plot of passage time from start to completion of WMF protocol

Three estimates are taken for the value of the delay to be inserted into the failure scenario, 0.5, 1.0 and 2.0 seconds (plotted as diamond, plus and box symbols respectively). A delay of 0.5 seconds is insufficient and means that the faster path remains faster than the slower one. A delay of 2.0 seconds overcompensates and errs in the other direction, making the modification of the faster path slower than the slow one, and still perhaps detectably different. Inserting a delay of 1.0 seconds would seem to be a reasonable compensation. Further tuning might identify a better value somewhere between 0.5 and 1.0.

Using the Imperial PEPA Compiler and DNAmaca together with the method of *stochastic probes* [3] to select different start and end points for passages it is possible to automate the process of repeated passage-time analysis. The objective of repeated analyses is to rule out attacks which are based on correlating the timing of one path through the system with the timing of another, and making an indirect inference from that. Demonstrating that these passage-time distributions are acceptably close for all observable paths through the system which do not involve user interaction or another source of random stochastic delay is sufficient to show that a timing attack based on aggregating the results of timed observations is infeasible.

By selecting a value for the maximum tolerated difference between the distributions for the original and the version of the protocol which is padded with delays it is possible to increase the difficulty of a timing attack to the point where the users of the networked application which uses this protocol are satisfied that a timing attack is sufficiently difficult to mount that it would be uneconomic for an attacker to do so.

5 Related work

The problem of timing information leakage in security protocols has previously been studied by Focardi *et. al.* [16], and in the context of cryptographic protocols in [17]. In this work the authors use an real-time extension of Security Process Algebra [15]. A discrete time domain is used and processes are assumed to be synchronous. In [16] the authors develop a timed version of the property of bisimulation-based nondeducibility on compositions and show that this is useful to check for the absence of covert information flows. In [17] this work is extended to cryptographic protocol analysis.

One approach which has some similarities to ours and has similar aims of reconciling security and timing behaviour has been taken in the work of Volpano and Smith on probabilistic noninterference for concurrent languages [23]. That work uses Markov chains to study timing leakages between concurrent threads arising from the probabilistic behaviour of the scheduler.

Papers by Agat and Sands [1, 2] have considered introducing dummy steps into an implementation in order to ensure that the fast path is no faster than the slow path. They construct type and transformation systems which transform a sequential imperative program P into a secure version P' which is semantically equivalent in a time-ignorant semantics and which would not allow leaks with respect to a time-observant semantics (expressed by a simulation result). Our work differs from theirs in that we consider protocols expressed at a high-level of design using annotated process algebras in contrast to the imperative language programs of Agat and Sands. Thus our work has the potential to apply to a range of satisfactory implementations of the protocol used instead of only a single instance.

6 Conclusions

In the innovative and experimental design of novel communications protocols it is necessary to consider both the security and performance of the new design, where the objective of the performance analysis may be to supplement the strong security provided by encryption and authentication.

Formal languages are helpful here because they provide an unambiguous statement of the protocol and because they are amenable to automated analyses. These analyses depend also on an unambiguous expression of the property which is being checked. In such a setting both the security and performance analyses can be re-run after a change to the protocol in order to ensure that the change has had no detrimental effect. We have applied this methodology here in the analysis of a high-level model of secure communications between a buyer and a seller.

Security and performance concerns are often at variance. It is frequently the case that application developers who are concerned with achieving peak performance view the security measures built into layers of application software as an overhead on execution time. However, in the networked world of today security is not an optional extra and so performance analysis and security analysis must work harmoniously together. In the example considered in the present paper we have shown how performance analysis can supplement a static analysis of system security, and how a control-flow analysis can add value to a stochastic model of system behaviour. Thus each analysis reinforces the usefulness of the other providing much-needed assistance in the design of reliable systems which give guarantees of security and performance properties.

Acknowledgements

The authors are supported by the Design Environments for Global ApplicationS project (DEGAS) IST-2001-32072 funded by the Future and Emerging Technologies Proactive Initiative on Global Computing. The authors thank the anonymous reviewers of the PASM workshop for their helpful comments. Monika Maidl provided detailed comments on an earlier version of this paper which helped us to improve the exposition throughout. The Imperial PEPA Compiler was developed by Jeremy Bradley and is available from http://www.doc.ic.ac.uk/ipc. The LySatool and the LySa extractor are available from http://www.imm.dtu.dk/cs_LySa/. The PEPA extractor is a component of the PEPA Workbench, from http://www.dcs.ed.ac.uk/pepa.

References

[1] Agat, J., *Transforming out timing leaks*, in: *Proceedings of the 27th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Boston, Massachusetts, 2000.

[2] Agat, J. and D. Sands, *On confidentiality and algorithms*, in: *Proceedings of the IEEE Symposium on Security and Privacy* (2001), pp. 64–77.

[3] Argent-Katwala, A., J. Bradley and N. Dingle, *Expressing performance requirements using regular expressions to specify stochastic probes over process algebra models*, in: *Proceedings of the Fourth International Workshop on Software and Performance* (2004), pp. 49–58.

[4] Bodei, C., M. Buchholtz, P. Degano, F. Nielson and H. Nielson, *Automatic validation of protocol narration*, in: *Proceedings of the 16th IEEE Computer Security Foundations Workshop (CSFW'03)* (2003), pp. 1–15.

[5] Bradley, J., N. Dingle, S. Gilmore and W. Knottenbelt, *Derivation of passage-time densities in PEPA models using IPC: The Imperial PEPA Compiler*, in: G. Kotsis, editor, *Proceedings of the 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems* (2003), pp. 344–351.

[6] Bradley, J., N. Dingle, S. Gilmore and W. Knottenbelt, *Extracting passage times from PEPA models with the HYDRA tool: A case study*, in: S. Jarvis, editor, *Proceedings of the Nineteenth annual UK Performance Engineering Workshop*, University of Warwick, 2003, pp. 79–90.

[7] Bradley, J., N. Dingle, P. Harrison and W. Knottenbelt, *Distributed computation of passage time quantiles and transient state distributions in large semi-Markov models*, in: *Proceedings of International Workshop on Performance Modeling, Evaluation, and Optimization of Parallel and Distributed Systems (PMEO-PDS 2003), Nice, France, April 26 2003*, 2003, pp. 281–289.
URL http://aesop.doc.ic.ac.uk/pubs/passage-pmeo2003/

[8] Brumley, D. and D. Boneh, *Remote timing attacks are practical*, in: *Conference proceedings of the 12th USENIX Security Symposium* (2003), pp. 1–14.

[9] Buchholtz, M., *LySa — a process calculus*, Web site hosted by Informatics and Mathematical Modelling at the Technical University of Denmark (2004), http://www.imm.dtu.dk/cs_LySa/.

[10] Buchholtz, M., C. Montangero, L. Perrone and S. Semprini, *For-LySa: UML for authentication analysis*, 2004, to appear.
URL http://www.imm.dtu.dk/pubdb/p.php?3235

[11] Buchholtz, M., H. Nielson and F. Nielson, *A calculus for control flow analysis of security protocols*, International Journal of Information Security (2004), available online at http://dx.doi.org/10.1007/s10207-004-0036-x.

[12] Burrows, M., M. Abadi and R. Needham, *A logic of authentication*, ACM Transactions on Computing Systems **8** (1990), pp. 18–36.

[13] Canevet, C., S. Gilmore, J. Hillston, M. Prowse and P. Stevens, *Performance modelling with UML and stochastic process algebras*, IEE Proceedings: Computers and Digital Techniques **150** (2003), pp. 107–120.

[14] Dolev, D. and A. Yao, *On the security of public key protocols*, in: *22nd Annual Symposium on Foundations of Computer Science* (1981), pp. 350–357.

[15] Focardi, R. and R. Gorrieri, *Classification of security properties. Part I: Information flow*, in: *Foundations of Security Analysis and Design*, LNCS **2171** (2001), pp. 331–396.

[16] Focardi, R., R. Gorrieri and F. Martinelli, *Real-time information flow analysis*, IEEE Journal on Selected Areas in Communications **21** (2003), pp. 20–35.

[17] Gorrieri, R. and F. Martinelli, *A simple framework for real-time cryptographic protocol analysis with compositional proof rules*, Science of Computer Programming **50** (2004), pp. 23–49.

[18] Hillston, J., "A Compositional Approach to Performance Modelling," Cambridge University Press, 1996.

[19] Knottenbelt, W., "Generalised Markovian analysis of timed transition systems," Master's thesis, University of Cape Town (1996).

[20] Kocher, P. C., *Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems*, in: N. Koblitz, editor, *Advances in Cryptology* (1996), pp. 104–113.

[21] Nielson, F., H. R. Nielson and H. Seidl, *A succinct solver for ALFP*, Nordic Journal of Computing **9** (2002), pp. 335–372.

[22] Nielson, H. R. and F. Nielson, *Flow logic: a multi-paradigmatic approach to static analysis*, in: *The Essence of Computation: Complexity, Analysis, Transformation*, LNCS **2566**, Springer, 2002 pp. 223–244.

[23] Volpano, D. and G. Smith, *Probabilistic noninterference in a concurrent language*, in: *Proceedings of the 11th IEEE Computer Security Foundations Workshop*, 1998, pp. 34–43.

A LySa

In this appendix we give an informal introduction to the LySa calculus. We focus on the features of the calculus which were useful in the present study. This appendix is not a definitive reference for LySa. The reader is referred to [11] for a more comprehensive introduction to the calculus and its capabilities.

Briefly, LySa is a variant of the Spi-calculus which includes pattern matching on values at input and decryption. LySa identifies the syntactic categories of *terms* and *processes*. Terms include *variables*; *names* used to represent keys, nonces, and uninterpreted data; and *encrypted values* $\{t_1, \ldots, t_k\}_{t_0}$ representing perfect symmetric key cryptography where a tuple of k terms are encrypted under a key t_0.

Process in LySa include the following syntactic forms:

Parallel: $P_1 \mid P_2$ is the parallel composition operator which evaluates its left and right operand in parallel, allowing them to match inputs and outputs.

Replication: $!P$ replicates as many copies of P as necessary and in this way provides a mechanism to generate long runs from terminating processes.

Terminated process: $\mathsf{0}$ is the terminated process, used to end a process term.

Restriction/name generation $(\nu\, n)\, P$ is the construct which introduces a new name n, to be used in the process term P. New names are needed for session keys and message content in protocol descriptions.

Output $\langle t_1, \ldots, t_k \rangle.P$ outputs the tuple $t_1, \ldots, t_k$ and continues as P.

Input $(t_1, \ldots, t_j;\, x_{j+1}, \ldots x_k).P$ describes a pattern match input. Input will synchronise with an output of length k but only if the first j values of the output are equal to the first j values of the input. In this case, each of the remaining $k - j$ values of the output will be bound component-wise to the variables placed syntactically after the semi-colon in the input. These values will be used for the variables in the continuation P.

Decryption $\mathsf{decrypt}\ t\ \mathsf{as}\ \{t_1, \ldots, t_j;\, x_{j+1}, \ldots, x_k\}_{t_0}\ \mathsf{in}\, P$ attempts to decrypt and pattern match the term t. Decryption succeeds when t is a tuple of length k encrypted under the key t_0. In that case, pattern matching takes place on the first j values and possibly results in binding of the last $k - j$ values, similarly to input.

B PEPA

This appendix provides a brief introduction to PEPA in order to make the paper self-contained. It can safely be skipped by anyone who already knows the PEPA language. For a full explanation which complements the brief description presented here the reader is referred to [18].

Prefix: The basic mechanism for describing the behaviour of a system with a PEPA model is to give a component a designated first action using the prefix combinator, denoted by a full stop. For example, $(\alpha, r).S$ carries out activity (α, r), which has action type α and an exponentially distributed duration with parameter r, and it subsequently behaves as S.

Choice: The component $P + Q$ represents a system which may behave either as P or as Q. The activities of both P and Q are enabled. The first activity to complete distinguishes one of them: the other is discarded. The system will behave as the derivative resulting from the evolution of the chosen component.

Constant: It is convenient to be able to assign names to patterns of behaviour associated with components. Constants are components whose meaning is given by a defining equation. The notation for this is $X \stackrel{\mathit{def}}{=} E$. The name X is in scope in the expression on the right hand side meaning that, for example, $X \stackrel{\mathit{def}}{=} (\alpha, r).X$ performs α at rate r forever.

Hiding: The possibility to abstract away some aspects of a component's behaviour is provided by the hiding operator, denoted P/L. Here, the set L identifies those activities which are to be considered internal or private to the component and which will appear as the unknown type τ.

Cooperation: We write $P \underset{L}{\bowtie} Q$ to denote cooperation between P and Q over L. The set which is used as the subscript to the cooperation symbol, the *cooperation set* L, determines those activities on which the *cooperands* are forced to synchronise. For action types not in L, the components proceed independently and concurrently with their enabled activities. We write $P \parallel Q$ as an abbreviation for $P \underset{L}{\bowtie} Q$ when L is empty.

However, if a component enables an activity whose action type is in the cooperation set it will not be able to proceed with that activity until the other component also enables an activity of that type. The two components then proceed together to complete the *shared activity*. The rate of the shared activity may be altered to reflect the work carried out by both components to complete the activity (for details see [18]).

In some cases, when an activity is known to be carried out in cooperation with another component, a component may be *passive* with respect to that activity. This means that the rate of the activity is left unspecified (denoted $\top$) and is determined upon cooperation, by the rate of the activity in the other component. All passive actions must be synchronised in the final model.

Performability of a secure electronic voting algorithm

Nigel Thomas [1]

School of Computing Science, University of Newcastle upon Tyne, UK

Abstract

This paper considers modelling the performance and reliability of a secure electronic voting scheme. The scheme provides secure verifiable blind voting, however there is a considerable administration overhead to this level of security. A Markovian process algebra is used to build a performance model of a basic system of n distributed voters. This model is shown to suffer from the familiar state space explosion problem. A simpler model is therefore developed to allow larger and more practically relevant systems to be studied. The original model is then extended to include the possibility that voters may fail and two modes of recovery are considered. The models are evaluated numerically using data obtained from measuring an implementation of this scheme in order to determine the accuracy of the approximations.

Keywords: secure voting scheme, performance modelling, PEPA

1 Introduction

Making electronic voting secure has been an active topic of research for more than twenty years and many secure electronic voting schemes have been introduced since the inception of anonymous channels to separate voters and votes by Chaum [2]. Voting scenarios occur in many situations to perform many tasks in distributed systems, as well as facilitating voting amongst human activities. The most publicly visible form of secure voting is the use of online systems for voting in political elections which has been introduced in several countries. This form of voting has several obvious requirements.

- Only registered voters are allowed to vote.
- Voters only vote once.
- It should not be possible to find out who voted, or how they voted.

[1] Email: `nigel.thomas@ncl.ac.uk`

This is the workshop version of the paper. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: `www.elsevier.nl/locate/entcs`

These factors mean that any voting scheme for use in this scenario has to provide adequate authentication, vote management and so-called *blinding* mechanisms, whilst operating over a potentially insecure communication medium. Fujioka et al [6], formalised these requirements as *completeness* (all votes counted correctly), *soundness* (a dishonest voter cannot disrupt the election), *privacy* (of votes), *unreuseability* (cannot vote twice), *eligibility* (to vote), *fairness* (of the vote), *verifiability* (of the result). In addition, Iversen [9] introduced the requirement of *receipt freeness*; many protocols issue receipts or tokens of some form to prove to the voter that the system behaved as it should. However, these receipts might be used by a dishonest voter to prove that they voted in a certain way, thus facilitating vote selling.

Many secure voting schemes rely in some way on encrypting data and even with fast processors encryption and decryption adds an overhead to data processing. However, the major overheads arise because of the additional communication that is required in order to ensure that the requirements of the secure vote are met. Secure voting schemes will generally use some form of anonymous channel, digital pseudonyms, blind signatures, trusted authorities and multiple key ciphers to separate the voter, the authority to vote, the vote itself and the counting of the vote. Clearly there is a substantial overhead in providing these measures and therefore the performance of such a system is of obvious practical interest.

2 A secure electronic voting algorithm

This paper considers a secure electronic voting scheme proposed by Fujoika et al [6] which has been implemented in at least two systems, SENSUS [5] and EVOX [7]. The scheme consists of an arbitrary number of voters, an administrator to issue authority to vote, and a counter to collect votes and to determine the result. An anonymous channel is used to communicate the vote between the voter and the collector / counter. The scheme is described thus.

- Preparation: Voter i
  - (i) Choose the voting strategy.
  - (ii) Commit to the strategy using a bit commitment scheme c_i.
  - (iii) Blind the committed ballot, b_i.
  - (iv) Sign the blinded ballot sv_i.
  - (v) Send to the administrator the signed blinded ballot, the blinded ballot and unique voter ID, ID_i.
- Administration: Administrator
  - (i) Receive message from voter i.
  - (ii) Check right to vote for voter i.
  - (iii) Check voter i has not voted already.
  - (iv) Verify the signature; if valid sign the blinded ballot, sa_i.
  - (v) Send sa_i to voter i.
  - (vi) When the administration period is over, publish a list containing every

$\{ID_j, b_j, sv_j\}$.

- Voting: Voter i
  (i) Unblind sa_i to give the ballot signed by the Administrator, ba_i.
  (ii) Check signature.
  (iii) Send $\{c_i, ba_i\}$ to the Counter through an anonymous channel.
- Collecting: Counter
  (i) Receive message from voter i.
  (ii) Check Administrators signature on ba_i; if valid add $\{N, c_i, ba_i\}$ to a list, where N is a unique reference number.
  (iii) When the collecting period is over, publish a list containing every $\{N, c_i, ba_i\}$.
- Opening: Voter i
  (i) Checks that the vote appears on the list published by the Counter; if not appeal.
  (ii) Send the bit commitment key k_i to the Counter through an anonymous channel.
- Counting: Counter
  (i) Use k_i to retrieve the voting strategy.
  (ii) Check the strategy is valid.
  (iii) When all votes are counted, publish the final result.

3 Simple PEPA model

A formal presentation of PEPA is given in [8], in this section a brief informal summary is presented. PEPA, being a Markovian Process Algebra, only supports actions that are negative exponentially distributed at given rates. Specifications written in PEPA represent Markov processes and can be mapped to a continuous time Markov chain (CTMC). Systems are specified in PEPA in terms of *activities* and *components*. An activity (α, r) is described by the *type* of the activity, α, and the *rate* of the associated negative exponential distribution, r. This rate may be any positive real number, or given as *unspecified* using the symbol $\top$. The syntax for describing components is given as:

$$P ::= (\alpha, r).P \mid P + Q \mid P/L \mid P \underset{L}{\bowtie} Q \mid A$$

The component $(\alpha, r).P$ performs the activity of type α at rate r and then behaves like P. The component $P + Q$ behaves either like P or like Q, the resultant behaviour being given by the first activity to complete. The component P/L behaves exactly like P except that the activities in the set L are concealed, their type is not visible and instead appears as the unknown type τ. Concurrent components can be synchronised, $P \underset{L}{\bowtie} Q$, such that activities in the *cooperation set* L involve the participation of both components. In PEPA the shared activity occurs at the slowest of the rates of the participants and if a rate is unspecified in a component, the component is passive with respect to the activities of that type. The parallel combinator $\|$ is used as shorthand to denote synchronisation with no shared activities, i.e. $P \| Q \equiv P \underset{\emptyset}{\bowtie} Q$. $A \stackrel{\mathit{def}}{=} P$ gives the constant A the behaviour of the component P.

The algorithm described above is modelled in PEPA as follows.

$$
\begin{aligned}
Voter0 &\stackrel{\mathit{def}}{=} (choose, c_1).Voter0_1 + (publishC, \top).Voter4 \\
Voter0_1 &\stackrel{\mathit{def}}{=} (bitcommit, b1).Voter0_2 + (publishC, \top).Voter4 \\
Voter0_2 &\stackrel{\mathit{def}}{=} (blind, b_2).Voter0_3 + (publishC, \top).Voter4 \\
Voter0_3 &\stackrel{\mathit{def}}{=} (blind, b_3).Voter0_4 + (publishC, \top).Voter4 \\
Voter0_4 &\stackrel{\mathit{def}}{=} (sign, s_1).Voter0_5 + (publishC, \top).Voter4 \\
Voter0_5 &\stackrel{\mathit{def}}{=} (sendA, s_2).(sendV, \top).Voter1 + (publishC, \top).Voter4 \\
Voter1 &\stackrel{\mathit{def}}{=} (unblind, u_1).Voter1_1 + (publishC, \top).Voter4 \\
Voter1_1 &\stackrel{\mathit{def}}{=} (unblind, u_2).Voter1_2 + (publishC, \top).Voter4 \\
Voter1_2 &\stackrel{\mathit{def}}{=} (verify, v_2).Voter1_3 + (publishC, \top).Voter4 \\
Voter1_3 &\stackrel{\mathit{def}}{=} (verify, v_3).Voter1_4 + (publishC, \top).Voter4 \\
Voter1_4 &\stackrel{\mathit{def}}{=} (sendC, s_6).(publishC, \top).Voter2 \\
&\quad +(publishC, \top).Voter4 \\
Voter2 &\stackrel{\mathit{def}}{=} (check, p * c_4).Voter3 \\
&\quad +(check, (1-p) * c_4).(sendCo, s_7).Voter4 \\
Voter3 &\stackrel{\mathit{def}}{=} (appeal, a_1).(sendCo, s_7).Voter4 \\
Voter4 &\stackrel{\mathit{def}}{=} (new, \top).Voter0
\end{aligned}
$$

$$
\begin{aligned}
Administrator &\stackrel{\mathit{def}}{=} (sendA, \top).(check, c_2).(check2, c_3).(verify, v_1). \\
&\quad (sign, s_3).(sign, s_4).(sendV, s_5).Administrator \\
&\quad +(publishA, p_1).(new, \top).Administrator
\end{aligned}
$$

$$
\begin{aligned}
Collector_i &\stackrel{\mathit{def}}{=} (sendC, \top).(verify, v_4).(verify, v_5).(add, a_2). \\
&\quad Collector_{i+1} + (publishA, \top).(publishC, p_2).Counter_i \\
&\quad ,\ 0 \leq i < N \\
Collector_N &\stackrel{\mathit{def}}{=} (publishA, \top).(publishC, p_2).Counter_N \\
Counter_i &\stackrel{\mathit{def}}{=} (sendCo, \top).(check, c_5).Counter_{i-1}\ ,\ 0 < i \leq N \\
Counter_0 &\stackrel{\mathit{def}}{=} (finalPublish, f).(new, fast).Collector0
\end{aligned}
$$

$$
Voters \stackrel{\mathit{def}}{=} Voter0 \underset{\{publishC, new\}}{\bowtie} \cdots \underset{\{publishC, new\}}{\bowtie} Voter0
$$

$$
\left(Collector_0 \underset{\{publishA, new\}}{\bowtie} Administrator \right) \underset{\left\{ \substack{sendA, sendV, sendC, \\ sendCo, publishC, new} \right\}}{\bowtie} Voters
$$

In essence this is a straightforward client/server process with two servers and N clients. Each $Voter$ component represents a single client. There are continuously repeated elections. In each election a voter may vote or not, thus the collector may receive a number of votes between 0 and N inclusive. The election period is terminated by the *Administrator* deciding to publish the voter list. This in turn causes the *Collector* to publish the list of votes. A *Voter* will then confirm their vote (or appeal first if their vote was not on the list and they voted) with the *Counter*. The *Counter* is forced to wait until every voter who voted has confirmed their vote, before publishing the result and then calling a new election.

If a voter has not voted it may still have begun the process. Thus in order to avoid the *Collector* from being blocked by a *Voter* the *publishC* action is enabled in all of the derivatives of the *Voter* to force it to abandon voting when the counting phase begins. Note that *publishC* is not enabled in the derivative $Voter0_5.sendA$ as once the *sendA* action has been performed, the *Administrator* will process the request before publishing the voter list.

