Available at www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT®

ELSEVIER

Performance Evaluation 53 (2003) 43–69

PERFORMANCE EVALUATION
An International Journal

www.elsevier.com/locate/peva

Iterative disaggregation for a class of lumpable discrete-time stochastic automata networks☆

Oleg Gusak[a,*], Tuğrul Dayar[b], Jean-Michel Fourneau[c]

[a] *School of Interdisciplinary Computing and Engineering, University of Missouri–Kansas City, 5100 Rockhill Road, Kansas City, MO 64110-2499, USA*
[b] *Department of Computer Engineering, Bilkent University, 06533 Bilkent, Ankara, Turkey*
[c] *Lab. PRiSM, Université de Versailles, 45 Av. des États Unis, 78035 Versailles Cedex, France*

Received 6 June 2000; received in revised form 26 August 2002

Abstract

Stochastic automata networks (SANs) have been developed and used in the last 15 years as a modeling formalism for large systems that can be decomposed into loosely connected components. In this work, we concentrate on the not so much emphasized discrete-time SANs. First, we remodel and extend an SAN that arises in wireless communications. Second, for an SAN with functional transitions, we derive conditions for a special case of ordinary lumpability in which aggregation is done automaton by automaton. Finally, for this class of lumpable discrete-time SANs we devise an efficient aggregation–iterative disaggregation algorithm and demonstrate its performance on the SAN model of interest.

Keywords: Discrete-time stochastic automata networks; Ordinary lumpability; Iterative disaggregation; Wireless asynchronous transfer mode system

1. Introduction

Recently, a modeling paradigm called stochastic automata networks (SANs) [3–7,9,12,15–22,27,28,30–32,35–37] has received attention. SANs provide a methodology for modeling Markovian systems with interacting components. The main idea is to decompose the system of interest into its components and to model each component separately. Once this is done, interactions and dependencies among components can be brought into the picture and the model finalized. With this decompositional approach, the potential state space of the system is equal to the product of the number of states of the individual components. The benefit of the SAN approach is twofold. First, each component can be modeled much easier compared

☆ This work is supported by TÜBİTAK-CNRS grant and is done while the first author was at the Department of Computer Engineering, Bilkent University.
* Corresponding author. Tel.: +1-816-235-5940; fax: +1-816-235-5159.
E-mail address: gusako@umkc.edu (O. Gusak).

0166-5316/02/$ – see front matter
PII: S0166-5316(02)00227-4

to the global system due to state space reduction. Second, space required to store the description of components is minimal compared to the case in which transitions from each global state are stored explicitly. However, all this happens at the expense of increased analysis time [4,5,7,9,12,15,16,20,35–37].

In an SAN (see [35, Chapter 9]), each component of the global system is modeled by a stochastic automaton. When automata do not interact (i.e., when they are independent of each other), description of each automaton consists of local transitions only. In other words, local transitions are those that affect the state of one automaton. Local transitions can be constant (i.e., independent of the state of other automata) or they can be functional. In the latter case, the transition is a function of the global state of the system. Interactions among components are captured by synchronizing transitions. Synchronization among automata happens when a state change in one automata causes a state change in other automata. Similar to local transitions, synchronizing transitions can be constant or functional.

A discrete-time system of N components can be modeled by a single stochastic automaton for each component [27,30]. Local transitions of automaton k, denoted by $\mathcal{A}^{(k)}$, $k \in \{0, 1, \ldots, N-1\}$, are modeled by the local transition probability matrix $L^{(k)}$ which has equal row sums of 1. When there are S synchronizing events in the system, automaton k has the transition probability matrix $R_s^{(k)}$ that represents the contribution of $\mathcal{A}^{(k)}$ to synchronization $s \in \{0, 1, \ldots, S-1\}$ and the corresponding diagonal corrector matrix $N_s^{(k)}$ whose each diagonal element is the sum of the elements in the corresponding row of $R_s^{(k)}$. When $\mathcal{A}^{(k)}$ is not involved in synchronizing event s, we have $R_s^{(k)} = L^{(k)}$. The underlying discrete-time Markov chain (DTMC) corresponding to the global system can be obtained from [27,30]

$$P = \bigotimes_{k=0}^{N-1} L^{(k)} + \sum_{s=0}^{S-1} \left(\bigotimes_{k=0}^{N-1} R_s^{(k)} - \bigotimes_{k=0}^{N-1} N_s^{(k)} \right). \tag{1}$$