3.1 Model analysis and simplification

Evaluating this model using the PEPA Workbench [4] the number of states for a two voter system is found to be 561, for 3 voters 8583 and for 4 voters 130245 states. Four voters was the largest system it was possible to analyse directly using the PEPA Workbench before running out of memory. The state space can be dramatically reduced by combining activities internal to the components. In this way it is possible to consider a model of 7 voters using the PEPA Workbench (602394 states). However, it is clear that this is much fewer voters than any practical situation would demand. Hence for larger values of N it is necessary to take a different modelling approach. It is also important to note that aggregating sequences of actions changes the behaviour of the model with respect to higher moments. A number of negative exponentially distributed actions in series will generally have a lower variance than a single negative exponentially distributed action with the same overall mean. However, since the Markovian assumption in this model is in itself an approximation there is little that can be accurately said about metrics based on higher moments even in the most detailed model.

Consider the following model specification.

$$\begin{aligned}
Voters0_i &\stackrel{def}{=} (sendA, i * v_1).Voters0_{i-1} + (publishA, \top).Voters0_0 \\
&\quad ,\ 0 < i \leq N \\
Voters0_0 &\stackrel{def}{=} (new, \top).Voters0_N + (publishA, \top).Voters0_0
\end{aligned}$$

$$\begin{aligned}
Administrator &\stackrel{def}{=} (sendA, \top).(admin, a).(sendV, s_5).Administrator \\
&\quad +(publishA, p_1).Administrator
\end{aligned}$$

$$\begin{aligned}
Voters1_0 &\stackrel{def}{=} (sendV, \top).Voters1_1 + (publishC, \top).Voters4 \\
Voters1_i &\stackrel{def}{=} (sendV, \top).Voters1_{i+1} + (publishC, \top).Voters4 \\
&\quad +(sendC, i * v_2).Voters1_{i-1} \quad 0 < i < N
\end{aligned}$$

$$Voters1_N \stackrel{def}{=} (publishC, \top).Voters4 + (sendC, N * v_2).Voters1_{N-1}$$
$$Voters4 \stackrel{def}{=} (new, \top).Voters1_0$$

$$Collector_i \stackrel{def}{=} (sendC, \top).(collect, c).Collector_{i+1}$$
$$+(publishA, \top).(publishC, p_2).Counter_i \ \ , \ 0 \leq i < N$$
$$Collector_N \stackrel{def}{=} (publishA, \top).(publishC, p_2).Counter_N$$
$$Counter_i \stackrel{def}{=} (sendCo, co).Counter_{i-1} \ \ , \ 0 < i \leq N$$
$$Counter_0 \stackrel{def}{=} (finalPublish, f).(new, fast).Collector0$$

$$\left(Collector_0 \underset{\{publishA\}}{\bowtie} Administrator\right) \underset{\left\{\substack{sendA, sendV, sendC,\\ publishC, publishA, new}\right\}}{\bowtie}$$
$$\left(Voters0_N \underset{\{new\}}{\bowtie} Voters1_0\right)$$

This model is essentially a modified closed queueing network, where the nodes are the waiting voters, the administrator, the voters who have cast their vote, the collector and the counter.

Figure 1 shows the growth of the state space and the number of transitions with the number of voters. Plots are shown for the *basic model*, the *aggregated model* (where internal actions have been aggregated) and the *queue-based model*.

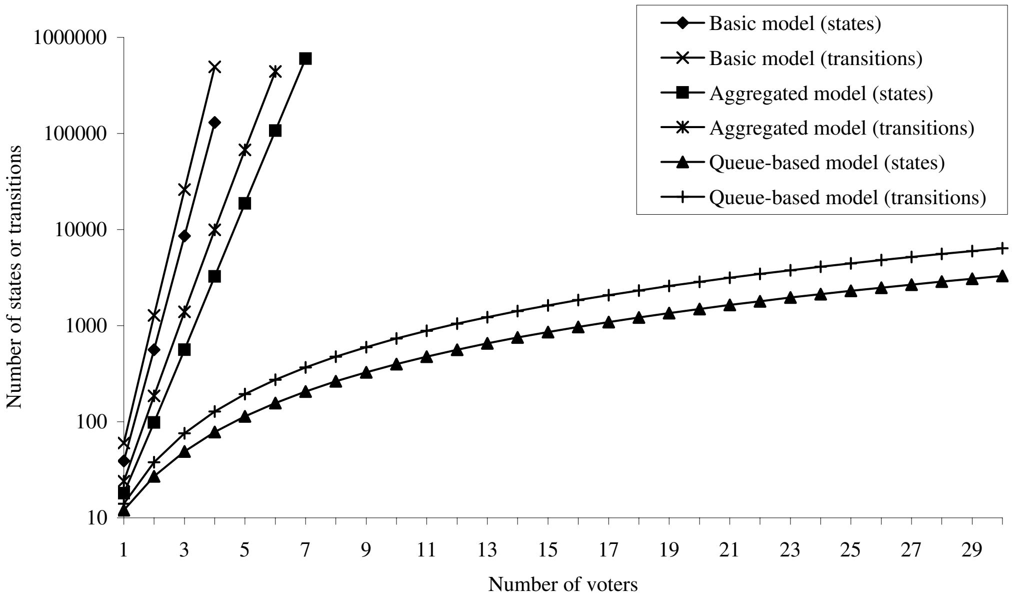


Fig. 1. the size of the underlying state space and number of transitions varied with the number of voters

From this plot it is clear that the basic and aggregated models have poor scalability and are only suitable for checking the behaviour of the system with very few voters. As such they can only reliably be used to check for errors in the scheme and do not give much in the way of practical performance information. For the two voter case the queue-based model has just 27 states, a ten voter model has just 399 states. The price paid for that is that a lot of information has been lost. For the most part this means that multiple sequential activities have been replaced by single activities with modified rates, potentially introducing greater error for measures based on higher moments. However, the voter behaviour has also been significantly altered. All the voters are now contained in two components, which are queues of voters who are waiting to vote or who have voted. There is now no discrimination as to what the status of an individual voter is. Instead it is only known the number of voters that have voted. In addition the appeal process has been removed, which greatly simplifies the behaviour. The delay for checking and possibly appealing is now incorporated entirely within the counter. This level of information is sufficient for performance analysis, however it would be insufficient for verifying the protocol. The combination of the basic model and the queue-based version allow both forms of analysis for potentially large values of N. However, it is important to note that the state space of the queue-based model grows slightly above linear with respect to the number of voters. Although the rate of state space increase is much less than the other two models, it is still such that a 1000 voter system will give rise to a state space of over three and a half million states.

4 Considering unreliable clients

In the previous models it is assumed that the clients and their messages are entirely reliable. It is of course possible that a client process may fail, causing a $Voter$ to be unable to vote, or confirm their vote. In general this failure may occur at any point in the exchange, thus causing the *Administrator*, *Collector* or *Counter* to expect a response when none is forthcoming.

Such a failure is modelled as a transition to a failed derivative in the $Voter$. In order to allow this $fail$ transition to occur in any derivative, all the derivatives become prefixed (note it is possible to achieve the same effect using other means, but it quickly becomes difficult to read). The action *SendV* is considered to be sent by the *Administrator* whether or not the $Voter$ can receive it. Thus, for modelling purposes, it is enabled in $VoterFail2$. Clearly this must happen before *publishC* if *sendA* has happened, likewise for the actions *new* and *publishC*. If a client is repaired after a new election has been called, the voter must wait until the next new election before being able to vote.

Since some voters will have voted and then subsequently failed, they may be unable to confirm their vote. In order that the *Counter* does not deadlock waiting for this confirmation, a *timeout* action is introduced. When the *timeout* occurs all remaining votes will be discarded. Clearly this may include some voters who are still active but have not yet confirmed their vote. Thus this action requires that voters are timely in confirming as well as voting. This timeout could be modelled as a simple negative exponential transition, however this obviously gives rise to a highly variable time to count the votes. In order to reduce the variance an Erlang distributed transition is formed from a series of negative exponential transitions in the form of a *Timer* component. This means of modelling timeouts in PEPA has been used previously by Bowman et al [1].

$$\begin{aligned}
Voter0 &\stackrel{def}{=} (choose, c_1).Voter0_1 + (fail, f).VoterFail \\
&\quad +(publishC, \top).Voter4 \\
Voter0_1 &\stackrel{def}{=} (bitcommit, b_1).Voter0_2 + (fail, f).VoterFail \\
&\quad +(publishC, \top).Voter4 \\
Voter0_2 &\stackrel{def}{=} (blind, b_2).Voter0_3 + (fail, f).VoterFail \\
&\quad +(publishC, \top).Voter4 \\
Voter0_3 &\stackrel{def}{=} (blind, b_3).Voter0_4 + (fail, f).VoterFail \\
&\quad +(publishC, \top).Voter4 \\
Voter0_4 &\stackrel{def}{=} (sign, s_1).Voter0_5 + (fail, f).VoterFail \\
&\quad +(publishC, \top).Voter4 \\
Voter0_5 &\stackrel{def}{=} (sendA, s_2).Voter0_6 + (fail, f).VoterFail \\
&\quad +(publishC, \top).Voter4 \\
Voter0_6 &\stackrel{def}{=} (sendV, \top).Voter1 + (fail, f).VoterFail2 \\
Voter1 &\stackrel{def}{=} (unblind, u_1).Voter1_1 + (fail, f).VoterFail \\
&\quad +(publishC, \top).Voter4 \\
Voter1_1 &\stackrel{def}{=} (unblind, u_2).Voter1_2 + (fail, f).VoterFail \\
&\quad +(publishC, \top).Voter4 \\
Voter1_2 &\stackrel{def}{=} (verify, v_2).Voter1_3 + (fail, f).VoterFail \\
&\quad +(publishC, \top).Voter4 \\
Voter1_3 &\stackrel{def}{=} (verify, v_3).Voter1_4 + (fail, f).VoterFail \\
&\quad +(publishC, \top).Voter4 \\
Voter1_4 &\stackrel{def}{=} (sendC, s_6).Voter1_5 + (fail, f).VoterFail \\
&\quad +(publishC, \top).Voter4 \\
Voter1_5 &\stackrel{def}{=} (publishC, \top).Voter2 + (fail, f).VoterFail \\
Voter2 &\stackrel{def}{=} (check, p * c_4).Voter3 + (check, (1-p) * c_4).Voter2_1 \\
&\quad +(fail, f).VoterFail1 \\
Voter2_1 &\stackrel{def}{=} (sendCo, s_7).Voter4 + (fail, f).VoterFail1 \\
Voter3 &\stackrel{def}{=} (appeal, a_1).Voter3_1 + (fail, f).VoterFail1 \\
Voter3_1 &\stackrel{def}{=} (sendCo, s_7).Voter4 + (fail, f).VoterFail1
\end{aligned}$$

$$\begin{aligned}
Voter4 &\stackrel{def}{=} (new,\top).Voter0 + (fail,f).VoterFail \\
VoterFail &\stackrel{def}{=} (publishC,\top).VoterFail1 \\
&\quad +(repair,r_1).(publishC,\top).Voter4 \\
VoterFail1 &\stackrel{def}{=} (new,\top).VoterFail + (repair,r_1).Voter4 \\
VoterFail2 &\stackrel{def}{=} (sendV,\top).VoterFail \\
&\quad +(repair,r_1).(sendV).(publishC,\top).Voter4
\end{aligned}$$

$$\begin{aligned}
Administrator &\stackrel{def}{=} (sendA,\top).(check,c_2).(check2,c_3).(verify,v_1). \\
&\quad (sign,s_3).(sign,s_4).(sendV,s_5).Administrator \\
&\quad +(publishA,p_1).(new,\top).Administrator
\end{aligned}$$

$$\begin{aligned}
Collector_i &\stackrel{def}{=} (sendC).(verify,v_4).(verify,v_5).(add,a_2).Collector_{i+1} \\
&\quad +(publishA,\top).(publishC,p_2).Counter_i \ \ , \quad 0 \leq i \leq N \\
Counter_i &\stackrel{def}{=} (sendCo,\top).(check,c_5).Counter_{i-1} \\
&\quad +(timeout,\top).Counter_0 \ \ , \ 0 < i \leq N \\
Counter_0 &\stackrel{def}{=} (finalPublish,f).(new,fast).Collector0
\end{aligned}$$

$$\begin{aligned}
Timer &\stackrel{def}{=} (PublishC,\top).(timestep,t_1).Timer_1 \\
&\quad +(new,\top).Timer \\
Timer_i &\stackrel{def}{=} (timestep,t_1).Timer_{i+1} + (new,\top).Timer \ , \ 0 < i < X \\
Timer_X &\stackrel{def}{=} (timeout,t_1).Timer + (new,\top).Timer
\end{aligned}$$

$$Voters \stackrel{def}{=} Voter0 \underset{\left\{\substack{publishC,\\ new}\right\}}{\bowtie} \cdots \underset{\left\{\substack{publishC,\\ new}\right\}}{\bowtie} Voter0$$

$$Timer \underset{\left\{\substack{new,timeout,\\ PublishC}\right\}}{\bowtie} \left(Collector_0 \underset{\left\{\substack{publishA,\\ new}\right\}}{\bowtie} Administrator\right) \underset{\left\{\substack{sendA,sendV,sendC,\\ sendCo,publishC,new}\right\}}{\bowtie} Voters$$

This model assumes that if a client fails then it will not be able to play any further part in the vote, even if the election was still taking place. Thus an adapted voter has been developed which has a slightly better recovery mechanism. In order to do this it is necessary to make certain assumptions about how the clients know what is happening in the election and the nature of the publishing of voters and votes. It is therefore assumed that on repair a client must re-initiate a connection. When the connection is re-initiated the voter process is informed of the current election, its status and any published lists. The voter can then determine the appropriate behaviour to undertake. It is assumed that this imposes a delay on the client (part of the repair delay), but does not affect the server. When modelling this scenario it is necessary to introduce additional failure states to include the information following repair which will be available to the voter. This may appear to be rather back-to-front, but it is the simplest way in which the correct behaviour can be recovered on repair in this form of model. All the agents up to $Voter1_5$ are as above, the remainder are redefined as follows.

$$
\begin{aligned}
Voter1_5 &\stackrel{def}{=} (publishC, \top).Voter2 + (fail, f).VoterFailYes \\
Voter2 &\stackrel{def}{=} (check, p * c_4).Voter3 + (check, (1-p) * c4).Voter21 \\
&\quad +(fail, f).VoterFailYes2 \\
Voter2_1 &\stackrel{def}{=} (sendCo, s_7).Voter4 + (fail, f).VoterFailYes2 \\
Voter3 &\stackrel{def}{=} (appeal, a_1).Voter3_1 + (fail, f).VoterFailYes2 \\
Voter3_1 &\stackrel{def}{=} (sendCo, s_7).Voter4 + (fail, f).VoterFailYes2 \\
Voter4 &\stackrel{def}{=} (new, \top).Voter + (fail, f).VoterFail1
\end{aligned}
$$

$$
\begin{aligned}
VoterFail &\stackrel{def}{=} (publishC, \top).VoterFail1 + (repair, r_1).Voter0 \\
VoterFail1 &\stackrel{def}{=} (new, \top).VoterFail + (repair, r_1).Voter4 \\
VoterFail2 &\stackrel{def}{=} (sendV, \top).VoterFail + (repair, r_1).Voter0_6 \\
VoterFailYes &\stackrel{def}{=} (publishC, \top).VoterFailYes2 + (repair, r_1).Voter1_5 \\
VoterFailYes2 &\stackrel{def}{=} (new, \top).VoterFail + (repair, r_1).Voter2
\end{aligned}
$$

In this adaptation it is assumed that if a voter has voted and active when the vote is counted then it will be able to take part in the count, regardless of how many failures have occurred between voting and counting. Furthermore, if a voter hasn't voted, is active and the votes have not been published then it can vote, again regardless of any failures that have occurred. $VoterFail$ now represents the behaviour whereby the voter hasn't voted in the current election and voting is still active. Repair from this state causes the voter to return to its initial behaviour. If the voter hasn't voted and the voting is over ($VoterFail2$) then a repairing voter will evolve to $Voter4$ to await a new election. The exception to this is if a voter fails and repairs during the exchange with the administrator; in this case the voter will return to waiting for the $sendV$ from the $Administrator$.

5 Numerical results

The models presented above are now evaluated numerically using data derived from an implementation of the voting scheme. The data is based on using RSA with a key length of 1024 bits, a maximum bit commitment length of 50 bits, a random padding of 100 bytes per message and a mix message block size of 110 bytes. By far the most significant time delays in the scheme are the decryption of the blinded votes and revelation messages. Other significant delays are encountered in the communication involved in sending the various messages and the overhead in signing the blinded messages. All other actions are very fast by comparison. Experiments with the implementation showed that the system is particularly sensitive to the padding length and mix message block lengths as these impact the slowest operations.

Figures 2 and 3 show the average number of votes cast in an election varied against the number of eligible voters where the average duration of an election is 1000ms. The mean time to choose the vote is 1ms, hence what is being measured here is the electronic means of submitting and counting the votes. Figure 2 shows the exact result as calculated from the basic model. Figure 3 shows the absolute percentage error introduced by each of the approximations; the aggregated model and the queue -based model.

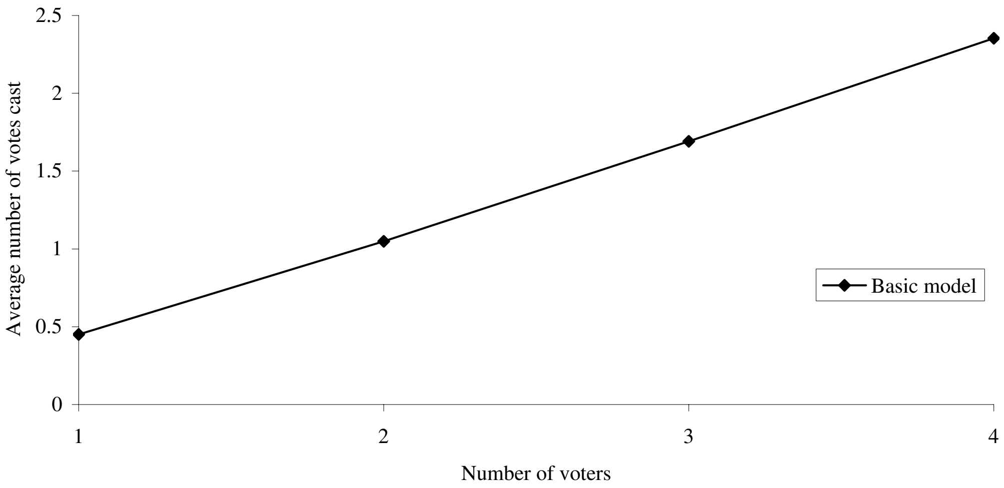


Fig. 2. The average number of votes cast in an election varied against the number of eligible voters

The errors introduced by the approximations are so small as to be indistinguishable were values for these models plotted in figure 2. The plot for the basic model shows a linear growth with the number of voters, i.e. the proportion of successful voters is constant. As expected the error introduced by the aggregate model is considerably less than that for the queue-based model. Although both are small for these values of N, both show a growing trend with N. Hence, whilst this plot indicates the error is acceptable, it cannot yet be assumed that the error would be acceptable for much larger values of N.

6 Conclusions

In this paper a set of models have been developed and analysed for a secure voting scheme. One of the major issues encountered has been the size of the state space of the underlying Markov chains. This means that only a very small

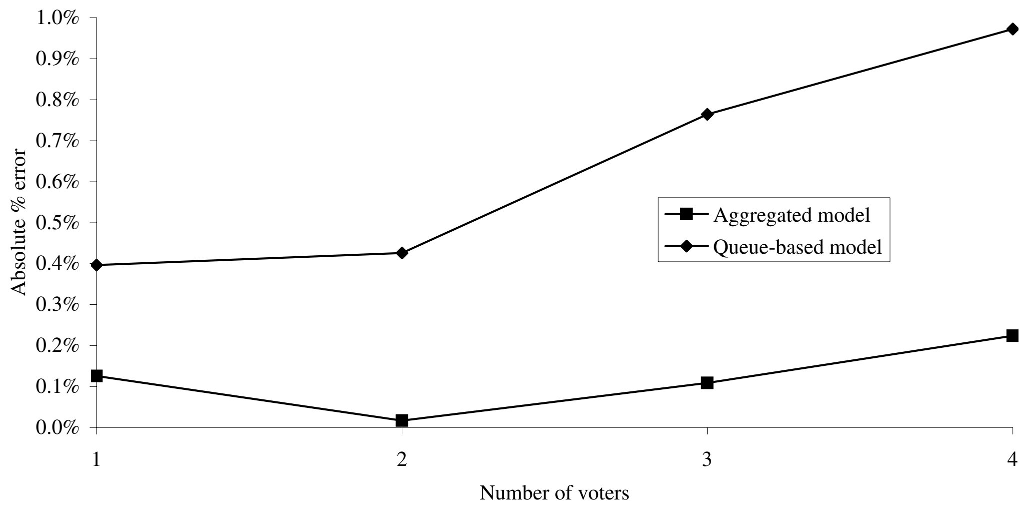


Fig. 3. The absolute percentage error of the average number of votes cast in an election varied against the number of eligible voters

number of voters can be considered in a model. Clearly this is not adequate for practical purposes and an alternative model has been developed to allow larger and more practically relevant models to be studied. These models have been evaluated numerically using data captured from an implemented version of this scheme. The results show an acceptably small degree of error. However, there is a growth in the error with the number of voters, so it is impossible to say at this stage whether models of practical numbers of voters can be analysed accurately in this way. This implies that the basic model must be evaluated for larger numbers of voters to gain greater confidence in the approximations introduced. It is possible that tools such as PRISM [10] and ipc [3] will be able to cope with such models where the PEPA Workbench could not. This remains an area of continuing investigation.

The parameters for this model are several orders of magnitude different from one another. This means there are considerable problems in gaining and using accurate results. In this study the consequences of these problems has not yet been addressed.

Acknowledgements

An implementation of this scheme was part of an final year undergraduate project at the University of Durham in 2000/01 undertaken by Oliver Davis and supervised by the author. The author is indebted to Mr Davis for introducing him to the voting scheme used in this paper.

The author would also like to thank the anonymous reviewers for *PASM 2004* who were extremely supportive and gave valuable criticism and insight.

It is heartening to encounter reviewers who give so much time to the review process.

References

[1] H. Bowman, J.W. Bryans and J. Derrick, Analysis of a Multimedia Stream using Stochastic Process Algebra, *The Computer Journal*, **44**(4), 2001.

[2] D. Chaum, Untraceable electronic mail, return addresses and digital pseudonyms, *Communications of the ACM*, 24(2), pp. 84-88, 1981.

[3] J. T. Bradley, N. J. Dingle, S. T. Gilmore, and W. J. Knottenbelt, "Derivation of passage-time densities in PEPA models using ipc: the Imperial PEPA Compiler," in *MASCOTS'03, Proceedings of the 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems* (G. Kotsis, ed.), (University of Central Florida), pp. 344–351, IEEE Computer Society Press, October 2003.

[4] G. Clarke, S. Gilmore, J. Hillston and N. Thomas, Experiences with the PEPA Workbench modelling tools, *IEE Proceedings – Software*, **146**(1), 1999.

[5] L. Cranor and R. Cryton, Sensus: A Security-Conscious Electronic Polling System for the Internet, in: *Proceedings of the Thirtieth Hawaii International Conference on System Sciences (HICSS 30)*, Volume 3, pp. 561-570, IEEE Computer Society, 1997.

[6] A. Fujioka, T. Okamoto and K. Ohta, A practical secret voting scheme for large scale elections, in: J. Seberry and Y. Zheng (eds), *Advances in Cryptology - Proceedings of AUSCRYPT '92*, LNCS 718, pp. 244-251, Springer-Verlag, 1993.

[7] M. Herschberg, Secure Electronic Voting Over the World Wide Web, MEng Thesis, MIT, 1997. *http://theory.lcs.mit.edu/~cis/voting/voting.html*

[8] J. Hillston, *A Compositional Approach to Performance Modelling*, Cambridge University Press, 1996.

[9] K. Iversen, A cryptographic scheme for computerised general elections, in: J. Feigenbaum (ed) *Advances in Cryptology – Proceedings of CRYPTO '91*, LNCS 576, pp. 405-419, Springer-Verlag, 1991.

[10] M. Kwiatkowska, G. Norman and D. Parker, PRISM: Probabilistic symbolic model checker, in: A. Field, P. Harrison, J. Bradley and U. Harder (eds.) *Computer Performance Evaluation: Modelling Techniques and Tools*, LNCS 2324, pp. 200-204, Springer Verlag, 2002.

Quantitative assessment of a peer-to-peer cooperative infrastructure using Stochastic Well-Formed Nets

Carlo Bellettini [1] Lorenzo Capra [2] Mattia Monga [3]

DICo – Dipartimento di Informatica e Comunicazione
Università degli Studi di Milano, Italy

Abstract

Traditional support tools for software engineers, normally based on a client-server architecture, are unsuitable to deal with the new issues emerging from the current (and future) cooperative work scenarios (where connectivity is intrinsically transient, the number of interacting partners dynamically changes, etc.). This paper presents a quantitative assessment of a fully decentralized, peer-to-peer, cooperative infrastructure. Stochastic Well-Formed Nets (SWNs) modelling the new peer-to-peer architecture, and a traditional (client-server) one, are developed and analysed: we used SWNs for their ability to directly exploit the symmetries intrinsically present in the modelled systems, thus greatly reducing the complexity of the analysis. The main goal is to compare the impact of the two alternative protocols on the collaborative work. Together with the performance figures of interest, methodological issues concerning the choice of the most appropriate model abstraction level, the adoption of a compositional modelling approach, and the management of the model complexity are also discussed.

Keywords: SWNs, peer-to-peer, configuration management

1 Introduction

Nowadays the Internet infrastructure is so pervasive that it is common that people connect their laptop computer from a range of different locations: office, home, the hotel hosting them for a conference, or the meeting room where they are working. This is sometimes called mobile computing and it forces designers of applications to take into account two new requirements: (1) users may

[1] Email: carlo.bellettini@unimi.it
[2] Email: capra@dico.unimi.it
[3] Email: mattia.monga@unimi.it

This is the workshop version of the paper. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

connect to the network from arbitrary locations (usually with different network addresses), and (2) they are not permanently connected. Thus, connectivity is intrinsically transient, and machine disconnection is not an exceptional case, but the normal way of operating.

Collaborative work over the Internet is often pursued by manipulating electronic artifacts that are shared with others by exploiting the mediation of some server. This scenario is still common and useful in several cases. However, since work activities are no more exclusive of office settings, users are often forced to work without any Internet connection. Thus, network applications that rely on servers are sometimes not desirable or even not feasible. In other words, the client/server approach is possible, and common, in all cases where a reliable and permanent network infrastructure is available to connect the participating nodes. However, in many cases, people would like to collaborate while they are supported through a much looser architecture, since they cannot or do not want to afford the cost of setting up a central server. The resulting reference architecture is a network of peers, each of which contributes to the overall logical structure in an equivalent way. Moreover, peers cannot be assumed to be always on-line, i.e., a peer is not always reachable by others. Since the network connection is intrinsically intermittent, peers may dynamically join and leave an ad-hoc community. They join it in impromptu meetings, where they synchronize their works.

We focussed our investigation on the collaborative work needed to produce software systems. Software developers typically collaborate by exchanging and sharing a number of files. Files are assigned to people according their responsibilities in the project. However, besides the person in charge of a file, several other collaborators sometimes need to view or modify it. In general, for each item we can identify the role of an *owner,* i.e., the individual who has created the artifact or who is in charge of carrying out the work on it. Moreover, there is a number of other collaborators involved in the project who need to manipulate items that are not under their direct control, i.e., artifacts they do not own. System designed to help software developers in keeping their work coherent and tracking the evolution of artifacts are called *configuration management tools.*

The underlying architecture of these tools affects the ongoing collaborative work. In fact, a server based solution forces people to keep their work consistent with the copy on the server machine. In traditional, client/server version management tools, two (or more) persons may work on the same artifact (a file): both are required to check-out the artifact from the server machine and after any modification it has to be accepted on the server machine by trying a commit (or check-in) operation. Different control policies for concurrent work are possible. Artifacts could be locked by the system while people are modifying them, in order to avoid concurrent modifications. Instead, *optimistic* concurrency control [9] means that if Alice and Bob are both working on `foo.c` and Alice commits her new version in the central repository, when Bob tries to check-in his own modified version he gets a conflict from the server and he must merge his modifications with Alice's ones before trying a new check-in.

PeerVerSy [1] is a version management tool that allows users for freely accomplishing their computations and collaborative actions as check-in and check-out also when they are not able to communicate with the machine that holds the reference copy of an artifact. Concurrency control is basically optimistic and an automatic reconciliation step is performed when connection is established again, possibly arising conflicts. The client/server approach assumes the availability of the network infrastructure even in the frequent case that no concurrent work is done on a particular item. It is perfectly reasonable and desirable that one could check-in a file which is under his/her control if no other developers want to manipulate it. Similarly, check-out operations can be performed also when the latest version of an artifact is available somewhere– not necessarily on repository servers– but for example on the local file system or on the file system of any connected node.

In the PeerVerSy approach there are no well known servers. Instead, it is based on the notion of the *authority* for a set of items (it *owns* them), and the copy of an artifact owned by the authority is the *master* copy. In addition to the master copy, other peers can keep a local copy (*replica*) of the documents they do not own in order to allow users to work on them even when the authority is not reachable or when the peer is disconnected from the network. In fact, a user can perform both check-in and check-out operations also from the local copies of a document and the only difference between the master copy and a replica is that a check-in of a new version becomes definitive and available for all users only when the authority accepts the changes and updates the master copy.

Intuitively, the peer-to-peer protocol may affect the number of conflicts generated during the collaboration. The analysis described in this paper aims at comparing the performances of a server based optimistic protocol with the PeerVerSy protocol. In particular, we wanted to measure the impact of the underlying architecture on the frequency of merge operations. In our experience with the tool we found that PeerVerSy was extremely attractive for small (< 10 people) groups of developers. In fact, specially when several organizations are involved (as it is often the case in academics and research settings), is sometimes unsuitable to set up a server based configuration management tool: a solution that allows people for using their own machine, installing and configuring the program on their own responsibility is much more appealing for its improved flexibility. Moreover, users are happy to be able to freely cooperate wherever they want, and even work at home without any Internet connection. However, we wanted to be able to assess the impact of the approach on the work in a quantitative fashion, out by means of stochastic models. Although initially a simple simulation environment for PeerVerSy was set up [11] (on which a preliminary evaluation of the system was performed), we eventually preferred modeling essentially because, notwithstanding approximations intrinsic in both approaches, due to unavoidable abstractions in the system representation, whenever the analytical model solution is feasible it is more accurate and allows a cheaper and more flexible assessment of the efficiency of the two alternative approaches for several and dynamically changing work scenarios. It is worth noting that our goal was to assess the impact of the protocol on collaborative work. Thus, our model completely ignores the efficiency of the implementation and the load of the network. Moreover, reliability of links and peers is not modeled: instead we just consider peers as *on-line* or *off-line* with respect to the Internet.

A critical issue concerns the detail level of the models, that when performing analysis must be carefully chosen in order to avoid state space explosion. The formalism adopted for the modeling and analysis is a *colored* flavour of stochastic Petri nets called Stochastic Well-formed Nets (SWNs) [4]. Both quantitative and qualitative analysis can be performed on these models. The peculiar feature of SWNs is that due to the particular syntax adopted for the model color structure behavioral symmetries can be automatically discovered and exploited to build an aggregate state space and corresponding stochastic process (a lumped Continuous Time Markov Chain).

The paper is organized as follows: In Section 2 the SWNs formalism is introduced; in Section 3 the models of the client-server and peer-to-peer protocols for collaborative work are presented; some methodological issues involving the adopted modelling approach are discussed; in Section 4 we present and comment on performance indices derived from the model steady-state analysis: these indices allow us to assess the impact of the two alternative solutions on cooperative work scenarios characterized by small group of developers; the complexity of the model analysis, both in terms of memory (state-space explosion) and time, is also addressed. Finally, in Section 5 we summarize the main results achieved and we outline possible directions for future work.

2 Stochastic Well-formed Nets

We recall here only the basic notation and concepts about the SWN formalism needed to understand the models presented in the paper. We assume that the reader is familiar with the PN and Generalized Stochastic PN formalisms [3] (GSPN, the unfolded version of SWN), and has some basic knowledge of high level PN (HLPN) extensions [8] (see [4] for a complete definition of SWNs)

As in all HLPN formalisms, tokens in places are associated with an identifier (color), similarly transitions are parameterized, so that different (color) instances of a given transition can be considered for enabling and firing. Arc functions associate each transition instance with a multiset of colored tokens to be withdrawn from/put into a place.

A SWN model is a eleven-tuple

$$\left(P, T, C, \mathcal{C}, W^{+}, W^{-}, H, \Phi, \Pi, \Omega, \mathbf{M}_0\right)$$

where P is the set of places, T is the set of transitions, $C = \{C_1, \ldots, C_n\}$ is the set of basic color classes, each basic color class C_i is a finite, non empty set of colors. Each basic color class can be partitioned into static subclasses $C_{i,1}, \ldots, C_{i,k}$, if it is necessary to make a distinction among groups of colors of the class. The splitting of basic classes is necessary to represent asymmetric qualitative and/or quantitative behaviors of components of the same nature [4].

$\mathcal{C}$ is a function associating a *color domain* to each place and transition. A color domain is defined as the Cartesian product of basic color classes (the same color class may appear several times in a color domain), hence the color associated with tokens in place p as well as the *color instances* of a transition t, take the form of *tuples* of basic color class elements.

The color domain of t is implicitly defined by the set $Var(t)$ of *variables* inscribing the arcs (whose kind may be input, output and inhibitor - I, O, H) surrounding t. Each variable (that may be also seen as a *projection* function) refers to a given basic color class C_{x_i}. The color domain $\mathcal{C}(t)$ is defined as $C_{x_1} \times \ldots \times C_{x_m},\ x_i \in Var(t)$. A color instance of t ((t, c)) can thus be interpreted as a consistent assignment of colors to the variables in $Var(t)$.

W^-, W^+ and H represent respectively the input, output and inhibitor arcs of the model, and the associated arc functions. The function f on an arc connecting place p and transition t is defined as $f : \mathcal{C}(t) \to Bag(\mathcal{C}(p))$ where $Bag(A)$ denotes the set of all multisets that can be built on set A; the syntax for expressing such functions takes the form of a sum of weighted and guarded tuples of *elementary functions* defined on the basic color classes. The allowed elementary functions are the *the projection*, selecting one element of a transition instance color tuple (projection symbols can be arbitrarily chosen: e.g., in our examples we use symbols $x, y, st, \ldots$), and the *diffusion* constant functions, returning the set of all elements in a given basic color class (S) or in a given static subclass ($S\,C_{i,j}$)[5].

Φ is a T-indexed function, expressing each transition guard: a guard is used to restrict the set of admissible color instances of a transition, to those satisfying a given predicate. The predicate must be expressed as a boolean expression whose basic terms are *standard predicates*, representing simple conditions on the transition instance tuple elements. Standard predicates are defined to test either equality of pairs of colors bound to variables in $Var(t)$ (e.g., $x = (<>)y$), or to check the membership of a color to a specified static subclass (e.g., $d(x) = (\neq)C_{i,k}$).

The transition priority Π is a T indexed function associating a priority level (in $\mathbb{N}$) to each transition; priority level 0 is reserved for *timed* transitions (graphically represented as white boxes), while all other priority levels are for *immediate* transitions (graphically represented as black bars), which fire in

[4] A basic color class may be circularly ordered, but for simplicity we omit this part of the definition here, being not used in our models.
[5] The successor of a projection is also allowed on ordered basic color classes.

zero time; conflicts between transitions with different priority level are deterministically resolved in favor of the transition with higher priority. Finally Ω is a T indexed function assigning a firing rate to each timed transition t ($\Omega(t)$ is the rate of the exponential pdf characterizing the random firing delay associated with t) and a *weight* to each immediate transition, used to probabilistically characterize the conflict resolution policy between immediate transitions with equal priority.

A transition instance (t, c) has concession in marking $\mathbf{M}$ iff

(i) for each input place p of t $W^-(t,p)(c) \leq \mathbf{M}(p) \ \wedge$

(ii) for each inhibitor place p of t $H(t,p)(c) > \mathbf{M}(p) \ \wedge$

(iii) $\Phi(t)(c) = \text{true}$;

(the $>, \leq$ relations and the$+, -$ operations are implicitly extended to multisets).

A transition instance (t, c) is enabled in marking $\mathbf{M}$ if it has concession in $\mathbf{M}$ and no other transition instance with higher priority has concession.

A transition instance which is enabled in marking $\mathbf{M}$ can fire, leading to the new marking $\mathbf{M}'$:

$$\forall p \in P, \mathbf{M}'(p) = \mathbf{M}(p) + W^+(t,p)(c) - W^-(t,p)(c)$$

Here we only recall that as a result of the adopted time representation, the reduced Reachability Graph of a SWN (the graph obtained by suitably removing the *vanishing* markings, i.e., those markings enabling at least one immediate transition) is isomorphic to a Continuous Time Markov Chain (CTMC) (see [4] for a detailed discussion).

The peculiar characteristic of SWNs is that (due to the particular syntax adopted for color domains, arc functions, and guards) behavioral symmetries can be automatically discovered and exploited to build an aggregate state space (called *symbolic reachability graph* or SRG) [5] and corresponding stochastic process (a *lumped* CTMC). However in order to fully characterize the underlying stochastic process, the immediate transitions priority and weight specification must be properly specified by the modeler (see [3] for a discussion of this topic concerning GSPNs).

3 The SWN models of two alternative architectures

In this section we present the SWN models of the client-server architecture adopted by consolidated configuration management tools, and the (new) peer-to-peer architecture implemented by the `PeerVerSy` tool. The models have been built to estimate the impact of the two approaches on small groups of developers freely cooperating, and intermittently connected to some network infrastructure (a LAN, the Internet, etc.). The starting point for producing models has been the `PeerVerSy` architecture specification [1], a mix of pseudocode and natural language, integrated with some UML diagrams.

The SWN models have been developed and analyzed (by means of the `GreatSPN` tool [6]) by the following steps: (1) first the model of the client-server architecture was built; (2) the (complex) model of the peer-to-peer architecture was defined by adopting a consolidated compositional approach, based on place superposition, supported by the GreatSPN tool [2]; (3) a submodel was refined to represent the infrastructure that realizes the distributed repository of artifacts; (4) since submodels (and the overall models as well) are fully parametric, we made explicit the relations among component parameters. The main model parameters are:

- the number of cooperating workers,
- the number of artifacts on which workers collaborate,
- the association between a set of artifacts and one worker who plays the role of authority for them.

Parameters are instantiated by properly defining the model *color domains* and *the initial marking*.

Throughout our work (e.g., in step (2)) we tried to establish a relation between different abstraction levels: a thorough discussion on how *behavior inheritance* notions can be used to establish consistency relationships between (sub)models at different abstraction levels may be found in [14] (by which we have been inspired).

Concerning the abstraction level of models, we observe that very detailed models (although may be very useful for integrating and completing the existing documentation) are hardly treatable for performance analysis because of the state-space explosion they trigger.

The following issues have been considered in choosing a convenient abstraction level for our models: (I) whether to represent the underlying communication infrastructure and resource contention (for CPU usage and communication links), (II) whether to represent a single artifact or a set of artifacts shared by the group of developers (the former situation being better supported by the client-server architecture, while the latter one by the peer-to-peer architecture), (III) whether to consider possible failures of the communication links or of the nodes of the network (obviously the client-server architecture presents a single point of failure, while the peer-to-peer architecture is intrinsically more fault-tolerant).

In this paper we decided to not consider neither the communication infrastructure nor the possible occurrence of faults. We focused indeed on the precise description of the client/peer life cycle. Moreover, we restrict our analyses to collaboration based on a single artifact. These simplifications on one side allowed us to study the performance of many more configurations than those manageable on the detailed models by the `GreatSPN` tool. On the other side, coherently with the final goal of the paper (i.e., providing a first assessment of the impact of the the peer-to-peer protocol on the cooperative work), they correspond to a worst-case assumption for the peer-to-peer architecture.

Another motivation for studying the system behavior with explicit representation of resource contention and/or in presence of faults separately, stems from the observation that the specification of related stochastic parameters (differing for several magnitude orders from the parameters characterizing user activities such as check-in and check-out) should cause stiffness in the underlying stochastic process.

Basic Color classes

Both SWN models have two basic colour classes: UID, denoting the group of cooperating workers, and ST denoting the possible status (*online,offline*) of a worker (ST is accordingly partitioned in two static subclasses *on,off* each of cardinality one). The cardinality of class UID is one of model's parameters. In addition, the SWN model of the peer-to-peer configuration tool has the basic class UPD, which denotes the status (*updated,non-updated*) (corresponding to the UPD's static subclasses *upd,noupd*) of the local copy of the artifact owned by a given worker.[6]

3.1 The SWN model of a traditional versioning system

The SWN model of the client-server architecture for a configuration management tool is depicted in Figure 1. In both models, timed transitions represent time-consuming activities (time being spent for decision and/or execution), while immediate transitions represent *logical* (sequences of) actions. On-line and off-line periods of a client/peer are also represented by means of a timed transition.

Color domains of the model

Let us briefly comment on the definition and the interpretation of the place and transitions color domains (the discussion made here applies to a consistent part of the peer-to-peer architecture's model). All places of the net but place $Status$ have color domain UID: a token $\langle u_1 \rangle$ in one of these places represents that worker u_1 has reached a particular status of his/her life-cycle.

In the real world, each client checks out a working copy with a specific version number, and by comparing this with the version of the last copy of the artifact on the server he is able to establish the up-to-date status. In order to exploit the symmetries of the model, we chose not to use the version numbers but to directly maintain the up-to-date status information in the place *uptodate*.

Place $Status$ has color domain $\mathcal{C}(Status) = UID \times ST$: a token $\langle u_1, on \rangle$ in $Status$, for example, means that u_1 is currently on-line.

[6] In the client-server model the up-to-date status of the artifact is denoted by the marking of the homonym place

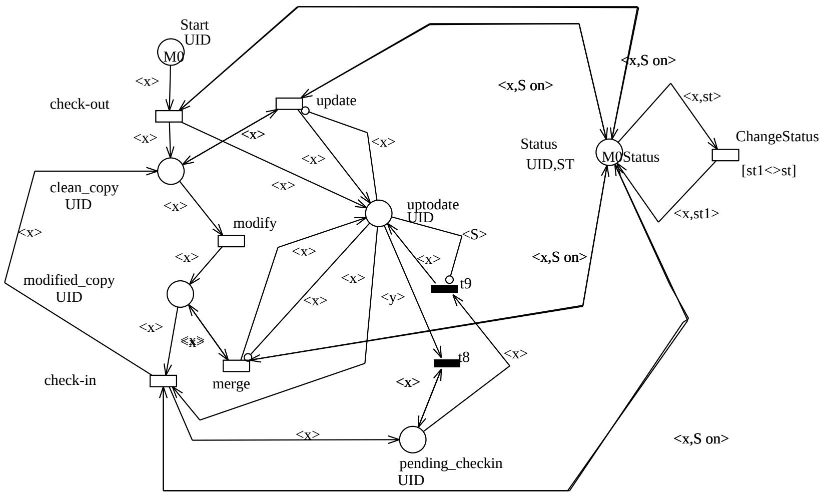


Fig. 1. The SWN model of a server-based version system

The set of model's timed transitions has color domain UID: a color instance of any transition in this group represents a particular action performed by a given worker during his/her life-cycle. These actions may be accomplished by *on-line* users only, because they require the access to the server. The only notable exception is the $modify$ action that can be accomplished off-line on a previous checked-out version of the artifact.

Timed transition $ChangeStatus$ has color domain $\mathcal{C}(ChangeStatus) = UID \times ST \times ST$: a color instance $\langle u_1, s1, s2 \rangle$ means that the current connection status ($s1$) of a worker (u_1) changes (the new status being $s2$) with a given frequency, that is one of the timing parameters of the model. In other words, transition $ChangeStatus$ models the *online* attitude of the workers.

The two immediate transitions $\{t8, t9\}$ (having color domains $UID \times UID$ and UID, respectively) permit us to keep coherent and correct the up-to-date status information after the check-in operations. In particular the cumulative effect of the firings of such transitions as consequence of a check-in is to empty the *uptodate* place except for the token with the user-id (UID) of the client that accomplished the check-in action.

3.2 The SWN model of a peer-to-peer version system

The SWN model of the peer-to-peer architecture for a configuration management tool is depicted in Figure 2. It comprises two main parts, each enclosed in the picture within a dashed box: i) the representation of the main peer life-cycle, obtained by simply refining the SWN model in Figure 1, ii) the representation of the infrastructure reaction to a peer that goes online.

With respect to the client-server architecture's model, places *Repository*

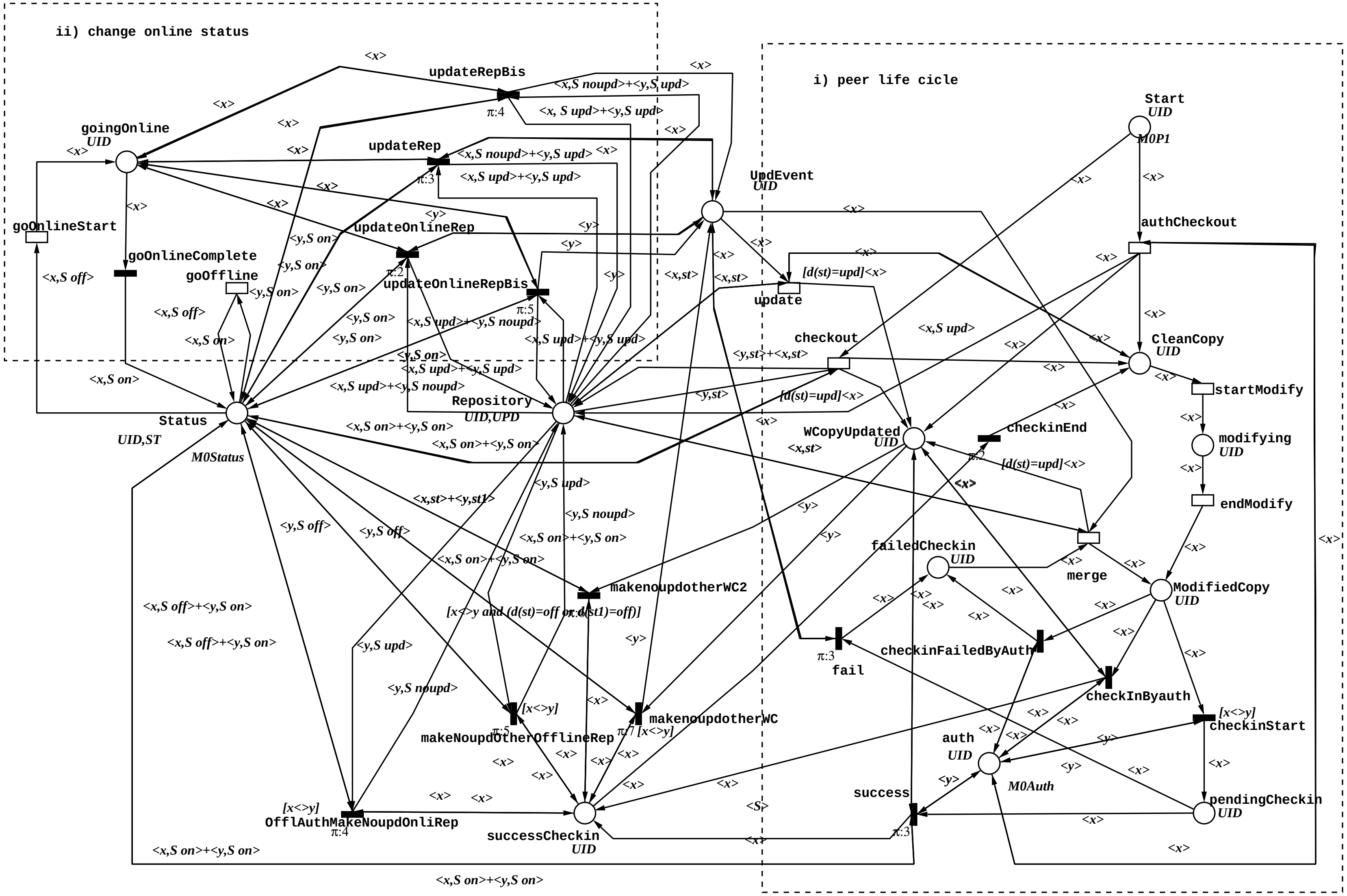


Fig. 2. The SWN model of a peer-to-peer version system

and $WCopyUpdated$ extend the *uptodate* place considering not only the status of the explicitly checked out copy ($WCopyUpdated$), but also the status of the repository replica automatically managed by the `PeerVerSy` infrastructure. The *Repository* place has color domain $UID \times UPD$ (a new color domain): a token $\langle u_1, noupd \rangle$ in place *Repository* means that the *working copy* owned by u_1 is not updated.