We refer to the tensor (i.e., [10]) representation in Eq. (1) associated with the DTMC as the descriptor of the SAN. Assuming that $\mathcal{A}^{(k)}$ has n_k states, the size of the potential state space of the global system is $n = \prod_{k=0}^{N-1} n_k$. When there are functional transitions, tensor products become generalized tensor products [32]. We denote by $\mathcal{A}^{(k)}[\mathcal{A}^{(l)}]$ a functional dependency between $\mathcal{A}^{(k)}$ and $\mathcal{A}^{(l)}$ if elements in the matrices of $\mathcal{A}^{(k)}$ depend on the state of $\mathcal{A}^{(l)}$.

Interactions among automata represented by events in an SAN are of the synchronous type. A synchronizing event can take place only in certain states of its involved automata. Recall that for each state of an automaton, the sum of the elements in the corresponding row of the local transition probability matrix is 1. Thus, the probabilities associated with synchronizing transitions are not reflected in the local transition probability matrices of automata. Hence, for the expression in Eq. (1) to form a stochastic matrix, normalization matrices are used. Some discrete-time systems that are modeled as a network of stochastic automata may experience changes in the states of its components due to independent events originating from outside the system rather than been triggered by its components. In this case, the evolution of the system due to local transitions can be treated as the complementary event of no other external events taking place. Therefore, normalization matrices are not needed. Hence, assuming that there are E independent events (i.e., $(E-1)$ external events) that affect the state of the system, the DTMC underlying the SAN can be obtained from

$$P = \sum_{e=0}^{E-1} \bigotimes_{k=0}^{N-1} P_e^{(k)}, \tag{2}$$

where the transition probability matrix $P_e^{(k)}$, $k \in \{0, 1, \ldots, N-1\}$, describes the evolution of $\mathcal{A}^{(k)}$ when event $e \in \{0, 1, \ldots, E-1\}$ takes place. Let γ_e be the probability of event e such that $\sum_{e=0}^{E-1} \gamma_e = 1$. Observe that P is a stochastic matrix if $(1/\gamma_e)P_e = \bigotimes_{k=0}^{N-1} P_e^{(k)}$ is a stochastic matrix that describes the evolution of the global system conditioned on event e taking place.

To the best of our knowledge, only a single SAN model whose descriptor is given by Eq. (1) has appeared in the literature [30]. It is an SAN model of the mutual exclusion algorithm of Lamport. There are $(p+2)$ automata and (p^2+3p) synchronizing events in the SAN, where p is the number of processes that access a shared resource. Note that due to the large number of synchronizing events, the SAN model is intractable even for moderate values of p. In [30], the description of the SAN is given for the case $p = 2$. Note also that therein numerical experiments with this SAN model do not appear. Examples of SAN models whose underlying DTMCs are given by Eq. (2) can be found in [19,29,38]. The potential state spaces of the SANs considered in [19,29,38] are relatively small and the number of automata in each model does not exceed 3. Hence, analyses of the DTMCs underlying these SAN models are not difficult. In this work, we contribute to the existing research on discrete-time SANs by modeling an application whose underlying MC is relatively dense and its potential state space grows exponentially with the integer parameters of the problem. We consider the wireless communication system that employs multiservices resource allocation policy introduced in [38]. We provide a new SAN model that is scalable with respect to the number of events in the system. Furthermore, we extend the SAN model by introducing a service for variable bit rate (VBR) requests.