Finally another place ($UpdEvent$) has been added in order to represent the fact that the system sends a signal to a peer that a new version of the artifact is available. We highlight here that this is an important peculiarity of the `PeerVerSy` system: it makes aware the peer as soon as possible that the working copy is not still up-to-date. Using such an information, the peers are sometime able to avoid modify actions on old versions of the artifact.

The updating of these three places is managed automatically by the infrastructure. In particular the upper part of the net represents what happens when a peer goes online. In the case that the version of such a peer is newer than those of the already on-line peers, the repository replica of such peers are automatically updated and a "New version" signal is sent to them. On the contrary in the case that the peer copy is older, its repository is updated and he receives the signal. This signal is thus an indication that the repository replica contains a more recent version of the checked out working copy. The bottom part of the net represents what happens when a check-in is accomplished: the working copy status of all the other peers is set to non-uptodate. The repository replica status of all the off-line peers is set to non-uptodate. A new version available signal is sent to all the online peers. All the conflicting pending check-in are then aborted ($fail$ transition). Finally is also considered the case that the check-in is accomplished by the authority when he/she is off-line. In this case no signal is sent and all working copies and repository replica are set as non-uptodate.

Regarding the peer life-cycle (sub)model, we had to differentiate the authority from normal peers in some operation because the prerequisites are different (e.g., the authority can accomplish a check-out, or a succeeding check-in, also if he/she is off-line). However the main change is in the check-in action. In fact this becomes a complex operation that is executed in several steps. In particular the peer can issue a check-in request also if he and/or the authority are not currently on-line. The status of such check-in is thus *pending*, waiting for a condition in which it is possible to evaluate whether the check-in succeeds or not. Several conflicting check-in can be at the same moment in the pending status.

4 Peer-to-peer against client-server: preliminary evaluation

Generally speaking, performance analysis of the SWN models can be used for several purposes: (1) for model validation, in order to compare performance measures of models at different detail of abstraction: in this way the approximation introduced by abstraction can be evaluated. (2) When comparing the two alternative architectures, to have a first hint on the impact on the reference scenario (small groups of users cooperating on the same document) of the peer-to-peer protocol: the difference between the results corresponding to alternative models allows one to derive some preliminary conclusions on the merits and problems of each, even if the absolute measures are not very precise. (3) In the model's tuning phase, to achieve more accurate and reliable performance indices: all approximations introduced in the modeling phase should be carefully considered, and a validation of the model against actual measures on prototypes should be performed. The results presented in this section are examples of second and third model usage scenarios.

4.1 Managing the model complexity

Before commenting on the obtained performance figures (selected among a number), let us describe the techniques we employed in order to manage the complexity of the analysis performed on the SWN models presented in the section above (in particular, of the peer-to-peer protocol model).

Exploitation of symmetries

The models we are considering in this study are *highly symmetric*, the symmetry of a SWN model being measured by the splitting level of basic color classes into static subclasses. We do appreciate that both models we have developed are fully symmetric with respect the color class UID, which means a *potential* reduction factor (with respect to the number of nodes of the ordinary model's RG) of $n!$, where n is the cardinality of UID.

In order to exploit the behavioral symmetries present in the models, however, we need to define a suitable initial *symbolic marking* (SM). The SM notion makes it possible to define an equivalence relation between states in a purely syntactical fashion: two ordinary markings are equivalent if they can be obtained from each other by applying a permutation on the objects of split color classes that preserves static subclasses. The SWN symmetry property ensures that equivalent firing sequences can fire from equivalent markings.

Referring to the model in Fig.1 the formal representation of its symbolic initial marking $\widehat{\mathbf{M}_0}$ is:

$$\widehat{\mathbf{M}_0}(Start) = \langle Z_{UID} \rangle; \widehat{\mathbf{M}_0}(Status) = \langle Z_{UID}, Z_{on} \rangle$$
$$Z_{UID} = n, Z_{on} = 1.$$

The symbolic marking representation is given in terms of *dynamic subclasses* (Z_{UID}, Z_{on} in the example). Dynamic subclasses define a *parametric partition* of static subclasses. Every dynamic subclass refers to a static subclass or to a whole basic class (in the event of non split class), and has an associated cardinality. The symbolic marking above represents the system configuration where all workers are in place (state) *Start*, and they are all *on line*. Note that it corresponds to exactly one ordinary marking (this is a frequent situation for the initial SM). A Symbolic Firing Rule allows the building of the Symbolic Reachability Graph (SRG) of the model, working directly at the level of the SM representation.

The formal representation of the symbolic initial marking $\widehat{\mathbf{M}_0}'$ of the model in Fig.2 is a bit more complex:

$$\widehat{\mathbf{M}_0}'(Start) = \langle Z^1_{UID} + Z^2_{UID} \rangle;\ \widehat{\mathbf{M}_0}'(Status) = \\ \langle Z^1_{UID} + Z^2_{UID}, Z_{on} \rangle;\ \widehat{\mathbf{M}_0}'(Auth) = \langle Z^1_{UID} \rangle \\ Z^1_{UID} = 1, Z^2_{UID} = n-1, Z_{on} = 1.$$

The marking $\widehat{\mathbf{M}_0}'$ represents a generic system configuration similar to the one described by $\widehat{\mathbf{M}_0}$ for the other model: it means in addition that one (arbitrary) user is chosen as the authority for the artifact. The identity of the authority is not fixed, so that $\widehat{\mathbf{M}_0}'$ actually corresponds to a class of ordinary markings. Since there are n different (equivalent) possible choices, we argue that in this SWN model the *actual* reduction factor due to the symmetry exploitation should be divided by n, i.e., it will be at most $(n-1)!$.

Reducing the number of vanishing markings

Symmetry exploitation alone is not sufficient to make the analysis of the model in Fig.2 manageable: due to the enormous number of vanishing SMs generated during the model's SRG construction (because of the frequent interleaving among immediate transition firings), only configurations with very small groups of users (< 4) can be actually analyzed. Structural analysis techniques may significantly improve the effectiveness of state-space based analysis. In [7] the basis for a *symbolic* approach to express structural relations in *coloured* nets is proposed. The main contributions provided in [7] are the possibility to represent structural relationships between transitions, such as conflict or causal connection, by means of algebraic manipulation of the colour annotations of the net. They show how the theory may be practically applied to the class of (colored) Unary PN to compute *extended conflict sets* (ECS): ECS (originally defined in [3] for GSPNs) represent sets of immediate transitions that are not independent, i.e., such that the firing of one transition in the set may affect the enabling status or the firing probability of the other transitions in the same set. This step on one side is essential to the correct net-level definition of the stochastic process representing the behaviour in time of stochastic Petri nets with immediate transitions (see [13]). On the other side, it allows a drastic reduction of the number of vanishing markings generated during the construction of the reachability graph: transitions belonging to different ECS may indeed fire according to an arbitrary order (fixed, for instance, by assigning different priority levels to different ECS), without altering the model (quantitative/qualitative) semantics. When considering high level stochastic Petri nets (such as SWN), the dependency relation should be stated in a symbolic form, to characterize in a compact and parametric way the *color instances* of a given (immediate) transition that can influence the enabling/firing probability of other transition color instances.

The limit of the results presented in [7] is the restricted class of HLPN that is treated. The structural analysis technique presented in [7] has been recently extended to SWNs [10]. Such extension is based on a rather complex formal treatment of the color annotations (arc functions and transition guards) appearing on the high level model: the formal aspects of the calculus is completely worked out in [12].

Let us only better explain the meaning of a structural symbolic relation, by giving the formula describing the structural conflict relation between transitions t and t' when they share an input place p:

$$\mathrm{SC}(t,t') = \overline{W^-(t,p) - W^+(t,p)}^{\,t} \circ W^-(t',p)$$

The first member $W^-(t,p) - W^+(t,p)$ is the functional description of the multiset of tokens actually removed from place p by the firing of an istance of t. The *support* of such expression $\overline{W^-(t,p) - W^+(t,p)}$ maps the multiset to which it is applied into the set taking those colours whose multiplicity is strictly greater than zero. The *transpose* of $\overline{W^-(t,p) - W^+(t,p)}$ is a functional description of the colours instances of t that actually removes tokens from p. Concluding, applying $\overline{W^-(t,p) - W^+(t,p)}^{\,t}$ to the set $\overline{W^-(t',p)}$ then the colour instances of t removing from p some token for which t' has need, are obtained.

In a similar manner it is possible to derive the functional description of the other structural relations (indirect conflict, causal connection, mutual exclusion) on which the ECS computation relies. By applying this structural analysis technique on the model in Fig.2 we were able to partition the set of immediate transitions into a number of independent ECS with different priority levels, so making the analysis of larger configurations of users feasible.

Removing transient markings

Another simple way for reducing (even if with a lower effectiveness than the two techniques above) the state-space growth of the SWN model in Fig.2 stems from the particular structure of the model's SRG. We can easily argue that this SRG (for any n) is not *strongly-connected*, however it contains *one maximal* strongly-connected component. A steady-state solution does exist, where transient states (i.e. those SMs, as the initial one, that are not home-states) have an associated null probability. We can thus skip xthe transient part of the SRG, obtaining an equivalent (in terms of stationary probability vector) stochastic process whose states have a non-null stationary probability. This actually corresponds to skipping the checkout actions, and it is achieved, for instance, by setting as initial SM of the SWN model $\widehat{\mathbf{M}_0}''$:

$$\widehat{\mathbf{M}_0}''(CleanCopy) = \langle Z^1_{UID} + Z^2_{UID}\rangle;\ \widehat{\mathbf{M}_0}''(Status) = \langle Z^1_{UID} + Z^2_{UID}, Z_{on}\rangle;\ \widehat{\mathbf{M}_0}''(Auth) = \langle Z^1_{UID}\rangle;$$

$$\widehat{\mathbf{M}_0}''(Repository) = \widehat{\mathbf{M}_0}''(Wcopy) = \langle Z^1_{UID} + Z^2_{UID}, Z_{upd}\rangle$$
$$Z^1_{UID} = 1, Z^2_{UID} = n-1, Z_{on} = 1, Z_{upd} = 1.$$

The simplification just described is also justified by the *transient* analysis made on the original model (that with initial marking $\widehat{\mathbf{M}_0}'$) by means of the `GreatSPN` tools: the state probabilities computed for small values of the time variable indeed shown a rapid convergence of the stochastic process towards its stationary state.

The reduction steps sketched above allowed us to extend the analysis of the peer-to-peer protocol to larger groups of workers. Table 1 shows the sizes of the SRG (in terms of number of nodes) for the two models we developed, faced to the corresponding RG sizes (divided by n). We do appreciate that the reduction factor provided by the SRG is higher for the client-server model than for the peer-to-peer model: this may be intuitively explained by the need of distinguishing the authority-peer from the normal peers. The analysis required 0.250s of CPU time with $n = 2$ and 310m with $n07$ on a Intel Xeon 2.4GHz. The data files generated by the GreatSPN tool with 8 members exceeded the maximum file size (4GB) set on our machine.

Table 1
SRG vs RG

	client-server		peer-to-peer	
♯	SRG (tang. + van.)	RG (tang. + van.)	SRG (tang. + van.)	RG (tang. + van.)
2	63+33	119+66	115+302	230+604
3	399+270	2023+1521	1392+5429	8007+32049
4	1890+1515	31871+30012	10696+55297	223756+1235224
5	7287+6645	478807+543435	61365+394070	5535385+39720045
6	24136+24417	6998159+9326538	286014+2192326	126941010+1141070760
7	71093+78393	100630663+154400169	1139128+10153639	2767706963+30425540229

4.2 Performance figures

In Fig. 3 some results are shown about the performance of the client-server configuration against the alternative peer-to-peer configuration. The plotted curves refer to the average firing frequency (*throughput*) of *checkin* and *merge* transitions, as a function of the number of cooperative workers (up to seven peers/clients are considered), divided by the number of workers.

The *merge* frequency is a metrics that very closely reflects the additional effort spent for recovery actions (whose quantitative evaluation being one of main goals of our analysis), that is introduced by the lack of synchronization typical of optimistic cooperative work scenarios. The *checkin* frequency instead is related to the advances in the project development.

The curves here plotted refer to a "default" setting of timing parameters (the same in both models), and have been chosen as representatives of a *possible* general trend, that however should be confirmed by a more complete

Fig. 3. Mean frequency of checkin and merge in the two alternative configurations

analysis.

Let us remark (and briefly comment on) the simplificative assumptions that were made in setting the model's timing parameters:

- all timed transitions are assigned the same firing delay, except for transition $modify$ that (according to the actual complexity of the involved activities) is assigned a double firing delay.

- any peer/client goes on-line and off-line with the same average frequency (in other words: on-line and off-line periods are assumed to have the same weight).

- we don't differentiate the authority's "work attitude" from that of normal peers: this is (once again) a worst-case assumption for the peer-to-peer model, since in a realistic scenario the authority is the main responsible of the artifact and the one that more often modifies it.

Even if the simplifications above should be carefully considered, the qualitative trend outlined by the curves in Fig. 3 seems reliable. The main highlighted outcome is that (generally speaking) the peer-to-peer protocol behaves in a satisfactory way, comparable to the server based solution. As expected, the authority has a great advantage from the peer-to-peer solution. However, other peers do not pay a big price for this. We experienced that changing some stochastic parameters of the models (e.g., the rate of $modify$ transition) leads to significantly different absolute values, but same curves shape.

5 Conclusion and future work

In this paper we have reported some preliminary results of modeling activity concerning a new peer-to-peer architecture for configuration management tools. A *colored* flavor of stochastic Petri Nets (SWNs) has been used, able to exploit the model symmetries during the analysis phase, thanks to its structured syntax. Two SWN models have been presented: the model of the new peer-to-peer architecture (called `PeerVerSy`), and the model of a traditional client-server architecture. The contribution of the modeling activity has been twofold: on one hand models can be considered as an integration to the available documentation (specially as concerns the `PeerVerSy` tool); on the other hand, they provide a first hint on the impact of the peer-to-peer protocol on the reference cooperative work scenario (small groups of users cooperating on the same document). Steady state analysis has been performed: a pair of performance metrics characterizing the cooperative work have been computed and discussed. The obtained results about the peer-to-peer configuration are promising and deserve further investigations.

Some methodological issues have been also considered: in particular, we focused on the techniques adopted for managing the model state-space explosion, and on the abstractions used both in the modeling phase and in the setting of stochastic model's parameters.

Besides on extending and completing the performance analysis here presented, we are currently working on a better characterization of the workers behavior by including possible scenarios so far not considered (e.g., the possibility of discarding the work done for example as consequence of a conflicting check-in), and by differentiating class of workers (basically authorities and normal peers). The consequences of a multiple documents (artifacts) scenario are also under investigation.

Acknowledgement

The authors would like to thank Davide Balzarotti, Carlo Ghezzi, and all the anonymous reviewers for their insightful comments on preliminary versions of this paper.

References

[1] Balzarotti, D., C. Ghezzi and M. Monga, *Freeing cooperation from servers tyranny*, in: E. Gregori, L. Cherkasova, G. Cugola, F. Panzieri and G. P. Picco, editors, *Web Engineering and Peer-to-Peer Computing*, Lecture Notes in Computer Science **2376**, Springer-Verlag, 2002 pp. 235–246.

[2] Bernardi, S., S. Donatelli and A. Horváth, *Compositionality in the GreatSPN tool and its use to the modelling of industrial applications*, Software Tools for Technology Transfer (2001).

[3] Chiola, G., M. Ajmone, G. Balbo and G. Conte, *Generalized stochastic Petri nets: A definition at the net level and its implications*, IEEE Transactions on Software Engineering **19** (1993).

[4] Chiola, G., C. Dutheillet, G. Franceschinis and S. Haddad, *Stochastic well-formed coloured nets for symmetric modelling applications*, IEEE Transactions on Computers **42** (1993), pp. 1343–1360.

[5] Chiola, G., C. Dutheillet, G. Franceschinis and S. Haddad, *A symbolic reachability graph for coloured Petri nets*, Theoretical Computer Science B (Logic, semantics and theory of programming) **176** (1997), pp. 39–65.

[6] Chiola, G., G. Franceschinis, R. Gaeta and M. Ribaudo, *GreatSPN 1.7: Graphical editor and analyzer for timed and stochastic Petri nets*, Performance Evaluation (1995).

[7] Dutheillet, C. and S. Haddad, *Conflict sets in colored Petri nets*, in: *Proceedings of the 5th International Workshop on Petri Nets and Performance Models*, Toulouse, France, 1993, pp. 76–85.

[8] Jensen, K. and G. Rozenberg, editors, "High-Level Petri nets: Theory and Applications," Springer-Verlag, 1991.

[9] Kung, H. T. and J. T. Robinson, *On optimistic methods for concurrency control*, ACM Transactions on Database Systems (TODS) **6** (1981), pp. 213–226.

[10] L.Capra, M. D. Pierro and G. Franceschinis, *An application example of a symbolic calculus for SWN structural relations*, in: *Proceedings of the 7th International Workshop on Discrete Event Systems*, Reims, France, 2004, to appear.

[11] Monga, M., *Supporting nomadic co-workers: an experience with a peer-to-peer configuration management tool*, in: S. Horiguchi, K. Ochimizu and T. Katayama, editors, *Proceedings of the International Symposium on Towards Peta-Bit Ultra-Networks*, IEEE (2003), pp. 79–87, invited paper.

[12] Pierro, M., "Structural analysis of conflicts and causality in GSPN and SWN: theory and applications." Ph.D. thesis, Università di Torino, Turin, Italy (2004).

[13] Teruel, E., G. Franceschinis and M. De Pierro, *Well-defined generalized stochastic Petri nets: A net-level method to specify priorities*, IEEE Transactions on Software Engineering **29** (2003), pp. 962–973.

[14] van der Aalst, W. and T. Basten, *Life-cycle inheritance, a Petri-net-based approach*, in: *Proceedings of ICATPN 97*, Lecture Notes in Computer Science **1248** (1997).

Analysis of a BMAP/D/1-Timer multiplexer

Gábor Horváth [1] Miklós Telek [2]

Department of Telecommunications
Budapest University of Technology and Economics
Budapest, Hungary

Abstract

In this paper we introduce and analyze a model of a multiplexer queue with a batch Markovian arrival process and a special, timer based, non-work-conserving service discipline. We show that the embedded process at departures is an M/G/1 type process with proper state partitioning, which can be efficiently analyzed by matrix geometric methods. We derive the expressions to compute the distribution of the waiting time. The paper ends with numerical experiments, and points out some interesting features of the system.

Keywords: multiplexer queue, BMAP, matrix geometric method, waiting time distribution

1 Introduction

There are network protocols in packet switched networks in which multiplexers compose fixed size packets from the incoming data units. Such a protocol is the ATM AAL2 protocol [1], which plays an important role in the radio access networks of 3G mobile telecommunication systems.

In an AAL2 multiplexer there is an additional component of transmission delay. When there is not enough data to compose a complete packet the multiplexer waits for incoming data to fill a packet. This additional delay can be limited with the use of a timer. When this timer expires the multiplexer transmits an incomplete packet.

A variant of this kind of multiplexer behaviour is discussed in [9]. In this paper we discuss a continuous time matrix geometric model of this multiplexer behaviour in case of batch Markovian arrival process (BMAP) arrival.

The paper is organized as follows. Section 2 gives the description of the considered queueing model, for which the queue size analysis is provided by

[1] Email: ghorvath@hit.bme.hu
[2] Email: telek@hit.bme.hu

This is the workshop version of the paper. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

Section 3. Based on the queue size analysis the computation of the waiting time distribution is presented in Section 4. Section 5 summarizes the algorithm. Numerical examples and the analysis of the results are provided in Section 6. Finally, Section 7 concludes the paper.

2 Model Description

The BMAP/D/1-Timer multiplexer queue is a continuous time stochastic process. The arrival process is a continuous time batch Markovian arrival process ([6]), where the batch size represents the number of data units arriving at an arrival instant. The server performs the multiplexing: it awaits L data units, compiles one package from them, and serves it with a deterministic service time Δ (according to the FCFS queueing discipline).

There is an extra waiting time for the jobs caused by the server when it waits for more arrivals if there are not enough data units to form a complete output packet. To decrease this waiting time, a timer is introduced (T). This timer ensures that when a data unit reaches the head of the queue (positions 1 to L, thus it will be included in the next packet), it will be served in time T, even if there are not enough arrivals to form a complete packet during this time. In that case the server forms a partially filled output packet, and serves it the same service time (Δ).

To give a better description of the service mechanism, we summarize the behavior in the following 3 points. These 3 points will be referred to many times in the sequel, and will be used as 3 cases requiring different treatments during the analysis:

P1. While there are L data units or more in the queue, the timer is not running. The server takes the first L data units out of the queue, compiles the packet, and starts the service. After the service ends (time Δ), and the queue size still exceeds L, a new service starts. *In our model the queue contains the waiting data units only; the ones whose service already started but not yet finished are not included, they are stored in a temporary buffer.*

P2. If the queue size gets below L but above 0 when the service begins (after taking out the L data units whose service has started), the timer starts. When the service is ready and the queue is still below L, no new service begins until the timer elapses, or the sufficient amount of data arrives (see Figure 1 (b)).

P3. If an arrival happens when the queue is empty, and the batch size of the arrival is below L, the timer is started immediately. The first service can start when the queue size exceeds L, or when the timer elapses. Of course the service can not start while the packet in the server did not leave the system (Figure 2).

The timer value (T) can even be smaller than Δ. In this case only arrivals into empty queue could be affected by the timer, if the remaining service time at the arrival instant is less than the timer. The system is non work-conserving with $T > 0$ and work-conserving with $T = 0$. We focus on the first case here.

Considering the requirements of the system, the setting of the timer is an engineering problem. The trade off is that if the timer is long, the packet utilization is high, but the waiting time increases. If the timer is short, the waiting time is less, but the packet utilization is lower. The packet utilization is characterized by the mean amount of data transmitted in a packet.

The incoming traffic of the multiplexer is described by a continuous time batch Markovian arrival process with cardinality m [6]. Its $m \times m$ generator is denoted by D. The arrival process itself is characterized by a set of matrices D_i ($\sum_{i=0}^{K} D_i = D$), where $[D_k]_{i,j}$ corresponds to the arrival of k data units with a state transition from i to j.

The background Markov process and the counting process are denoted by $J(t)$, and $N(t)$. Then $N(t), J(t)$ together form a Markov chain with the following generator:

$$\boldsymbol{\mathcal{Q}} = \begin{bmatrix} D_0 & D_1 & D_2 & \dots \\ & D_0 & D_1 & \dots \\ & & D_0 & \dots \\ & & & \ddots \end{bmatrix}.$$

In the following Sections we denote the transient state transition matrix of this Markov chain by $P(k,t)$, using the following definition:

$$[P(k,t)]_{i,j} \;=\; Pr(N(t) = k,\ J(t) = j \mid N(0) = 0,\ J(0) = i). \qquad (15)$$

The BMAP is a strong modelling tool for traffic characterization in Markovian analysis. It can be constructed from measured traffic behavior [8, 5].

3 Queue Length Distribution at Packet Departures

The queueing system of Section 2 can be characterized by a discrete time Markov chain at packet departure instants. Embedding at departures usually leads to a so-called M/G/1 type structure, as it does in our case, too. The transition probability matrix builds up as follows:

$$\boldsymbol{\mathcal{X}} = \begin{bmatrix} \boxed{\qquad\qquad \boldsymbol{\mathcal{B}} \qquad\qquad} & \cdots \\ \boxed{\qquad\qquad \boldsymbol{\mathcal{A}} \qquad\qquad} & \cdots \\ \quad\boxed{\qquad\quad \boldsymbol{\mathcal{A}} \qquad\quad} & \cdots \\ \qquad\quad\boxed{\qquad \boldsymbol{\mathcal{A}} \qquad} & \cdots \\ \qquad\qquad\quad\boxed{\quad \boldsymbol{\mathcal{A}} \quad} & \cdots \\ \qquad\qquad\qquad\qquad\quad \ddots & \vdots \end{bmatrix} \tag{16}$$

Usually M/G/1 type matrices are defined by their quadratic matrix blocks. Now we define the matrix by its block rows, because it simplifies the definition. The states inside the $m \times m$ blocks are reflecting the state of the arrival process; a transition from block i to block j means that the queue size changed from i to j since the last departure instant.

Matrix row $\boldsymbol{\mathcal{A}}$ corresponds to case P1 of Section 2. In this case the inter departure time is exactly Δ, since the timer does not play a role. At the next embedded point the server will decrease the queue by L, so the queue size change equals to the arrivals during Δ minus L. Thus, $\boldsymbol{\mathcal{A}}$ is defined by:

$$\boldsymbol{\mathcal{A}} = \begin{bmatrix} P(0,\Delta) & P(1,\Delta) & P(2,\Delta) & P(3,\Delta) & \dots \\ & P(0,\Delta) & P(1,\Delta) & P(2,\Delta) & \dots \\ & & P(0,\Delta) & P(1,\Delta) & \dots \\ & & \ddots & \ddots & \vdots \\ & & & P(0,\Delta) & \dots \end{bmatrix}_{L\times m \text{ rows}}, \tag{17}$$

where $P(k,\Delta)$ is given by Eq. (15).

The definition of $\boldsymbol{\mathcal{B}}$ is more complex due to the effect of the timer. We further divide $\boldsymbol{\mathcal{B}}$, to distinguish between the completely idle (P2) and not completely idle (P3) cases:

$$\boldsymbol{\mathcal{B}} = \left[\begin{array}{c} \hat{\boldsymbol{\mathcal{B}}} \\ \hline \\ \tilde{\boldsymbol{\mathcal{B}}} \\ \\ \end{array}\right]. \tag{18}$$

The first block row (with height m) describes the transitions from the idle buffer (these matrices are denoted by $\hat{\boldsymbol{\mathcal{B}}}$, and correspond to P3 in Section 2), the other rows describe transitions from buffer levels 1 – L-1 (P2 in Section 2).

In both cases the timer starts (in case of $\hat{\boldsymbol{\mathcal{B}}}$ after the first arrival, if the batch size is less than L), and the next transition may happen later than Δ. The evolution of the queue size between levels 1 and L is important to capture the effect of the timer. Therefore we define the following two Markov chain generators. The continuous time Markov chain generator $\boldsymbol{Q}$ follows the queue size increase process between 1 and L, and $\boldsymbol{Z}(t)$ is the transition probability matrix of the buffer size increase process during time t (thus $\boldsymbol{Z}(t) = e^{\boldsymbol{Q}t}$):

$$\boldsymbol{Q} = \begin{bmatrix} D_0 & D_1 & \dots & D_{L-2} \\ & D_0 & \dots & D_{L-3} \\ & & \ddots & \vdots \\ & & & D_0 \end{bmatrix}, \qquad \boldsymbol{Z}(t) = \begin{bmatrix} P(0,t) & P(1,t) & \dots & P(L-2,t) \\ & P(0,t) & \dots & P(L-3,t) \\ & & \ddots & \vdots \\ & & & P(0,t) \end{bmatrix}. \tag{19}$$

$\boldsymbol{\Pi}(t)$ describes the behavior of the timer. $[\boldsymbol{\Pi}(t)]_{i,j}$ is the probability that starting with i ($1 \le i \le L$) data unit in the buffer the state of the system will be j just after the start of the next service. The next service can start when the necessary number of data units have arrived until time t (first term in Eq. (20)), or at time t the buffer content is served even if a full packet does not come out (second term of Eq. 20). This probability matrix is computed by:

$$\boldsymbol{\Pi}(t) = \int_0^t e^{\boldsymbol{Q}\tau} d\tau \cdot \begin{bmatrix} D_{L-1} & D_L & \dots & D_K & & & \\ D_{L-2} & D_{L-1} & \dots & D_{K-1} & D_K & & \\ \vdots & \vdots & \ddots & \dots & \dots & D_K & \\ D_1 & D_2 & \dots & \dots & \dots & \dots & D_K \end{bmatrix} + e^{\boldsymbol{Q}t} \cdot \begin{bmatrix} I_{m\times m} & 0 & \dots \\ I_{m\times m} & 0 & \dots \\ \vdots & \vdots & \\ I_{m\times m} & 0 & \dots \end{bmatrix}. \tag{20}$$

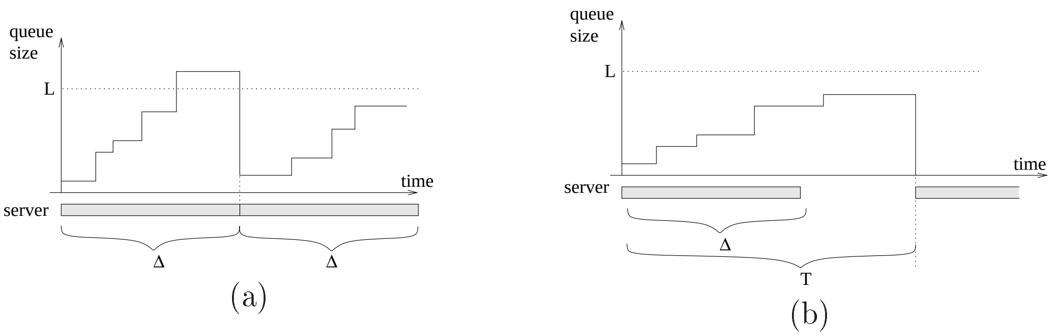


Fig. 1. Possible scenarios starting with buffer size 1 to L

If the buffer size is between 1 and $L-1$ ($\tilde{\boldsymbol{\mathcal{B}}}$), it has to be investigated how many data units arrive during the Δ interval (during which the server is busy). If the arriving data units increase the buffer above L, the service of a new packet starts just after finishing the previous one (this events are expressed by the first matrix term of Eq. 21, and are depicted on Figure 1 (a)). If the buffer level is still below L (second term of Eq. 21, shown in Figure 1 (b)), an additional delay follows, with a maximal length of $(T-\Delta)^+$, since the timer started at the moment when the buffer decreased below L. Thus:

$$\tilde{\boldsymbol{\mathcal{B}}} = \begin{bmatrix} P(L-1,\Delta) & P(L,\Delta) & \dots \\ P(L-2,\Delta) & P(L-1,\Delta) & \dots \\ \vdots & \vdots & \dots \\ P(1,\Delta) & P(2,\Delta) & \dots \end{bmatrix} + \boldsymbol{Z}(\Delta)\cdot\boldsymbol{\Pi}((T-\Delta)^+), \tag{21}$$

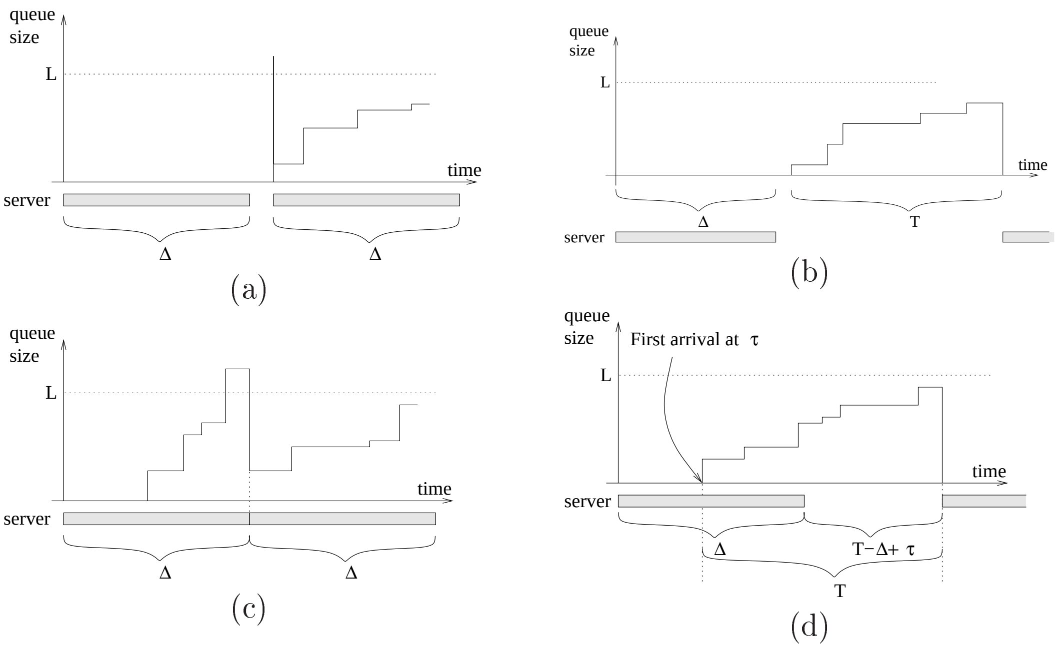


Fig. 2. Possible scenarios starting with empty buffer

If the buffer became empty at the beginning of the service of a packet ($\hat{\boldsymbol{\mathcal{B}}}$), we have two cases. It is possible that there are no arrivals during the server busy time (during Δ). In this case the next arrival can bring the queue above L, immediately causing a start of a new packet service (first term of Eq. 22, and Figure 2 (a)); or the queue remains below L, and a waiting period comes (waiting for more data units) for a maximal length of T, since the arrival into the empty queue initiated the timer (second term of Eq. 22, and Figure 2 (b)). The second case is when there were arrivals during the server busy time. In this case the arrivals can bring the queue above L, and the service of a new packet starts just after the end of service of the previous one (third term of Eq. 22, and Figure 2 (c)). The arrival can leave the system below L, and a waiting period is started with a maximal length of $(T-(\Delta-\tau))^+$, since the timer started at the first arrival time (τ) and from the timer $\Delta - \tau$ time already expired when the server becomes empty (fourth term of Eq. 22, and Figure 2 (d)). Thus we have:

$$\begin{aligned}\hat{\boldsymbol{\mathcal{B}}} &= P(0,\Delta)(-D_0)^{-1}\left[D_L\ D_{L+1}\ \ldots\ D_K\right] + \\ &+ P(0,\Delta)(-D_0)^{-1}\left[D_1\ D_2\ \ldots\ D_{L-1}\right]\cdot \boldsymbol{\Pi}(T) + \\ &+ \left[P(L,\Delta)\ P(L+1,\Delta)\ \ldots\right] + \\ &+ \int_0^{\Delta} e^{D_0\tau}\cdot\left[D_1\ D_2\ \ldots\ D_{L-1}\right]\cdot \boldsymbol{Z}(\Delta-\tau)\cdot\boldsymbol{\Pi}((T-\Delta+\tau)^+)d\tau.\end{aligned} \tag{22}$$

The steady state distribution of the embedded Markov chain (16) is partitioned according to the state partitioning in the following way:

$$\boldsymbol{x} = \Bigg[\underbrace{\left[p_0\ p_1\ \ldots\ p_{L-1}\right]}_{\boldsymbol{x}_0}\ \underbrace{\left[p_L\ p_{L+1}\ \ldots\ p_{2L-1}\right]}_{\boldsymbol{x}_1}\ \cdots\Bigg],$$

where the p_is are vectors of size m. In this section we defined the $\boldsymbol{\mathcal{A}}$ and $\boldsymbol{\mathcal{B}}$ blocks of $\boldsymbol{\mathcal{X}}$. The steady state probability vector that satisfies

$$\boldsymbol{x} = \boldsymbol{x}\,\boldsymbol{\mathcal{X}}$$

can be obtained by a matrix analytic method summarized in Appendix D.

4 Waiting Time Distribution

The waiting time distribution $P(W > w)$ is the probability that the waiting time of an arriving batch (measured from its arrival to its departure) exceeds a given threshold w. It can be easily computed if some parameters are kept fixed. These parameters are: the length of the buffer at the arrival (k), the remaining server occupation time (t_1), and the maximal departure time measured from the point when the the buffer descends below level L (t_2). If we know the particular values of these parameters, the waiting time distribution $P_W(k, t_1, t_2)$ can be computed by the following way:

$$P_W(k,t_1,t_2) = \begin{cases} h & \text{if } W^* < t_1 + \left\lceil \frac{k}{L}\right\rceil \Delta, \\ 0 & \text{if } k < L,\ \ W^* \geq t_1 + t_2 + \left\lceil \frac{k}{L}\right\rceil \Delta, \\ & \text{if } k \geq L,\ \ W^* \geq t_1 + (T-\Delta)^+ + \left\lceil \frac{k}{L}\right\rceil \Delta, \\ \left[e^{\boldsymbol{Q}(W^*-\Delta)}h\right]_{\{k/L\}} & \text{otherwise,} \end{cases} \tag{23}$$

where h is a vector of ones with size m.

The first item corresponds to the case when the server occupancy time plus the service time of packets in the queue exceeds the waiting time requirement. In this case $P(W > w)$ equals to one.

The second item covers the case when the waiting time requirement is surely satisfied. This happens if the waiting time requirement is larger than the server occupancy time, plus the service time of the packets in the queue, plus the maximal possible delay caused by the timer. The latter quantity is t_2 if the buffer size is less than L (this is the definition of t_2). It is $(T - \Delta)^+$ if $k \geq L$, because the server will be occupied when the last segment gets below L in the buffer.

The third item means that the waiting time exceeds w if the L long block in the queue, which k belongs to, is still not filled up until $w - \Delta$. $[e^{\boldsymbol{Q}t}]_i$ is the probability that the arrival process did not generate enough arrivals to leave this block until time t, if there were i data units in the block at the beginning. $\{k/L\}$ (where $\{\}$ denotes the remainder of the division) is the buffer position inside the L long block after the arrival.

To obtain the waiting time distribution, we have to multiply $P_W(k, t_1, t_2)$ by the probability of the given k, t_1 and t_2 parameters.

Since the resulting expression is long, we split it to 3 parts to make it easier to follow. This splitting is according to the partitioning of the P1,P2,P3 cases in Section 2. In each part we produce the waiting time properties multiplied by their weights (q_i), and the sum of the weights used at the end to normalize the results (q_{n_i}).

P1 Last departure left the system above L ($i > L$, with probability p_i). Let assume a tagged packet arrival at time t (measured from the last service instant). The length of the queue will be the sum of the queue size at the last departure (i), the number of data units arrived before the tagged one (j), and the size of the arrived batch (k). The server is busy for at least $\Delta - t$, and the last fragment of the data waits at most $T - \Delta$ after the server becomes idle. In this case:

$$q_1 = \sum_{i=L}^{\infty} p_i \int_0^{\Delta} \sum_{j=0}^{\infty} P(j,t) \sum_{k=1}^{\infty} D_k \cdot P_W(i+j+k, \Delta - t, (T-\Delta)^+) dt \quad (24)$$

This expression considers the arrivals during Δ, thus the weight is the following (which can also be obtained from Eq. (24) by setting $P_W(i, t_1, t_2) = 1$):

$$q_{n_1} = \sum_{i=L}^{\infty} p_i \int_0^{\Delta} e^{Dt} dt D_A h$$

P2 The last departure left the system between 1 and $L - 1$. Waiting time of arrivals during Δ can be calculated similarly to the previous case, but the queue can be still below L after Δ. Then the server becomes idle (reflected by the second term of Eq. 25), and the timer runs for at most $T - \Delta$ (since it has been started at the beginning of the last service). Thus we have:

$$q_2 = \sum_{i=1}^{L-1} p_i \Bigg[\int_0^{\Delta} \sum_{j=0}^{\infty} P(j,t) \sum_{k=1}^{\infty} D_k \cdot P_W(i+j+k, \Delta - t, (T-\Delta)^+) dt +$$

$$+ \sum_{\ell=0}^{L-i-1} P(\ell,\Delta) \int_0^{(T-\Delta)^+} \sum_{j=0}^{L-\ell-i-1} P(j,t) \sum_{k=1}^{\infty} D_k \cdot P_W(i+\ell+j+k, 0, T-\Delta-t) dt \Bigg] \tag{25}$$

Now we covered the arrivals during Δ, plus the arrivals until the packet was filled up totally or the timer elapsed. Thus q_{n_2} is:

$$q_{n_2} = \sum_{i=1}^{L-1} p_i \int_0^{\Delta} e^{Dt} dt D_A h + \sum_{i=1}^{L-1} p_i \sum_{\ell=0}^{L-i-1} P(\ell,\Delta) \sum_{j=0}^{L-\ell-i-1} \int_0^{T-\Delta} P(j,t) dt D_A h$$

P3 The last departure left the system empty. For Δ the server is still busy. We consider 2 subcases according to the time of the first arrival, that can happen before or after Δ. In both cases the timer is started at the first arrival.

· The first arrival occurs during Δ. The waiting time probability consists of 3 parts. The first corresponds to the first arrival. The second computes the other remaining arrivals in Δ (conditioned on the time point of the first arrival), and the third term considers the case when the buffer is still below L after Δ.

$$q_{3a} = p_0 \Bigg[\int_0^{\Delta} e^{D_0 t} \sum_{k=1}^{\infty} D_k P_W(k, \Delta - t, (T_{CU} - \Delta + t)^+) dt +$$

$$+ \int_0^{\Delta} e^{D_0 t} \sum_{\ell=1}^{\infty} D_\ell \cdot$$

$$\cdot \int_0^{\Delta - t} \sum_{j=0}^{\infty} P(j,s) \sum_{k=1}^{\infty} D_k P_W(\ell + j + k, \Delta - t - s, (T - \Delta + t + s)^+) ds\ dt +$$

$$+ \int_0^{\Delta} e^{D_0 t} \sum_{\ell=1}^{L-1} D_\ell \sum_{i=0}^{L-\ell-1} P(i, \Delta - t) \cdot$$

$$\cdot \int_0^{(T-\Delta+t)^+} \sum_{j=0}^{L-\ell-i-1} P(j,s) \sum_{k=1}^{\infty} D_k P_W(\ell + i + j + k, 0, T - \Delta + t - s) ds\ dt \Bigg] \tag{26}$$

The weight of this case consists of the arrivals during Δ and – if the packet is still not ready after Δ – the arrivals till the packet is filled up or the timer elapses:

$$q_{n_{3a}} = p_0 \int_0^{\Delta} e^{Dt} dt D_A h + \int_0^{\Delta} e^{D_0 t} \sum_{k=1}^{L-1} D_k \sum_{\ell=0}^{L-k-1} P(\ell, \Delta - t) \cdot \\ \cdot \sum_{j=0}^{L-\ell-k-1} \int\limits_0^{(T-\Delta+t)^+} P(j,s) ds \; dt \; D_A h \tag{27}$$

· After Δ the buffer is still empty. The first term is related to the waiting time of the first arrival. If the batch size of the first arrival is less than L, other arrivals may come at time t in the $(0,T)$ interval (second term), experiencing $T-t$ remaining timer value. Thus, q_{3b} and $q_{n_{3b}}$ are:

$$q_{3b} = p_0 e^{D_0\Delta}(-D_0)^{-1} \sum_{k=1}^{\infty} D_k P_W(k,0,T) + \\ + p_0 e^{D_0\Delta}(-D_0)^{-1} \sum_{\ell=1}^{L-1} D_\ell \int\limits_0^T \sum_{j=0}^{L-\ell-1} P(j,t) \sum_{k=1}^{\infty} D_k P_W(\ell+j+k, 0, T-t) dt, \tag{28}$$

and

$$q_{n_{3b}} = p_0 e^{D_0\Delta}(-D_0)^{-1} D_A h + p_0 e^{D_0\Delta}(-D_0)^{-1} \sum_{\ell=1}^{L-1} D_\ell \sum_{j=0}^{L-\ell-1} \int\limits_0^T P(j,t) dt D_A h. \tag{29}$$

Finally the waiting time distribution is computed by:

$$P(W > w) = \frac{q_1 + q_2 + q_{3a} + q_{3b}}{q_{n_1} + q_{n_2} + q_{n_{3a}} + q_{n_{3b}}}$$

The ith moment of the waiting time can be obtained similarly since knowing the same three parameters (k, t_1, t_2) the waiting time moments are easy to compute.

5 The Analysis Algorithm

Putting together the results of Sections 3 and 4, the algorithm works as follows:

(i) Construct the BMAP description of the traffic source.
If there are many different traffic sources, each has to be modelled by individual BMAP, and aggregated together by Kronecker operations. For

example, if $D_k^{(i)}$ are the BMAP matrices of traffic source i and we have P independent BMAPs, the aggregated BMAP is constructed as:

$$
\begin{aligned}
&D_0 = D_0^{(1)} \oplus D_0^{(2)} \oplus \cdots \oplus D_0^{(P)} \\
&D_1 = D_1^{(1)} \otimes I \otimes \cdots \otimes I + I \otimes D_1^{(2)} \otimes \cdots \otimes I + \cdots + I \otimes I \otimes \cdots \otimes D_1^{(P)} \\
&D_2 = D_2^{(1)} \otimes I \otimes \cdots \otimes I + I \otimes D_2^{(2)} \otimes \cdots \otimes I + \cdots + I \otimes I \otimes \cdots \otimes D_2^{(P)} \\
&\quad + D_1^{(1)} \otimes D_1^{(2)} \otimes I \otimes \cdots \otimes I + D_1^{(1)} \otimes I \otimes D_1^{(3)} \otimes \cdots \otimes I + \ldots \\
&\quad + I \otimes I \otimes \cdots \otimes D_1^{(P-1)} \otimes D_1^{(P)}
\end{aligned}
$$

... and so on.

(ii) Construct $\boldsymbol{\mathcal{A}}$ and $\boldsymbol{\mathcal{B}}$ matrices according to Section 3. Numerical problems might arise during this step. To overcome these, we used the following procedure.
   - The computation of $P(k,t)$ matrices is according to Appendix C. The number of computed $P(k,\Delta)$ matrices determine the size of $\boldsymbol{\mathcal{A}}$ and $\boldsymbol{\mathcal{B}}$.
   - In the computation of $\boldsymbol{\Pi}(t)$ the two quantities, $\int_0^t e^{\boldsymbol{Q}x}dx$ and $e^{\boldsymbol{Q}t}$, can be efficiently computed together using randomization (see Appendix B).
   - In $\hat{\boldsymbol{\mathcal{B}}}$ the integral can be evaluated by the trapezoid rule or by the Simpson rule. The $e^{D_0 t}$ matrix is calculated by randomization again.