Iterative solution methods for SANs employ an efficient vector–descriptor multiplication algorithm [15,27]. Implementation issues and complexity analysis of the algorithm for the case of generalized tensor products are considered in [15,16]. Improved versions of the algorithm that consider multiplication of a subset of states in the vector with the descriptor are discussed in [8]. Various iterative solution methods for SANs that take advantage of the efficient vector–descriptor multiplication algorithm have been studied in the last 10 years. In particular, application of projection methods to SANs is discussed in [5,35,36] and experiments with circulant preconditioners for SANs appear in [9]. In [37], a recursive implementation of iterative methods based on splittings that take advantage of the tensor structure of the SAN descriptor is introduced. An iterative aggregation-disaggregation (IAD) algorithm for SANs, in which aggregation at each iteration is done with respect to the states of an automaton chosen adaptively, appears in [4]. Nevertheless, we remark that so far numerical solution methods have been studied on continuous-time SAN models.

Lumping (i.e., exact aggregation) [24] is another approach that can aid in the analysis of systems with large state space. In the rest of the paper we use the concepts of lumping and exact aggregation interchangeably. Different kinds of lumpability in finite Markov chains (MCs) and their properties are considered in [1]. Results on exact aggregation of large systems whose MCs are composed of tensor products appear in [2,23,33]. Notion of exact performance equivalence for SANs is introduced in [6] and application of ordinary and exact lumpability to SANs is discussed in [3]. We remark that existing work considers the application of exact aggregation only to continuous-time MCs resulting from tensor-based formalisms, although the presented results are valid for DTMCs as well.

In this work, we consider the application of ordinary lumpability to discrete-time SANs. Let the state space $\mathcal{S} = \{0, 1, \ldots, n-1\}$ of a MC given by P be partitioned into K subsets $\mathcal{S}_0, \mathcal{S}_1, \ldots, \mathcal{S}_{K-1}$ such that $\bigcup_{k=0}^{K-1} \mathcal{S}_k = \mathcal{S}$ and $\mathcal{S}_k \cap \mathcal{S}_l = \emptyset$ for $k \neq l$. Following [1], we say a MC is ordinarily lumpable with respect to the partitioning $\mathcal{S}_0, \mathcal{S}_1, \ldots \mathcal{S}_{K-1}$ if for all states $x, y \in \mathcal{S}_k$ and subsets $\mathcal{S}_l$, $k, l = 0, 1, \ldots, K-1$, $\sum_{j \in \mathcal{S}_l} P(x, j) = \sum_{j \in \mathcal{S}_l} P(y, j)$. In other words, P is ordinarily lumpable if each block in the partitioning

of P has equal row sums. In contrast to the existing work on exact aggregation of SANs and related high-level formalisms, we consider an SAN in its general form with functional transitions. In other words, we assume that the descriptor corresponding to the SAN model at hand is a sum of generalized tensor products. We derive easy to check conditions on descriptions of automata and their ordering that enable us to identify a class of ordinarily lumpable partitionings in which lumping happens automaton by automaton. We remark that most of the existing work on exact aggregation of tensor-based formalisms amounts to defining equivalence relations among states in a component of the modeled system or among the components of the system. For instance, in [2] where exact aggregation is applied to hierarchical Markovian models, aggregation is done with respect to identical classes of customers inside low level models, with respect to identical low level models, and with respect to identical states of the high-level model. The goal of our work is to identify ordinarily lumpable partitionings of P induced by the block structure of tensor product. Obviously, this kind of ordinarily lumpable partitionings is a special case of the lumpability and performance equivalence considered in [2,6]. On the other hand, simplicity of the conditions for ordinary lumpability that we impose on the SAN description allows us to show that some of the SAN models that have been considered before are lumpable.

Finally, we remark that existing research on ordinary lumpability in tensor-based formalisms consider the analysis of the aggregated system, whereas we aim at solving the original system and do not assume a specific Markov reward structure. For a class of ordinarily lumpable discrete-time SANs, we introduce an aggregation–iterative disaggregation (AID) algorithm that takes advantage of the tensor structure of the lumpable partitionings and discuss its implementation details. The introduced algorithm is a modified version of Koury–McAllister–Stewart's IAD algorithm [34] in which aggregation is performed once and disaggregation is a block Gauss–Seidel (BGS) step. To the best of our knowledge, implementation and results of experiments with this kind of IAD algorithm for MCs composed of tensor products is not known in the literature. The algorithm considered in [4] utilizes a different approach in which aggregation at each iteration is done with respect to the states of an automaton chosen adaptively. It should also be emphasized that in the experiments of Buchholz [4] the disaggregation phase of the algorithm is a power iteration, which is inferior to BGS.