(iii) Solve the steady state of the obtained M/G/1 type Markov chain. We used a method presented in [7], summarized in Appendix D. The computation of the steady state probabilities is stopped when they get small (e.g., $p_i < 10^{-6}$).

(iv) Compute the waiting time distribution according to Appendix A. All arising numerical issues are already mentioned. We note that the matrices of which the matrix exponential is computed are either small (of size $m \times m$) or do not depend on the integral variable, so instead of numerical integration they can be computed by randomization. The only exception is q_{3a}, where the numerical integration is unavoidable.

6 Numerical Results

We implemented the above discussed computation method in MATLAB, and also developed a simulation tool in OMNeT++ ([2]) to check the correctness of both the expressions and the MATLAB implementation. The figures show both the MATLAB (with lines) and the simulation results (indicated with points).

We evaluated the following example. There are N traffic sources with on-off behavior. The on and the off periods are exponentially distributed, with parameters α and β, respectively. During the on period each traffic source generates traffic with exponentially distributed inter arrival times, with intensity λ. At each arrival instant the source generates k data units with probability $p(1-p)^{L-k}/(1-(1-p)^L)$, which is a geometrical distribution with parameter p truncated at L. The multiplexer assembles packages containing a maximum of L data units.

These parameters have the following values through the examples:

$L =$	8
$\alpha =$	1/1000
$\beta =$	1/1500
$\lambda =$	50
$N =$	5
$p =$	(changing)
$\Delta =$	(changing)

Figure 3 shows the distribution of the waiting time with different timer values. We used parameters $p = 0.8$ and $\Delta = 2$ in this example. To magnify the effect of timer, the parameters have been chosen to give low load. The figure verifies the assumption that the higher the timer value is, the bigger is the waiting time. Furthermore, two jumps can be observed in the curves,

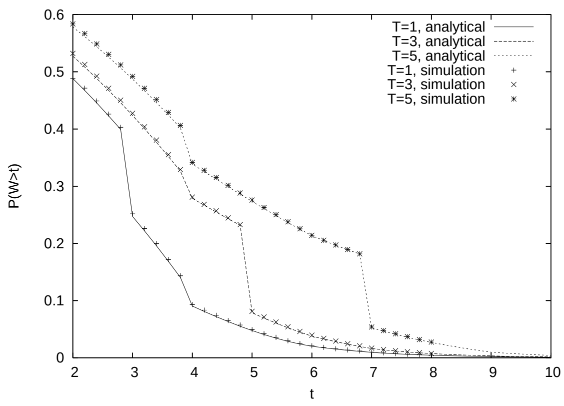
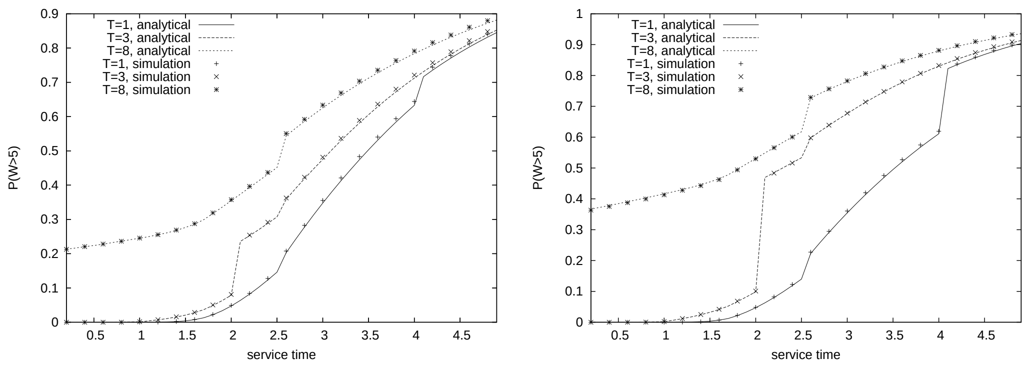


Fig. 3. Distribution of the waiting time

one at 2Δ, the one at $T + \Delta$. The explanation of the jumps is the following. Let assume there is a given amount of data ($< L$) waiting in the queue, and the server is idle (timer is running). With a newly arrived batch (the tagged arrival) the queue size exceeds L, so a packet is compiled and the service starts. The second fragment of the tagged arrival (that has not been included in the packet) waits exactly Δ if enough arrivals are coming during the server occupancy time, or it waits exactly T, if there are not enough arrivals in T. These two cases (increased by the time spent in the server) provide the jumps on the distribution function. The jump at $T + \Delta$ also contains the probability that a batch arrives into an empty system, and there were not enough arrivals until the timer elapsed.

In our second example the service time is varying between 0 and 5, and we examine the probability of exceeding the $w = 5$ waiting time. Figure 4 (a) and (b) correspond to the $p = 0.4$ and to the $p = 0.8$ setting. There are



(a) (b)

Fig. 4. $P(W > 5)$ vs. service time

jumps again at $\Delta = w/2 = 2.5$ and at $\Delta = w - T$. The explanation is the same as in the first example, the first jump is related to the jump at 2Δ on the distribution function, and the second one is related to the one at $T + \Delta$. It is an interesting feature of the system that the $P(W > w)$ probability does not reduce to zero by decreasing the service time to zero (i.e., increasing the output capacity to infinity) when $T > w$ as it is in Figure 4 with $T = 8$.

In all the examples the computation of one point of the waiting time distribution with our MATLAB implementation took few seconds on a Pentium4 2.4 GHz machine, while the simulation could give acceptable results only after about 1 minute. The only exception is the last example, where the execution took a longer time (1-2 minutes) at $N = 15$. Of course, with a C implementation the speed of our analytical algorithm can be increased substantially.

7 Conclusion

In this paper we provided a model and an algorithm to compute the the waiting time distribution of BMAP/D/1-Timer multiplexer queues. The proposed algorithm is designed for an efficient numerical implementation.

The last section presents a numerical example. The analytical results showed a rather perfect fit with the simulation results. In this example we also investigated the special features of the waiting time distribution of such queues.

References

[1] *ITU-T Recommendation I.363.2, B-ISDN ATM Adaptation Layer Type 2 Specification, Toronto, 1997.*

[2] *OMNeT++ Discrete Event Simulation System, http://www.omnetpp.org.*

[3] Bini, D. and B. Meini, *On the Solution of a Nonlinear Matrix Equation arising in Queueing Problems*, SIAM Journal on Matrix Analysis and Applications **17** (1996), pp. 906–926.

[4] Gross, D. and D. Miller, *The randomization technique as a modeling tool and solution procedure for transient Markov processes*, Operations Research **32** (1984), pp. 343–361.

[5] Horváth, A., G. I. Rózsa and M. Telek, *A map fitting method to approximate real traffic behaviour*, in: *8th IFIP Workshop on Performance Modelling and Evaluation of ATM & IP Networks*, Ilkley, England, 2000, pp. 32/1–12.

[6] Latouche, G. and V. Ramaswami, "Introduction to Matrix Analytic Methods in Stochastic Modeling," ASA-SIAM, 1999.

[7] Neuts, M. F., "Structured Stochastic Matrices of M/G/1 Type and their Applications," Dekker, 1989.

[8] Ryden, T., *An em algorithm for estimation in Markov modulated Poisson processes*, Computational statist. and data analysis **21** (1996), pp. 431–447.

[9] Saito, H., *Performance evaluation of AAL2 voice multiplexer*, Performance Evaluation **42** (2000), pp. 57–83.

A Numerical method for the waiting time distribution

Expressions Eq. (24), (25), (26) and (28) are difficult to compute efficiently, because numerical integration is usually slow and inaccurate. Fortunately most of the numerical integrations can be eliminated. Substituting Eq. (23) into the above mentioned expressions, most of the integrals will take the form $\int_a^b e^{\boldsymbol{Q}t}dt$, which can be efficiently computed by randomization (see Appendix B). In favor of completeness, here we give q_1, q_2, q_{3a} and q_{3b} with all possible numerical integrations eliminated.

Substituting Eq. (23) into Eq. (24), we get:

$$q_1 = \sum_{i=L}^{\infty} p_i \sum_{j=0}^{\infty} \sum_{k=1}^{\infty} \Big(\int_0^{s_1} P(j,t)dtD_k h + \int_{s_1}^{s_2} P(j,t)dtD_k \left[e^{\boldsymbol{Q}(w-\Delta)}h\right]_{\{\frac{i+j+k}{L}\}} \Big). \tag{30}$$

The limits of the integral s_1 and s_2 are derived from the definition of $P_W(k,t_1,t_2)$ (see Eq. (23)); below s_1 $P_W(\bullet)$ equals to 1, and above s_2 it equals to zero, thus:

$$\begin{aligned} s_1 &= \min\{\Delta, \Big(\Delta + \lceil\frac{i+j+k}{L}\rceil\Delta - w\Big)^+\} \\ s_2 &= \min\{\Delta, \Big(\Delta + \lceil\frac{i+j+k}{L}\rceil\Delta + (T_{CU}-\Delta)^+ - w\Big)^+\} \end{aligned} \tag{31}$$

Similarly, Eq. (25) can be transformed to the following:

$$\begin{aligned} q_2 = &\sum_{i=1}^{L-1} p_i \sum_{j=0}^{\infty} \sum_{k=1}^{\infty} \Big(\int_0^{s_1} P(j,t)dtD_k h + \int_{s_1}^{s_2} P(j,t)dtD_k \left[e^{\boldsymbol{Q}(w-\Delta)}h\right]_{\{\frac{i+j+k}{L}\}} \Big) + \\ &\sum_{i=1}^{L-1} p_i \sum_{l=0}^{L-i-1} P(l,\Delta) \sum_{\ell=0}^{L-i-1} \Bigg[\sum_{k=1}^{L-i-\ell-j-1} \int_0^{((T-\Delta)^+-w+\Delta)^+} P(j,t)dtD_k \left[e^{\boldsymbol{Q}(w-\Delta)}h\right]_{\{\frac{i+j+k+\ell}{L}\}} + \\ &\sum_{k=L-i-\ell-j}^{\infty} \Bigg(I_{\{w<\lceil\frac{i+j+k+\ell}{L}\rceil\Delta\}} \cdot \int_0^{(T-\Delta)^+} P(j,t)dtD_k h + \\ &I_{\{\lceil\frac{i+j+k+\ell}{L}\rceil\Delta\le w<\lceil\frac{i+j+k+l}{L}\rceil\Delta+(T-\Delta)^+\}} \cdot \int_0^{(T-\Delta)^+} P(j,t)dtD_k \left[e^{\boldsymbol{Q}(w-\Delta)}h\right]_{\{\frac{i+j+k+l}{L}\}} \Bigg) \Bigg]. \end{aligned} \tag{32}$$

Where the integral limits s_1 and s_2 are already defined by Eq. (31).

q_{3a} can be transformed to the following – much longer but numerically better computable – expression:

$$
\begin{aligned}
q_{3a} = p_0 \sum_{k=1}^{L-1} \Bigg(& \int_0^{(2\Delta-w)^+} e^{D_0 t} dt D_k h + \\
& + I_{\{w<T+\Delta\}} \int_{\max((2\Delta-w)^+,(\Delta-T)^+)}^{\Delta} e^{D_0 t} dt D_k \left[e^{\boldsymbol{Q}(w-\Delta)} h \right]_k \Bigg) + \\
+ p_0 \sum_{k=L}^{\infty} & \left(\int_0^{s_1} e^{D_0 t} dt D_k h + \int_{s_1}^{s_2} e^{D_0 t} dt D_k \left[e^{\boldsymbol{Q}(w-\Delta)} h \right]_{\{\frac{k}{L}\}} \right) + \\
+ p_0 \int_0^{\Delta} & e^{D_0 t} dt \sum_{\ell=1}^{\infty} D_\ell \sum_{j=0}^{\infty} \sum_{k=1}^{\infty} \Bigg[I_{\{\ell+j+k<L\}} \Bigg(\int_0^{(2\Delta-w-t)^+} P(j,s) ds D_k h + \\
& + \Big(\int_{(2\Delta-w-t)^+}^{(\Delta-t-T)^+} P(j,s) ds + \\
& + I_{\{w<T+\Delta\}} \int_{(\Delta-t-T)^+}^{\Delta-t} P(j,s) ds \Big) D_k \left[e^{\boldsymbol{Q}(w-\Delta)} h \right]_{k+\ell+j} \Bigg) + \\
& + I_{\{\ell+j+k\geq L\}} \left(\int_0^{s_1-t} P(j,s) ds D_k h + \int_{s_1-t}^{s_2-t} P(j,s) ds D_k \left[e^{\boldsymbol{Q}(w-\Delta)} h \right]_{\{\frac{l+j+k}{L}\}} \right) \Bigg] dt \\
+ p_0 \int_0^{\Delta} & e^{D_0 t} \sum_{\ell=1}^{L-1} D_\ell \sum_{i=0}^{L-\ell-1} P(i,\Delta-t) \sum_{j=0}^{L-\ell-i-1} \Bigg[\\
& \sum_{k=1}^{L-\ell-i-j-1} \int_0^{(T+t-w)^+} P(j,s) ds \left[e^{\boldsymbol{Q}(w-\Delta)} h \right]_{k+l+i+j} + \\
& + \sum_{k=L-\ell-i-j}^{\infty} \Big(I_{\{w<\lceil \frac{\ell+i+k+\ell}{L} \rceil \Delta\}} \int_0^{(T-\Delta+t)^+} P(j,s) ds D_k h + \\
& + I_{\{\lceil \frac{\ell+i+k+\ell}{L} \rceil \Delta \leq w < \lceil \frac{\ell+i+k+\ell}{L} \rceil \Delta + (T-\Delta)^+\}} \int_0^{(T-\Delta+t)^+} P(j,s) ds \, D_k \left[e^{\boldsymbol{Q}(w-\Delta)} h \right]_{\{\frac{k+i+j+\ell}{L}\}} \Bigg] dt.
\end{aligned}
\tag{33}
$$

$q_{n_{3a}}$ can also be rewritten into a numerically more efficient form. First we rewrite Eq. (27) in matrix form:

$$
q_{n_{3a}} = p_0 \int_0^{\Delta} e^{Dt} dt D_A h + \int_0^{\Delta} e^{D_0 t} \left[D_1 \dots D_{L-1} \right] e^{\boldsymbol{Q}(\Delta-l)} \cdot \int_0^{(T-\Delta+t)^+} e^{\boldsymbol{Q}s} ds \, dt \begin{bmatrix} D_A h \\ \vdots \\ D_A h \end{bmatrix}. \tag{34}
$$

With some algebra it simplifies to:

$$q_{n_{3a}} = p_0 \Bigg(\int_0^{\Delta} e^{Dt} dt D_A h + \Bigg[[P(1,\Delta)\ P(2,\Delta)\ \ldots\ P(L-1,\Delta)] -$$

$$- \int_0^{\Delta} e^{D_0 t} dt \left[D_1 \ \ldots \ D_{L-1} \right] e^{\boldsymbol{Q}T} \Bigg] (-\boldsymbol{Q})^{-1} \begin{bmatrix} D_A h \\ \vdots \\ D_A h \end{bmatrix} \Bigg),$$

which does not contain any numerical integral. For q_{3b} we have the following expression:

$$q_{3b} = p_0 e^{D_0 \Delta} (-D_0)^{-1} \Bigg[\sum_{k=1}^{L-1} I_{\{w<T+\Delta\}} \cdot D_k \left[e^{\boldsymbol{Q}(w-\Delta)} h \right]_k + \sum_{k=L}^{\infty} \Big(I_{\left\{ w < \lceil \frac{k}{L} \rceil \Delta \right\}} \cdot D_k h +$$

$$+ I_{\left\{ \lceil \frac{k}{L} \rceil \Delta < w < \lceil \frac{k}{L} \rceil \Delta + (T-\Delta)^+ \right\}} \cdot D_k \left[e^{\boldsymbol{Q}(w-\Delta)} h \right]_{\left\{ \frac{k}{L} \right\}} \Big) +$$

$$+ \sum_{\ell=1}^{L-1} D_l \sum_{j=0}^{L-\ell-1} \Big[\sum_{k=1}^{L-\ell-j-1} \int\limits_0^{(T+\Delta-w)^+} P(j,t) dt D_k \left[e^{\boldsymbol{Q}(w-\Delta)} h \right]_{\ell+j+k} +$$

$$+ \sum_{k=L-\ell-j}^{\infty} \Big(I_{\left\{ w < \lceil \frac{\ell+j+k}{L} \rceil \Delta \right\}} \cdot \int_0^T P(j,t) dt D_k h +$$

$$+ I_{\left\{ \lceil \frac{\ell+j+k}{L} \rceil \Delta < w < \lceil \frac{\ell+j+k}{L} \rceil \Delta + (T-\Delta)^+ \right\}} \cdot \int_0^T P(j,t) dt D_k \left[e^{\boldsymbol{Q}(w-\Delta)} h \right]_{\left\{ \frac{\ell+j+k}{L} \right\}} \Big) \Big] \Bigg].$$

B Randomization Algorithm

The randomization algorithm is an efficient and numerically stable method to compute $\boldsymbol{P}(t) = e^{\boldsymbol{Q}t}$ and $\boldsymbol{L}(t) = \int_0^t e^{\boldsymbol{Q}\tau} d\tau$ (see [4]). First we have to convert our continuous time generator $\boldsymbol{Q}$ to a stochastic matrix $\boldsymbol{Q}^*$:

$$q = \max_i |q_{ii}|$$

$$\boldsymbol{Q}^* = \boldsymbol{Q}/q + \boldsymbol{I}$$

Then, the following expressions provide the two quantities we need:

$$\boldsymbol{P}(t) = \sum_{i=0}^{\infty} \boldsymbol{Q}^{*i} e^{-qt} \frac{(qt)^i}{i!}$$

$$\boldsymbol{L}(t) = \frac{1}{q} \sum_{i=0}^{\infty} \boldsymbol{Q}^{*i} \sum_{i+1}^{\infty} e^{-qt} \frac{(qt)^j}{j!}$$

These can be computed together in the same iterative algorithm. With large $\boldsymbol{Q}$ matrices the most expensive step is the matrix multiplication (to calculate the ith power of $\boldsymbol{Q}$ from its $i-1$th), has to be computed only once. Then, weighting $\boldsymbol{Q}^{*i}$ with appropriate poisson coefficients gives $\boldsymbol{P}(t)$, and $\boldsymbol{L}(t)$.

If $e^{\boldsymbol{Q}t}$ is not needed, only $a \cdot e^{\boldsymbol{Q}t}$ or $e^{\boldsymbol{Q}t} \cdot b$ has to be computed (with a

and b being vectors), then the numerical solution is much faster, because the matrix-matrix multiplications are replaced with vector-matrix multiplications.

A further advantage of this algorithm is that its error can be bounded in advance, since $\boldsymbol{Q}^{*i}$ is stochastic. If the summation runs up to N, the error is less than $1 - \sum_{i=0}^{N} e^{-qt} \frac{(qt)^i}{i!}$.

C Computation of $P(k,t)$ matrices

The infinite matrix $\boldsymbol{\mathcal{Q}}$ (in Section 2) is the generator of the arrival process. Thus, its transition probability matrix at time t is:

$$\begin{bmatrix} P(0,t) & P(1,t) & P(2,t) & \dots \\ & P(0,t) & P(1,t) & \dots \\ & & P(0,t) & \dots \\ & & & \ddots \end{bmatrix} = e^{\boldsymbol{\mathcal{Q}}t}.$$

Multiplying both sides by $[\ I_{m\times m}\ \ 0\ \ 0\ \ \dots\ \ 0\ ]$ we get:

$$[\ P(0,t)\ \ P(1,t)\ \ \dots\ ] = [\ I_{m\times m}\ \ 0\ \ 0\ \ \dots\ ] \cdot e^{\boldsymbol{\mathcal{Q}}t}$$

These infinite vectors and matrices have to be cut at index K, which should be chosen to cover "most" arrivals during t. The block vector randomization method (see Appendix B) can be used to calculate $P(k,t)$. Alternatively, it is possible to use the following randomization based recursive relation:

$$P^{(n)}(k,t) = \sum_{i=0}^{k} P^{(n-1)}(i,t) \cdot D'_{k-i} \cdot e^{-qt} \frac{(qt)^i}{i!},$$
$$P^{(0)}(0,t) = I_{m\times m} \cdot e^{-qt}, \quad P^{(0)}(k,t) = 0_{m\times m} \text{ if } k > 0,$$

where $q = \max_{i\in S} [D_0]_{i,i}$, $D'_0 = D_0/q + I$ and $D'_k = D_k/q,\ k > 0$. The further n is increased, the more exact result we get: as written in Appendix B, the error is less than $1 - \sum_{i=0}^{n} e^{-qt} \frac{(qt)^i}{i!}$.

D Buffer Length Distribution Embedded at PDU Departure Instants

For this problem we use a numerical method presented in [7]. Here we give a short summary about this method, without going into details. Vectors $\boldsymbol{x}_i$ are computed by the following recursion, if $i \geq 1$:

$$\boldsymbol{x}_i = \left(\boldsymbol{x}_0 \widehat{\boldsymbol{B}}_i + \sum_{j=1}^{i-1} \boldsymbol{x}_j \widehat{\boldsymbol{A}}_{i+1-j} \right) \left(\boldsymbol{I} - \widehat{\boldsymbol{A}}_1 \right)^{-1} \tag{35}$$

This recursion is commonly referred as Ramaswami formula, which is numerically stable because it includes only additions and multiplications. The iteration can be stopped if the elements of the result vector are very close to zero. Matrices $\widehat{\boldsymbol{A}}_i$ and $\widehat{\boldsymbol{B}}_i$ are obtained by the following backward recursion, using the fundamental matrix G:

$$\widehat{\boldsymbol{B}}_k = \boldsymbol{B}_k + \widehat{\boldsymbol{B}}_{k+1}\boldsymbol{G} \tag{36}$$

$$\widehat{\boldsymbol{A}}_k = \boldsymbol{A}_k + \widehat{\boldsymbol{A}}_{k+1}\boldsymbol{G} \tag{37}$$

The recursion starts at the index where $\widehat{\boldsymbol{B}}_K = 0$ and $\widehat{\boldsymbol{A}}_K = 0$ ($\boldsymbol{B}_{K-1}$ or $\boldsymbol{A}_{K-1}$ are the last non zero matrices).

The computation of matrix $\boldsymbol{G}$ (also known as fundamental matrix) is the most critical point in the solution of M/G/1 type Markov chains. We suggest the following simple iteration to obtain it, but we also note that when the utilization approaches 1, there are much faster (but more complex) methods (see [3]):

$$\boldsymbol{G}_0 = \boldsymbol{0}$$

$$\boldsymbol{G}_{k+1} = (\boldsymbol{I} - \boldsymbol{A}_1)^{-1} \sum_{\nu=0,\nu\neq 1}^{\infty} \boldsymbol{A}_\nu \boldsymbol{G}_k^\nu$$

This iteration can be stopped when the sum of the elements in each row of $\boldsymbol{G}$ are close to 1.