In summary, we remark that the number of discrete-time SAN models that appear in the literature are just a handful, they are relatively simple, and we do not know of a study of iterative methods for analyzing them. In our paper, we consider a relatively complex and meaningful SAN model of a wireless TDMA system. Although the concept of ordinary lumpability for SANs has been discussed elsewhere, we stress that in our paper it is applied to SANs having functional dependencies and that our aim is to obtain the complete steady state vector of the underlying MC without imposing any restrictions on the associated Markov reward structure. Besides, we consider the implementation of an IAD algorithm for SANs that was not done before partially due to the fact that an aggregate matrix needs to be formed and solved at each iteration, but more importantly, because the implementation of BGS for SANs used in its disaggregation phase was not clear until recently. The technique devised in our paper handles these problems gracefully, albeit for a class of discrete-time SANs.

In Section 2 we introduce the wireless ATM model that is investigated. In doing so, we first present the basic model with two types of services, show that it can be modeled with a fixed number of events and then add a third service giving a more general model. In Section 3 we give the result regarding ordinary lumpability for a class of discrete-time SANs with functional transitions and introduce the efficient AID algorithm. In Section 4 we provide the results of numerical experiments and in Section 5 we conclude.

2. A wireless ATM system

The application that we consider arises in wireless asynchronous transfer mode (ATM) networks. In [38], a multiservice resource allocation policy is developed to integrate two types of service over a time division multiple access (TDMA) system in a mobile communication environment. These are the constant bit rate (CBR) service for two types of voice calls (i.e., handover calls from neighboring cells and new calls) and the available bit rate (ABR) service for data transfer. A single cell and a single carrier frequency is modeled.

The TDMA frame is assumed to have C slots. Handover CBR requests have priority over new CBR calls and they, respectively, arrive with probabilities p_{h} and p_{n}. We do not consider multiple handover or new CBR call arrivals during a TDMA frame since the associated probabilities with these events are small. Each CBR call takes up a single slot of a TDMA frame but may span multiple TDMA frames, whereas each data packet is small enough to be served in a single TDMA slot. When all the slots are full, incoming CBR calls are rejected. The number of CBR calls that may terminate in a given TDMA frame depends on the number of active CBR calls. Since the probability of a CBR call departure, p_s, is usually small, we introduce the parameter M ($<C$) which specifies the maximum number of CBR calls that can terminate in a given TDMA frame. Thus, the number of CBR calls that can terminate in a given TDMA frame is modeled as a truncated binomial process with parameter p_s.

Data is queued in an FIFO buffer of size B and has the least priority. The arrival of data packets is modeled as an on–off process. The process moves from the on state to the off state with probability α and from the off state to the on state with probability β. The load offered to the system is defined as $\lambda = \beta/(\alpha + \beta)$. Assuming that the time interval between two consecutive on periods is t, the burstiness of such an on–off process is described by the square coefficient of variation, $S_C = \mathrm{Var}(t)/[E(t)]^2$. In terms of λ and S_C, $\beta = 2\lambda(1-\lambda)/(S_C + 1 - \lambda)$ and $\alpha = \beta(1-\lambda)/\lambda$. When the on–off process is in the on state, we assume that $i \in \{0, 1, 2, 3\}$ data packets may arrive with probability $p_d(i)$. The mean arrival rate of data packets is defined as $\rho = \sum_{i=1}^{3} i \times p_d(i)$ and the global mean arrival rate of data packets is given by $\Gamma = \lambda\rho$. If the number of arriving data packets exceeds the free space in the buffer plus the number of free slots in the current TDMA frame, then the excess packets are blocked. The arrival process of data and the service process of CBR calls we consider are quite general and subsume those in [38]. Furthermore, compared to the model in [38], the number of events in our SAN model is independent of C.