It only remains to compute $\boldsymbol{x}_0$:

$$\boldsymbol{x}_0 = (\boldsymbol{\kappa}\tilde{\boldsymbol{\kappa}})^{-1}\boldsymbol{\kappa}.$$

Where $\boldsymbol{\kappa}$ is the solution of $\boldsymbol{\kappa} = \boldsymbol{\kappa}\boldsymbol{K}$, with $\boldsymbol{K}$ defined by:

$$\boldsymbol{K} = \sum_{\nu=0}^{\infty} \boldsymbol{B}_\nu \boldsymbol{G}^\nu.$$

And $\tilde{\boldsymbol{\kappa}}$ is computed by:

$$\boldsymbol{A}^* = \sum_{\nu=0}^{\infty} \boldsymbol{A}_\nu \sum_{k=0}^{\nu-1} \boldsymbol{G}^k$$

$$\boldsymbol{B}^* = \sum_{\nu=0}^{\infty} \boldsymbol{B}_\nu \sum_{k=0}^{\nu-1} \boldsymbol{G}^k$$

$$\tilde{\boldsymbol{\kappa}} = \boldsymbol{h} + \boldsymbol{B}^*(\boldsymbol{I} - \boldsymbol{A}^*)^{-1}\boldsymbol{h}.$$

Performance Evaluation of a Real-time Simulation Architecture using Probabilistic Model Checking

Nil Geisweiller [1] Jeremie Bonte

DTIM, Office National d'Études et de Recherches Aérospatiales (ONERA), Toulouse, France

Abstract

The goal of this paper is to show how to use probabilistic model checking techniques in order to achieve quantitative performance evaluation of a real-time distributed simulation. A simulation based on the *High Level Architecture* (HLA) is modelled as a stochastic process, a *Continuous Time Markov Chain* (CTMC), using the stochastic algebra PEPA. Next a property representing a performance constraint is evaluated applying *Continuous Stochastic Logic* CSL formula on the CTMC model using the probabilistic model checker PRISM. Finally a first experiment is made to compare the model with a real case.

Keywords: real-time distributed simulation, probabilistic model checking, performance evaluation, PEPA, PRISM.

1 Introduction

Real-time simulations are under heavy time constraints. The particularity of a real-time simulation is that the logical time of the simulation must be the same as the real one. To keep the real and logical time equal, the computation of the simulation must be fast enough. In order to get enough computation power, a simulation used to be split into several smaller simulations distributed across different machines. Unfortunately the smaller simulations lose the global view of the system and need to collect enough data from each-other's simulations. A middleware, *High Level Architecture* (HLA), is devoted to realising these exchanges of data carried by a network. The problem here is that the network and the middleware induces some delay in their communications which can hamper the realism of the simulation. In

[1] Email: `nil.geisweiller@cert.fr`

This is the workshop version of the paper. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: `www.elsevier.nl/locate/entcs`

order to know if a distributed simulation is going to behave correctly, certain time constraints need to be checked.

The aim of this paper is to show how a distributed simulation can be modelled as a probabilistic process in order to verify, using probabilistic model checking techniques, whether or not some required constraints are satisfied by the system. These constraints cannot always hold due to the intrinsic nature of the system, but can however be expected to hold in a majority of execution cases with reasonable confidence. The probabilistic modelling permits the quantification of this confidence and allows the representation of complex behaviours in a simpler way. For example, in the modelling of a network it is not necessary to model how a message gets lost but only the probability that the message gets lost.

To achieve this purpose, a HLA distributed simulation is modelled in a stochastic process, a *Continuous Time Markov Chain* (CTMC). Then, the *Continuous Stochastic Logic* (CSL) is used to formalise performance constraints required by the simulation to behave correctly. The probabilistic model checker PRISM allows the construction of the CTMC model and the verification that a CSL formula is satisfied by the CTMC model.

The remainder of this paper is organised as follows. Section 2 recalls the preliminary notions needed in the course of the paper : *High Level Architecture* (HLA), *Continuous Time Markov Chain* (CTMC) and the stochastic algebra PEPA *(Performance Evaluation Process Algebra)*, *Continuous Stochastic Logic* (CSL) and the software PRISM. Section 3 describes a case study of a real-time distributed simulation based on HLA, explains in detail the CTMC modelling using PEPA. Section 4 contains a time constraint formalised in CSL needed to be verified by the model and section 5 the obtained results with the model checker PRISM. Section 6 contains comparison with the model and a real simulation case. Finally, Section 7 gives some conclusions and future directions of the work.

2 Fundamental concepts

2.1 The HLA architecture

The High Level Architecture (HLA) provides the specification of a common technical architecture to combine multiple simulations into a larger simulation. It provides the structural basis for simulation interoperability. The combined simulation created is called a *federation* and each simulation that is combined to form a federation is called a *federate*. All federates are interconnected through a software called *Runtime Infrastructure* (RTI). The RTI is in charge of all communications through the network, which can be a local (LAN) or wide (WAN) like the Internet. The RTI has to keep the coherence of the simulation. It has two main functions, the object management and the time management; however in the case of real-time simulation, time management is not used. A quick description of the object manager can be found below.

Federates can publish or subscribe to an object class. A publisher of an object class can register an object instance of this class, the federate is then called the owner of the registered object. Each object has a set of attributes and, while the simulation is running the owner can update the attributes' values (UAV) of the object. Finally, the RTI reflects the new attributes' values (RAV) to all subscribers of the object class.

2.2 Finite continuous time Markov model

The distributed simulation has been modelled in a *Continuous Time Markov Chain* (CTMC). In this section, the definition of CTMC is presented, it is explained how it is possible to get non-exponential distribution with CTMC and why this class of models has been chosen to describe a real-time distributed simulation.

A Markov process is a stochastic process without any memory. It means that the probability for taking a transition depends only on the current state and not on the past of the process. Note, any process can be represented as a Markov process, including the history of the process in every current state. However this could mean that the amount of states may be infinite (possibly uncountable) which can be hard to verify. A finite continuous time Markov chain (CTMC) is a finite graph defining a Markov process in a continuous time domain, where the vertices of the graph represent the states and the values on the transitions represent the rate at which the transition is taking place. Below is a definition of finite continuous time Markov process. Further details can be found in [10].

Definition 2.1 *Let S be a finite set of states, matrix $Q : S \times S \mapsto \mathbb{R}_+$ the generator matrix, vector $E : S \mapsto \mathbb{R}_+$, $E(s) = \sum_{s' \in S} Q(s, s')$. Let $i : S \mapsto [0, 1]$ be a probabilistic distribution called the initial distribution. A* Continuous Time Markov Process *is a random process $\mathbf{X} = (X_t)_{t \in \mathbb{R}_+}$ defined by the following probabilistic law :*

- $\forall s \in S,\ P\{X_0 = s\} = i(s)$
- $\forall s, s' \in S\ \forall t, \Delta t \in \mathbb{R}_+$,

$$P\{X_{t+\Delta t} = s' | X_t = s\} = \begin{cases} Q(s, s') \times \Delta t + o(\Delta t) & \textit{if } s \neq s' \\ 1 - (E(s) + Q(s, s')) \times \Delta t + o(\Delta t) & \textit{if } s = s' \end{cases}$$

with $\lim_{\Delta t \to 0} \frac{o(\Delta t)}{\Delta t} = 0$

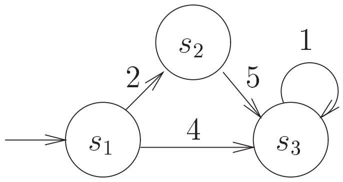


Fig. 1. *Example of a Continuous Time Markov Chain*

The above definition describes the behaviour of the system by probabilistic infinitesimal variation with respect to time. The global behaviour of the process is obtained by resolving a differential system. Thus, the probability of staying in a state s, decreases according to an exponential decreasing function with respect to time. From a microscopic point of view $Q(s, s')$ represents the increasing coefficient for taking the transition towards the state s' assuming that the system is in the state s. It also corresponds globally to the mean frequency of taking the transition (s, s'). Thus the mean time spent in s before taking a transition is $\frac{1}{E(s)}$. The probability for moving from state s to s' by a single transition, denoted $P(s, s')$, is $P(s, s') = \frac{Q(s,s')}{E(s)}$ if $E(s) \neq 0$, $P(s, s') = 0$ if $E(s) = 0$.

For example, a Markov process defined by $S = \{s_1, s_2, s_3\}$, $I(s_1) = 1$, $I(s_2) = I(s_3) = 0$, $Q(s_1, s_2) = 2$, $Q(s_1, s_2) = 4$, $Q(s_2, s_3) = 5$ $Q(s_3, s_3) = 1$, $Q(s_1, s_1) = Q(s_2, s_1) = Q(s_2, s_2) = 0 = Q(s_3, s_1) = Q(s_3, s_2) = 0$ (see *Figure* 1 for the corresponding graph), follows the probabilistic distributions $P\{X_t = s_1\} = e^{-6t}$, $P\{X_t = s_3\} = \frac{1}{3}(1 - e^{-2t})(1 - e^{-5t}) + \frac{2}{3}(1 - e^{-4t})$, $P\{X_{t+t'} = s_3 | X_t = s_3\} = 1$, $P\{X_{t+t'} = s_1 | X_t = s_3\} = P\{X_{t+t'} = s_1 | X_t = s_2\} = 0$.

Then considering a CTMC with an initial state s_i it means that the initial distribution i is such that $i(s_i) = 1$ and $i(s) = 0$ for all $s \neq s_i$. Usually the states of a CTMC are described by a finite set of bounded variables such as boolean, bounded integer or enumerate. Therefore the state space is defined by the Cartesian product of each variable domain. These variables are called state variables.

The consequence of having a description with the memory-less property is that the only type of distribution for taking a transition from the current state is exponential. However it is possible to approximate any distribution by composing in parallel or series a finite number of states (see [9] for further details). This has been made for modelling the network in section 3.3. Thus, in spite of the memory-less property of a Markov process it is still possible to represent (or approximate) general distributions without generating an infinite number of states. This important fact allows to model complex random processes with realistic distributions and then apply probabilistic model checking techniques on the model.

2.3 Performance Evaluation Process Algebra PEPA

PEPA as *Performance Evaluation Process Algebra* is a stochastic process algebra defined by Hillston [6] which permits to describe any finite CTMC. PEPA can be seen as a continuous time probabilistic extension of the *Calculus of Communicating Systems* (CCS) [8] or the *Communicating Sequential Processes* (CSP) [7].

Definition 2.2 *Let $\mathcal{A}$ be a set of* action *names. τ designs an unknown action and $\top$ an unknown rate. The syntax for terms in PEPA is defined as follows :*

$$P ::= (\alpha, r).P | P \underset{L}{\bowtie} Q | P + Q | P/L | P[l] | A$$

$(\alpha, r).P$ is the Prefix *operator which represents the elementary transition of the associated CTMC, α is an action name, r the rate for taking the transition and P the description of the process after taking the transition. $P \underset{L}{\bowtie} Q$,* cooperation *operator, is the synchronisation product of two processes P and Q on the* cooperation set *of actions $L \in \mathcal{A} \backslash \{\tau\}$. $P + Q$ is the* choice *operator. P/L is the* hiding *operator which keeps interactions off belonging to L between the process and the others. $P[l]$ is the action* renaming *operator, l being a mapping $\mathcal{A} \mapsto \mathcal{A}$. A is any* constant *defined by $A \stackrel{def}{=} P$.*

A given PEPA term designs a certain CTMC, the associated CTMC is gotten in two steps :

(i) getting the *derivation graph* induced by a set of Plotkin style semantic rules, called *Operational Semantics of PEPA* [6]. Basically, a derivation graph is a CTMC with action names added on the transitions.

(ii) getting the CTMC from the derivation graph by eliminating the action names of the transitions and lumping the equivalent transitions.

The Operational Semantics is quite standard and not recall here (see [6] for further explanation). In the case study of the paper only synchronisation of *active* (known rate) with *passive* (unknown rate, noted $\top$) transition has been used. The semantic of such a synchronisation is quite simple since the resulting transition rate is only defined by the active transition rate.

Example 2.3 *A definition using PEPA formalism of the CTMC of the* Figure *1 is given below :*

- $Model \stackrel{def}{=} Initial \underset{\{\alpha\}}{\bowtie} Loop$
- $Initial \stackrel{def}{=} (\tau, 2).(\tau, 5).Final + (\alpha, 4).Final$
- $Final \stackrel{def}{=} (\alpha, 1).Final$
- $Loop \stackrel{def}{=} (\alpha, \top).Loop$

The state space is formed of the terms induced by the Operational Semantics

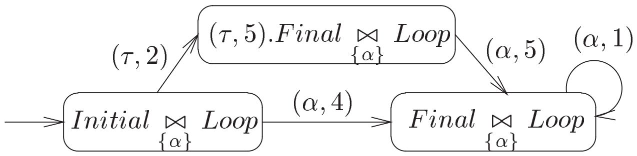


Fig. 2. *The derivation graph of the PEPA term Model*

of PEPA from the term Model. See Figure 2 *for the corresponding derivation graph and* Figure 1 *for the corresponding CTMC.*

2.4 The Continuous Stochastic Logic CSL

The Continuous Stochastic Logic (CSL) has been introduced in [1] in order to formalise a class of properties on the CTMC. CSL is the continuous time extension of PCTL *(Probabilistic Computation Tree Logic)* [5] which is the probabilistic extension of CTL *(Computation Tree Logic)* [2]. Its properties allows to express that the probability of having a certain behaviour meets a certain bound in the interval $[0, 1]$. A short definition of CSL is given below.

Definition 2.4 *the CSL formulae have the following syntax*

$$\Phi ::= \mathbf{true} \Big| \mathbf{a} \Big| \Phi \wedge \Phi \Big| \neg\Phi \Big| \mathcal{P}_{\bowtie p}[\Psi] \Big| \mathcal{S}_{\bowtie p}[\Phi]$$

$$\Psi ::= \mathcal{X}\Phi \Big| \mathcal{X}^I\Phi \Big| \Phi\ \mathcal{U}\ \Phi \Big| \Phi\ \mathcal{U}^I\ \Phi$$

where $\bowtie \in \{<, \leq, >, \geq\}$*,* $p \in [0, 1]$ *and* I *is an interval of* $\mathbb{R}$*.* Φ *denotes a* state formula *and* Ψ *denotes a* path formula*.*

The semantic is given below.

- *A state formula is interpreted over a CTMC with an initial state, also known as a current state.*
  - **true** *is always satisfied.*
  - **a** *is an* atomic *formula, usually a simple propositional variable. An atomic formula is satisfied when its interpretation in the current state is true.*
  - Boolean operators $\wedge$ *(resp.* $\neg$*) corresponds to the logic* **and** *(resp.* **not***).*
  - *A* probabilistic *formula* $\mathcal{P}_{\bowtie p}[\Psi]$ *is satisfied when the probability measure of the set of paths starting from the current state and satisfying the path formula* Ψ*, meets the bound* $\bowtie p$*.*
  - *A* steady-state *formula* $\mathcal{S}_{\bowtie p}[\Phi]$ *is satisfied when the probability of being in a state satisfying* Φ *when the time tends to infinity, meets the bound* $\bowtie$*. This limit is always defined, see [10].*
- *A path formula is interpreted over CTMC and a path* σ*.*
  - *The* next *formula* $X\Phi$ *is satisfied when* Φ *is satisfied by the second state of* σ*.*

- *The* bounded next *formula $X^I\Phi$ is satisfied when Φ is satisfied by the second state of σ and the time to reach this state is in the interval I.*
- *The* until *formula $\Phi_1\ \mathcal{U}\ \Phi_2$ is satisfied if Φ_1 is satisfied until Φ_2 is satisfied on the long run of the path σ.*
- *The* bounded until *formula $\Phi_1\ \mathcal{U}^I\ \Phi_2$ has the same semantic as the* until *formula but the time to reach Φ_2 from the beginning must be in the interval I.*

For example, the property, *the probability for the system to loose a message before 100s is lower than 0.01*, can be simply formalised by

$$\mathcal{P}_{<0.01}[\mathbf{true}\,\mathcal{U}^{<100}\ \mathbf{lost}]$$

where the atomic formula **lost** is true in each state representing the loss of a message.

In the case of verifying a CTMC generating from a PEPA term the state variables are folded in the initial term and the terms induced from the Operational Semantics of PEPA. Since the state space is defined by the induced terms the states variables have to be gotten from the terms. PRISM chooses to enumerate the states of each components described in the term. For the aim of simplicity, in this paper, it has been considered that each constant term (a constant term is defined by $\stackrel{def}{=}$) corresponds to a boolean variable. Such a boolean variable b is true in a state s if and only if there exists an equivalent term t which describe this state such that t contains the constant b without any prefix before it. For example, as the example 2.3, $Model$, $Initial$, $Final$ and $Loop$ are boolean variables. $Model$ and $Initial$ are true only in the state s_1, since s_1 is described by $Initial \underset{\{\alpha\}}{\bowtie} Loop$ which is equivalent to $Model$ (because of the constant definition $\stackrel{def}{=}$). $Final$ is only true in s_3 and $Loop$ is true in every state. $Final$ is not true in the state s_2 since $(\tau, 5)$ is before $Final$ and there is no equivalent state which contains $Final$ without any prefix before it.

2.5 The probabilistic model checker PRISM

PRISM is a probabilistic symbolic model checker which have been developed at the University of Birmingham by D. Parker, G. Norman and M. Kwiatkowska. It permits the description of Markov processes in a compositional way, takes in input description of each component of the system and according to the synchronisation between the components, generates the CTMC corresponding to the whole system. It is then possible to verify the CSL properties.

PRISM can take in input a several types of Markov processes, *discrete-time Markov chain* (DTMC), *Markov decision process* (MDP) and *Continuous-time Markov chain* (CTMC). It uses its own language to describe this different processes, handles labels on the transition to allow synchronisation product and variable states to describe the state space and rules to describe the transitions (see [11] for a full documentation of PRISM). It also accepts PEPA [6] (Performance Evaluation Process Algebra) to describe CTMC.

3 Case study

3.1 Description

In this case study, a federation composed of three federates and the RTI has been modelled. Each federate publishes a different object class and registers an object instance of this class. This object has only one attribute with a given length. The federates subscribe to all object classes published by the other federates. Each federate is a process located on a separate machine connected by a 100MBits/s Full duplex cable. The life of this process consists of three activities : computing, sending updated attribute values (UAV) of its object following a given frequency to each-other's federates and receiving reflected updated attribute values (RAV) from the other federates. The federation is a real-time simulation, so it does not use the time management function of the RTI.

A distributed RTI, such as CERTI [3] has been considered. This type of RTI is split into several processes : a central process called RTIG which is located, in our case study, on a separated machine, and one process per federate call RTIA located on the federate machine. The federate asks the RTIA to send the UAV through the network and the RTIA transmits the message to the RTIG. The RTIG then broadcasts the update to all subscribing RTIA's. In order to simplify the modelling, the RTIA process is modelled in the same process as the federate.

3.2 Methodology of the modelling

Since the model is quantitative, information about the rates of the transitions of the CTMC must be provided from the reality. The inverse of the transition rate of a CTMC transition corresponds to the first moment [2] (i.e the mean time) of the distribution for taking this transition. An exponential distribution (which is the distribution of a transition in a CTMC) is entirely described by its first moment, but a general distribution must require more moments, even every moment, to be entirely described. Some experiments and measures of 1) real real-time distributed simulations and 2) isolated network delay response and 3) RTIG delay response have been made to obtain the different rate values and other moments of the divers distributions. Modelling an exponential

[2] The nth moment of a random variable corresponds to the mean time of this variable after being raised to the power of n

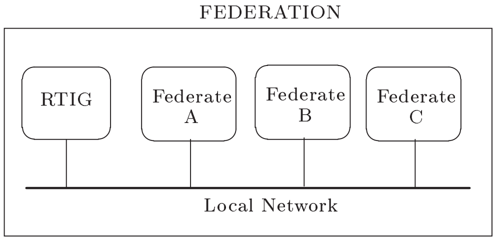


Fig. 3. *HLA Distributed Simulation*

distribution using CTMC is of course very easy, but when the distribution is not exponential it is necessary to make a composition of states (see subsection 2.2) which approximates this distribution. In this case study this procedure has been done completely manually using personal heuristic. However this procedure could probably be done automatically using learning algorithm. An important point of this phase is to maximise the precision of the approximation and minimise the number of the states added, in order to calm down the explosion of the state space.

3.3 Specification

Depending on a certain arrangement of the federates, the speed of the RTI and the throughput of the network, it is a proportion of the cases, reasonable to expect that of crossing the network within a certain deadline will be met. It is important to model not only the network but all the components and the middleware used to communicate and to verify this during a normal activity since all of these components will influence the duration of the data transfer.

To simplify the modelling, the model has been divided in several submodels or components. Each submodel has been represented as a PEPA term, the whole model being obtained by composition of all submodels. The three components are :

(i) The *federates+RTIA* which compute the evolution of the simulation

(ii) The *RTIG* (middleware that realise the intercommunications between the federates), whose role it is to decide on which federates the different messages are dispatched.

(iii) The *network* which conveys the messages between the federates and the RTIG.

Each component has inputs and outputs. By convention every action name representing an input (resp. output) begins by *in* (resp. *out*). It is important to notice that the name of the actions are local for each component. The communication between the different components has been made by renaming adequately their names before the synchronisation product, more precisely by connecting the output to the input properly between the Federates, RTIG and network. An input transition has always been passive, i.e. its rate is unknown, noted $\top$. An output transition has always been active, i.e with a known rate.

3.3.1 The federates

In the remainder of the description *federate+RTIA* is only denoted by *federate*. A federate computes cycles. At the end of each cycle it sends its UAV (named respectively a, b and c for the Federate A, B and C) to the RTIG. At any time during its cycle it can receive UAVs from the attributes of the other federates to whom it has subscribed. It does not wait to receive the updated values to keep working. In order to verify the time for crossing the network in a normal activity, it is necessary to distinguish between the messages of UAVs that are already in the network and those that have just been sent by the federate. Thus, before sending a message on the network, the federate adds a mark, denoted $_{\uparrow}$, on the messages used with the model checker to make the measurement. Under the assumption that no more than one marked message of the same updated values can be in the system at the same time, it is possible to write a CSL formula that measures the transmission time for a message from the source federate to the destination federates during normal system activity. It has been determined that in this model a marked message must be send at least every six cycles to guarantee that only one marked message is in the system at the same time. The value six comes from the number of entities which is crossed by the message in order to go from one federate to another. Furthermore, this property has been formalised and proved with PRISM before checking the property which concerns the performance measurement (see section 4).

Each federate results from the parallel product of two subprocesses. The first one ($SendFedA$ for Federate A) describes the process of sending UAVs, and the second one ($FedAgot$ for Federate A) describes the process for receiving RAVs. The PEPA term of Federate A is defined in *Table* 1. *Table* 2 contains the details of its inputs, outputs and rate.

The only goal of FedAgot is to memorise which last RAV has been received by Federate A. The model does not represent the content of a message, only its length, the attribute it represents and its mark. It is important to note that marked message and not marked message have the same behaviour [3].

Federate B (as Federate C) is similar to Federate A apart of the renaming of the letter A by B, a by b, b by a, c by c.

3.3.2 The RTIG

The role of the RTIG is to correctly dispatch the updated values from the federates to the federates which have subscribed to these updated values. The RTIG contains no buffer, it can treat only one message at a time. UAVs from the federates are taken from the network by the RTIG which then transmits

[3] This is easy to prove by checking that the definitions of the elementary behaviours are the same for both the marked and unmarked message.

$$FedA \stackrel{def}{=} SendFedA \underset{\emptyset}{\bowtie} FedAgot$$

$$SendFedA \stackrel{def}{=} \underbrace{(outa, \lambda_A) \ldots (outa, \lambda_A)}_{5}.SendFedA_{\uparrow}$$

$$SendFedA_{\uparrow} \stackrel{def}{=} (outa_{\uparrow}, \lambda_A).SendFedA$$

$$FedAgot \stackrel{def}{=} (inb, \top).FedAgotb$$

$$+(inb_{\uparrow}, \top).FedAgotb_{\uparrow}$$

$$+(inc, \top).FedAgotc$$

$$+(inc_{\uparrow}, \top).FedAgotc_{\uparrow}$$

$FedAgotb$, $FedAgotb_{\uparrow}$, $FedAgotc$, $FedAgotc_{\uparrow}$ have the same definition as $FedAgot$.

Table 1
Definition of the PEPA term $FedA$

$outa$: sending an unmarked UAV a to the network,

$outa_{\uparrow}$: sending a marked UAV a to the network,

inb : receiving an unmarked RAV b from the network,

$inb_{\uparrow}$, inc, $inc_{\uparrow}$: receiving RAV from the network,

λ_A : corresponding to the cycle frequency of Federate A, usually around 40Hz.