The performance measures of interest are the dropping probability of handover CBR calls, the blocking probability of new CBR calls, and the blocking probability of data packets. Note that dropping refers to the rejection of an existing call, whereas blocking refers to the rejection of a new call or packet.

We model this system as a discrete-time SAN whose descriptor has the form in Eq. (2). We assume that state changes in the system occur at TDMA frame boundaries. In particular, data packet and call arrivals to the system happen at the beginning of a frame and data packet transmissions finish and CBR calls terminate at the end of the frame. Since each data packet is transmitted in a single slot, in a particular state of the system we never see slots occupied by data packets.

2.1. The basic SAN model

The basic SAN model consists of three automata and three events. States of all automata are numbered starting from 0. We denote the state index of automaton k by $s\mathcal{A}^{(k)}$; $\mathcal{A}^{(0)}$ represents the data source;

$\mathcal{A}^{(1)}$ represents the current TDMA frame; and $\mathcal{A}^{(2)}$ represents the data buffer. We define the three events e_0, e_1, e_2 that correspond to, respectively, 0, 1, 2 CBR arrivals during the current TDMA frame. Event e_a happens with probability γ_a, $a \in \{0, 1, 2\}$, where $\gamma_0 = \bar{p}_\mathrm{n}\bar{p}_\mathrm{h}$, $\gamma_1 = p_\mathrm{n}\bar{p}_\mathrm{h} + p_\mathrm{h}\bar{p}_\mathrm{n}$, $\gamma_2 = p_\mathrm{h}p_\mathrm{n}$, and $\bar{q} = 1 - q$ when $q \in [0, 1]$.

$\mathcal{A}^{(0)}$ has two states that correspond to the on and off states of the data source. Transitions in this automaton happen independently of the other automata. Hence, we have

$$P_{e_0}^{(0)} = P_{e_1}^{(0)} = P_{e_2}^{(0)} = \begin{bmatrix} \bar{\beta} & \beta \\ \alpha & \bar{\alpha} \end{bmatrix}.$$

The current TDMA frame is modeled by $\mathcal{A}^{(1)}$ that has $(C+1)$ states. If $s\mathcal{A}^{(1)} = i$, then the current TDMA frame has i active CBR connections. The contribution of $\mathcal{A}^{(1)}$ to synchronization e_a, $a \in \{0, 1, 2\}$, is given by the matrix $P_{e_a}^{(1)}$ of order $(C + 1)$ with ijth element

$$P_{e_a}^{(1)}(i, j) = \begin{cases} \gamma_a \binom{i+a}{i+a-j} p_s^{i+a-j}(1-p_s)^j, & i + a \le C, 0 \le i + a - j < M, \\ \gamma_a \binom{C}{C-j} p_s^{C-j}(1-p_s)^j, & i + a > C, 0 \le i + a - j < M, \\ \gamma_a \sum_{k=j}^{i+a} \binom{i+a}{i+a-k} p_s^{i+a-k}(1-p_s)^k, & i + a \le C, i + a - j = M, \\ \gamma_a \sum_{k=j}^{C} \binom{C}{C-k} p_s^{C-k}(1-p_s)^k, & i + a > C, i + a - j = M, \\ 0 & \text{otherwise} \end{cases}$$