Table 2
Inputs, outputs and rate of $FedA$

these values (RAV) to the federates that are concerned with them. *Table* 3 defined the term $RTIG$ and $Reflect\_aSubscribers$. *Table* 4 contains the meaning of the action names and the rate of the term $RTIG$. If the RTIG is free and takes a UAV (a, b or c) from the network then it must reflect this updated value to the subscribers of this attributes. The process defined by the term $Reflect\_aSubscribers$ does reflect the updated value a to the federates which have subscribed to the attributed a, in this example Federates B and C. The definitions of $Reflect\_a_{\uparrow}Subscribers$, $Reflect\_bSubscribers$, $Reflect\_b_{\uparrow}Subscribers$, $Reflect\_cSubscribers$, $Reflect\_c_{\uparrow}Subscribers$ follow the same idea.

3.3.3 The network

The network is a local area network. Basically it is a full duplex cable connected to the machines housing the federates and the machine housing

$$RTIG \stackrel{def}{=} (ina, \top).Reflect\_aSubscribers$$
$$+(ina_{\uparrow}, \top).Reflect\_a_{\uparrow}Subscribers$$
$$+(inb, \top).Reflect\_bSubscribers$$
$$+(inb_{\uparrow}, \top).Reflect\_b_{\uparrow}Subscribers$$
$$+(inc, \top).Reflect\_cSubscribers$$
$$+(inc_{\uparrow}, \top).Reflect\_c_{\uparrow}Subscribers$$
$$Reflect\_aSubscribers \stackrel{def}{=} (outaB, \lambda_{RTIG}).(outaC, \lambda_{RTIG}).RTIG$$
$$+(outaC, \lambda_{RTIG}).(outaB, \lambda_{RTIG}).RTIG$$

Table 3
Definition of the PEPA term $RTIG$

$ina, \ldots, inc_{\uparrow}$: RTIG receives a UAV from the network,

$outaC, \ldots, outc_{\uparrow}B$: RTIG sends a RAV to the network for the federate A, B or C

λ_{RTIG} : rate for dispatching a message, depends on the speed of the machine housing the RTIG.

Table 4
Definition of the inputs, outputs and rate of the PEPA term $RTIG$

the RTIG. The communications from the federates to the RTIG is carried out by one half duplex cable, and the communications from the RTIG to the federates are carried out by the other half.

It has been previously measured that the distribution of the delay for transporting a message on a network is not exponential. However every measure made on a LAN (by sending pings) has given a Erlang-k law [4]. An Erlang-k law with rate λ is obtained by a series of k steps of exponential law with rate $\frac{\lambda}{k}$, which is represented by a Markov chain of k states connected in a serial way (see the *figures* 4 and 5). This may be explained because a message is split in several packets before to be transmitted and each packet has to cross several logical and physical entities before arriving to the destination.

The network results from the parallel composition of two processes, $LANF2R$ which conveys UAVs from the federates to the RTIG and $LANR2F$ which conveys RAVs from the RTIG to the federates. The subprocesses *Erlang-k*[...] are used to obtain a realistic delay distribution. *Table* 5 contains the definition of the PEPA term $Network$ and *Table* 6 the meanings of

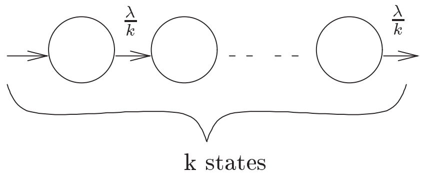


Fig. 4. *Markov chain of a Erlang-k law with a rate* λ

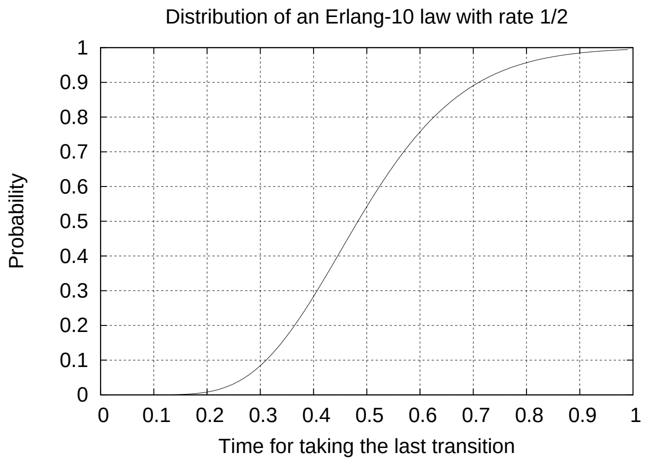


Fig. 5. Distribution of an Erlang-k law

its inputs, outputs and rate.

3.3.4 The model of the real-time distribution simulation

The whole model is obtained by the composition of the federates, the RTIG and the network. The biggest part of the definition concerns the renaming of the action names of the federates plus RTIG to connect them to the network. *Table* 7 contains the definition of the PEPA term that models the real-time distributed simulation.

The set of labels $\mathcal{I}$ contains all the interactions implied in the communication between the federates and the network as $\mathcal{J}$ between the RTIG and the network, see *Table* 8 for their definitions.

4 Performance evaluation with CSL

In this section it is shown how CSL can be used to formalise the time constraints needed to be verified by the system. In the case of a real-time simulation the main property needed to be verified is a rough deadline of the time required for data to leave one federate and reach the others.

An example is given by formalising that the duration for a UAV for crossing the network and the RTIG from the federate A to the federate C is less than 1ms in at least 90% of the cases.

$$Network \stackrel{def}{=} LANF2R \underset{\emptyset}{\bowtie} LANR2F$$

$$LANF2R \stackrel{def}{=} (ina2R, \top).Erlang\text{-}k\_UAVa$$

$$+(ina_{\uparrow}2R, \top).Erlang\text{-}k\_UAVa_{\uparrow}$$

$$+(inb2R, \top).Erlang\text{-}k\_UAVb$$

$$+(inb_{\uparrow}2R, \top).Erlang\text{-}k\_UAVb_{\uparrow}$$

$$+(inc2R, \top).Erlang\text{-}k\_UAVc$$

$$+(inc_{\uparrow}2R, \top).Erlang\text{-}k\_UAVc_{\uparrow}$$

$$Erlang\text{-}k\_UAVa \stackrel{def}{=} \underbrace{(\tau, \lambda_a/k)\ldots(\tau, \lambda_a/k)}_{k-1}.(outa2R, \lambda_a/k).LANF2R$$

$$LANR2F \stackrel{def}{=} (ina2B, \top).Erlang\text{-}k\_RAVaFedB$$

$$+(ina_{\uparrow}2B, \top).Erlang\text{-}k\_RAVa_{\uparrow}FedB$$

$$+(ina2C, \top).Erlang\text{-}k\_RAVaFedC$$

$$+(ina_{\uparrow}2C, \top).Erlang\text{-}k\_RAVa_{\uparrow}FedC$$

$$+(inb2A, \top).Erlang\text{-}k\_RAVbFedA$$

$$+(inb_{\uparrow}2A, \top).Erlang\text{-}k\_RAVb_{\uparrow}FedA$$

$$+(inb2C, \top).Erlang\text{-}k\_RAVbFedC$$

$$+(inb_{\uparrow}2C, \top).Erlang\text{-}k\_RAVb_{\uparrow}FedC$$

$$+(inc2A, \top).Erlang\text{-}k\_RAVcFedA$$

$$+(inc_{\uparrow}2A, \top).Erlang\text{-}k\_RAVc_{\uparrow}FedA$$

$$+(inc2B, \top).Erlang\text{-}k\_RAVcFedB$$

$$+(inc_{\uparrow}2B, \top).Erlang\text{-}k\_RAVc_{\uparrow}FedB$$

$$Erlang\text{-}k\_RAVaFedB \stackrel{def}{=} \underbrace{(\tau, \lambda_a/k)\ldots(\tau, \lambda_a/k)}_{k-1}.(outa2B, \lambda_a/k).LANR2F$$

Table 5
Definition of the PEPA term *Network*

First, it has been proved with PRISM that if a marked message is sent from Federate A then no other marked message from A is presented in the other parts of the system. This is in order not to confuse the sending marked message with an other wandering marked message.

$ina2R, \ldots, inc_{\uparrow}2R$: taking in UAVs from the federates to the RTIG,
$outa2R, \ldots, inc_{\uparrow}2R$: sending out UAVs from the federates to the RTIG,
$ina2B, \ldots, inc_{\uparrow}2B$: taking in RAVs from the RTIG to the federates,
$outa2B, \ldots, outc_{\uparrow}2B$: sending out RAVs from the RTIG to the federates,
λ_a : rate of the transfer of the UAVs/RAVs a, depends on
the length of the message and the throughput,
k : number of steps of the Erlang law, depends on the second moments
of the delay distribution of the UAVs/RAVs.

Table 6
Definition of the inputs, outputs and rate of the PEPA term $Network$

$$RealTimeDistSimu \stackrel{def}{=} (FederatesConnect \underset{\mathcal{I}}{\bowtie} Network \underset{\mathcal{J}}{\bowtie} RTIGConnect)$$

$$\begin{aligned}
&FederatesConnect \stackrel{def}{=} FedA[outa \leftarrow ina2R, outa_{\uparrow} \leftarrow ina_{\uparrow}2R,\\
&inb \leftarrow outb2A, inb_{\uparrow} \leftarrow outb_{\uparrow}2A, inc \leftarrow outc2A, inc_{\uparrow} \leftarrow outc_{\uparrow}2A]\\
&\underset{\emptyset}{\bowtie} FedB[outb \leftarrow inb2R, outb_{\uparrow} \leftarrow inb_{\uparrow}2R,\\
&ina \leftarrow outa2B, ina_{\uparrow} \leftarrow outa_{\uparrow}2B, inc \leftarrow outc2B, inc_{\uparrow} \leftarrow outc_{\uparrow}2B]\\
&\underset{\emptyset}{\bowtie} FedC[outc \leftarrow inc2R, outc_{\uparrow} \leftarrow inc_{\uparrow}2R,\\
&ina \leftarrow outa2C, ina_{\uparrow} \leftarrow outa_{\uparrow}2C, inb \leftarrow outb2C, inb_{\uparrow} \leftarrow outb_{\uparrow}2C]
\end{aligned}$$

$$\begin{aligned}
&RTIGConnect \stackrel{def}{=} RTIG[ina \leftarrow outa2R, ina_{\uparrow} \leftarrow outa_{\uparrow}2R,\\
&inb \leftarrow outb2R, inb_{\uparrow} \leftarrow outb_{\uparrow}2R, inc \leftarrow outc2R, inc_{\uparrow} \leftarrow outc_{\uparrow}2R,\\
&outaB \leftarrow ina2B, outa_{\uparrow}B \leftarrow ina_{\uparrow}2B, outaC \leftarrow ina2C, outa_{\uparrow}C \leftarrow ina_{\uparrow}2C,\\
&outbA \leftarrow ina2A, outb_{\uparrow}A \leftarrow ina_{\uparrow}2A, outbB \leftarrow ina2B, outb_{\uparrow}B \leftarrow ina_{\uparrow}2B,\\
&outcA \leftarrow ina2A, outc_{\uparrow}A \leftarrow ina_{\uparrow}2A, outcB \leftarrow ina2B, outc_{\uparrow}B \leftarrow ina_{\uparrow}2B]
\end{aligned}$$

Table 7
Definition of the whole model of the real-time distributed simulation

$$\begin{aligned}
Erlang\text{-}k_UAVa_{\uparrow} \Rightarrow\ & \neg Reflect_a_{\uparrow}Subscribers\\
& \wedge\neg Erlang\text{-}k_RAVa_{\uparrow}FedB\\
& \wedge\neg Erlang\text{-}k_RAVa_{\uparrow}FedC\\
& \wedge\neg FedBgota_{\uparrow} \wedge \neg FedCgota_{\uparrow}
\end{aligned}$$

According to the definition of the PEPA terms of the model, $Erlang\text{-}k\_UAVa_{\uparrow}$ is true when the UAV $a_{\uparrow}$ is put in the network from the Federate A to the

$\mathcal{I} = \{ina2R, ina_\uparrow 2R, inb2R, inb_\uparrow 2R, inc2R, inc_\uparrow 2R,$

$outa2B, outa_\uparrow 2B, outa2C, outa_\uparrow 2C,$

$outb2A, outb_\uparrow 2A, outb2C, outb_\uparrow 2C,$

$outc2A, outc_\uparrow 2A, outc2B, outc_\uparrow 2B\}$

$\mathcal{J}$ is similar to $\mathcal{I}$ but in is substituted by out and out is substituted by in in each label.

Table 8
Definition of the cooperation sets $\mathcal{I}$ and $\mathcal{J}$

RTIG. $Reflect\_a_\uparrow Subscribers$ is true when a message $a_\uparrow$ is being treated by the RTIG. $Erlang\text{-}k\_RAVa_\uparrow FedB$ (resp. $Erlang\text{-}k\_RAVa_\uparrow FedC$) is true when the network contains a RAV $a_\uparrow$ to Federate B (resp. C). $FedBgota_\uparrow$ (resp. $FedCgota_\uparrow$) is true when Federate B (resp. C) has received the RAV $a_\uparrow$.

It is then possible to verify the probabilistic desired performance constraint formalised below :

$$Erlang\text{-}k\_UAVa_\uparrow \Rightarrow P_{>0.9}[true\, \mathcal{U}^{<0.001} FedCgota_\uparrow]$$

5 Results with PRISM

A description of this model has been written in the probabilistic model checker PRISM. The characteristic of the system can be tuned by some parameters as length of the UAV messages, speed of the machine housing the RTIG and throughput of the network. It is then possible to check if the time constraints are verified depending on the chosen parameters. It is important to realise that the relation between the parameters of the model and the parameters of the reality is not quite direct, indeed the model is an abstraction of the reality and a lot of details have not been explicitly described, as for instance the different levels of the communication protocol. Thus it is necessary to calibrate all the rates of the model in the way that if the parameter values of the model and the parameter values of the reality are the same, then they both give the same results. The calibration is simply done by adding coefficients at the rates of the model and changing them until the model and the reality coincide for a given set of parameter values.

In this case study only the dimension of the length has been considered to calibrate the model with the reality. However on this dimension the model has given good predictive results (see section 6). So the results given by the model with respect to the speed of the RTIG and the throughput of the network are only theoretic and probably do not give the expected realistic results.

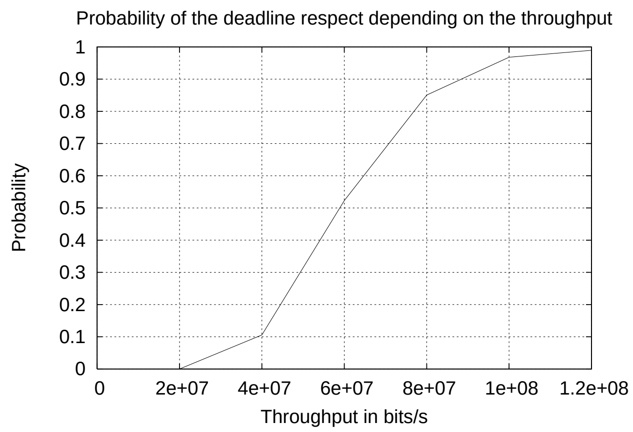


Fig. 6. PRISM results with respect to the throughput

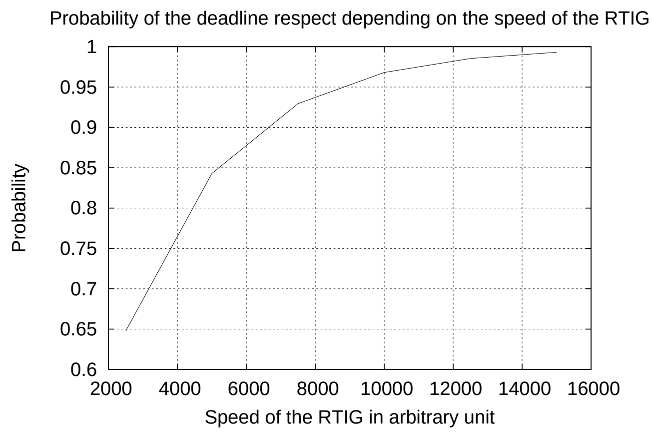


Fig. 7. PRISM results with respect to the speed of the RTIG

The graphs of this section show the probability for respecting the deadline in the duration of the trip of the messages with respect to different parameters as speed of the RTIG, throughput of the network and length of the UAVs. These results have been computed using an additional CSL operator provided by PRISM which permits the computation of the probability measure of a set of executions verifying a certain property. The results of these performance evaluations are shown in the above graphs.

The *Figures* 6 and 7 shows the probability of the deadline respect according to the throughput of the network and the speed of the RTIG. *Figure* 8 shows the probability of verifying the deadline respect according to the time of the deadline varying from 0ms to 1ms and for three different lengths of the UAVs, 64, 256 and 512 octets.

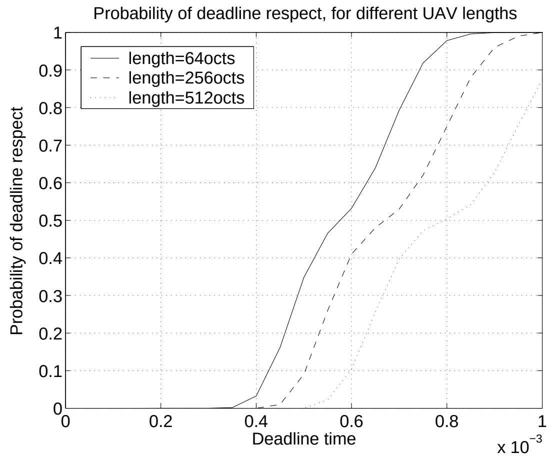


Fig. 8. PRISM results with respect to the deadline time for divers lengths

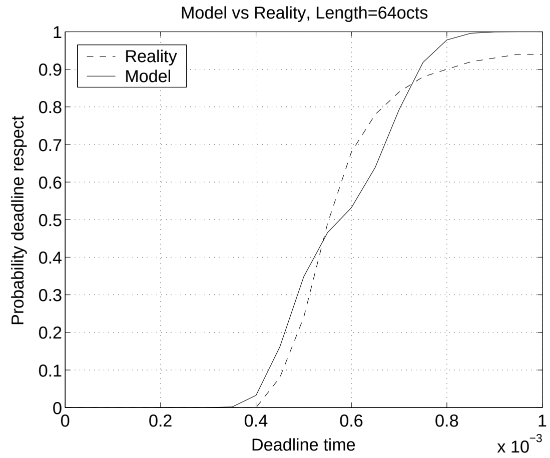


Fig. 9. Model vs Reality, length=64octs

6 First comparison results with a real case

This section contains a first comparison of the model with a real system. A real-time simulation has been distributed on two machines connected by a full duplex cable of 100Mbits/s. As in the model, the system is composed of three federates and a RTIG. In order to measure the duration for sending a UAV from a federate to another with the same clock, the three federates have been housed by the same machine. The RTIG has been housing by an Intel Pentium 4 1500MHz and the federates have been housing by a quadri-processor Intel 800MHz. This first experiment has provided useful information to calibrate the rates of the transitions of the model. The *figures* 9, 10 and 11 shows the comparison of the model with the real case over different lengths of the UAVs. It appears that the model is a good approximation of the reality over the different lengths. It has been observed that the Erlang-k distribution chosen to model the network, instead of the basic exponential distribution, has greatly contributed to the realism of the model.

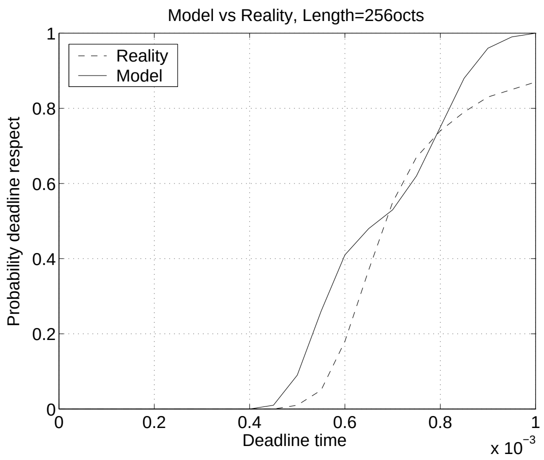


Fig. 10. Model vs Reality, length=256octs

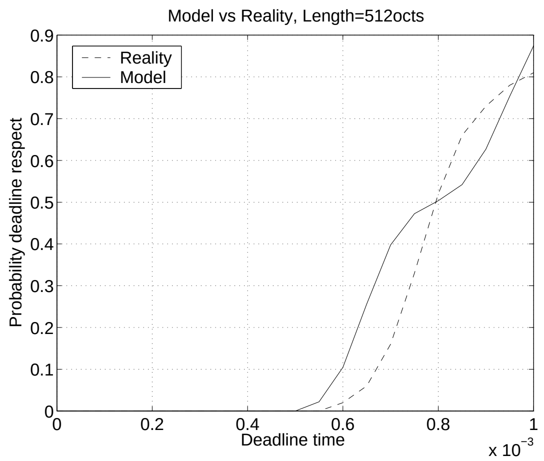


Fig. 11. Model vs Reality, length=512octs

7 Conclusion and Future work

A real-time distributed simulation based on the High Level Architecture HLA has been modelled in a continuous time Markov chain using the process algebra PEPA. A time constraint on the duration for transmitting a message from a federate to another has been formalised in CSL and verified with the model checker PRISM over different parameter values. A first experimentation on different lengths of messages exchanged by the federates has been made and has given good results over these values.

Validation over the other parameters, extension of the model for more complex networks as the Internet are in progress.

8 Acknowledgements

Special thanks to Bruno d'Ausbourg, Kirsty Forward, Boris Nordmann, Gregg Smith, Pamela Wilkie and four anonymous readers from the workshop.

References

[1] A. Aziz, K. Sanwal, V. Singhal and R. K. Brayton, *Verifying continuous-time Markov chains*, in: Rajeev Alur and Thomas A. Henzinger, editors, *Eighth International Conference on Computer Aided Verification CAV*, Lecture Notes in Computer Science **1102** (1996), pp. 269–276.
URL `citeseer.nj.nec.com/aziz96verifying.html`

[2] Clarke, E. M. and E. A. Emerson, *Design and synthesis of synchronization skeletons using branching time temporal logic*, in: *Proc. Logics of Programs Workshop*, LNCS **131**, Yorktown Heights, New York, 1981, pp. 52–71.

[3] d'Ausbourg, B. and P. Siron, *CERTI : Evolutions of the ONERA RTI prototype*, in: *Fall 2002 Simulation Interoperability Workshop*, 2002.

[4] Erlang, A. K., *Solution of some problems in the theory of probabilities of significance in automatic telephone exchanges*, P. O. Electrical Engineering Journal **10** (1917), pp. 189–197.

[5] Hansson, H. and B. Jonsson, *A logic for reasoning about time and reliability*, Formal Aspects of Computing **6** (1994), pp. 512–535.
URL `citeseer.nj.nec.com/hansson94logic.html`

[6] Hillston, J., *Compositional Markovian modelling using a process algebra*, in: W. J. Stewart, editor, *Proceedings of the 2nd International Workshop on Numerical Solution of Markov Chains* (1995).

[7] Hoare, C., *Communicating sequential processes*, in: *Comm. ACM*, 1978, pp. 666–677.

[8] Milner, R., "A Calculus of Communicating Systems," Lecture Notes in Computer Science **92**, Springer-Verlag, 1980.

[9] Newman, D. and A. Reddy, *Rational approximation to ex and tro related functions*, J. Approximation Theory **25** (1979), pp. 21–30.

[10] Norris, J. R., "Markov Chains," Cambridge University Press, 1997.

[11] Parker, D., G. Norman and M. Kwiatkowska, *PRISM user's guide* (2004).