for $i, j = 0, 1, \ldots, C$. $P_{e_a}^{(1)}(i, j)$ is the probability that the number of CBR connections in the current TDMA frame changes by $(j - i)$ when a CBR calls arrive. When all a arriving CBR calls can be accommodated in the TDMA frame (i.e., $i + a \le C$), the departing CBR calls $(i + a - j)$ are chosen out of $(i + a)$ calls. The additional condition $0 \le i + a - j < M$ is exercised to make sure that the number of CBR calls in the current TDMA frame cannot increase by more than a and cannot decrease by more than M CBR calls in the frame duration. When all of the a arriving CBR calls cannot be accommodated in the current TDMA frame (i.e., $i + a > C$), the excessive $(C - i - a)$ CBR calls are rejected. Hence, the departing $(C - j)$ CBR calls are chosen out of C CBR calls (see second line in the expression for $P_{e_a}^{(1)}(i, j)$). The third and the fourth lines in the expression for $P_{e_a}^{(1)}(i, j)$ are for the case when M CBR calls depart in the frame duration. Since the departure of CBR calls is modeled as a truncated binomial process, the probability that M CBR calls depart in a given TDMA frame is equal to the probability that M and more (i.e., up to $(i + a)$ if $i + a \le C$ or up to C if $i + a > C$) CBR calls depart in the frame duration. In summary, each $P_{e_a}^{(1)}$, $a = 0, 1, 2$, is a banded matrix with M diagonals below and a diagonals above the main diagonal.

The data buffer is modeled by $\mathcal{A}^{(2)}$ that has $(B + 1)$ states. If $s\mathcal{A}^{(2)} = i$, then the buffer has i data packets. Transitions of this automaton depend on the state of the data source, $\mathcal{A}^{(0)}$, and the state of $\mathcal{A}^{(1)}$.

For synchronization e_a, $P_{e_a}^{(2)}$ is given by

$$\begin{bmatrix}
g_1(0,a) & g_0(-1,a) & g_0(-2,a) & g_0(-3,a) & & & & & & & \\
g_1(1,a) & g_0(0,a) & g_0(-1,a) & g_0(-2,a) & g_0(-3,a) & & & & & & \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots & & & & & \\
g_1(C,a) & g_0(C-1,a) & g_0(C-2,a) & g_0(C-3,a) & g_0(C-4,a) & \cdots & g_0(-3,a) & & & & \\
 & g_0(C,a) & g_0(C-1,a) & g_0(C-2,a) & g_0(C-3,a) & g_0(C-4,a) & \cdots & g_0(-3,a) & & & \\
 & & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & & \\
 & & g_0(C,a) & g_0(C-1,a) & g_0(C-2,a) & g_0(C-3,a) & g_0(C-4,a) & \cdots & g_0(-3,a) & & \\
 & & & g_0(C,a) & g_0(C-1,a) & g_0(C-2,a) & g_0(C-3,a) & \cdots & g_0(-2,a) & g_2(3,a) & \\
 & & & & g_0(C,a) & g_0(C-1,a) & g_0(C-2,a) & \cdots & g_0(-1,a) & g_2(2,a) & \\
 & & & & & g_0(C,a) & g_0(C-1,a) & \cdots & g_0(0,a) & g_2(1,a) & \\
 & & & & & & g_0(C,a) & \cdots & g_0(1,a) & g_2(0,a) &
\end{bmatrix}.$$

The function $g_0(l,a)$ is defined as

$$g_0(l,a) = \begin{cases} p_d(FS(a)-l), & s\mathcal{A}^{(0)} = 1, 0 \le (FS(a)-l) \le 3, \\ 1, & s\mathcal{A}^{(0)} = 0, FS(a) = l, \\ 0 & \text{otherwise}, \end{cases}$$

where $FS(a) = [C - (s\mathcal{A}^{(1)} + a)]^+$ and $[x]^+ = \max(x, 0)$. Note that $FS(a)$ denotes the number of free slots in the current TDMA frame taking into account up to $a \in \{0, 1, 2\}$ possible CBR arrivals. The parameter l in g_0 corresponds to the difference between the number of packets admitted to the data buffer and the number of packets that departed from the data buffer in the TDMA frame duration. Note that l takes values from $\{-3, -2, \ldots, C\}$. Finally, we have $g_1(i,a) = \sum_{l=i}^{C} g_0(l,a)$, where $i \in \{0, 1, \ldots, C\}$, and $g_2(j,a) = \sum_{l=i}^{3} g_0(-l,a)$, where $j \in \{0, 1, 2, 3\}$.

The stochastic one-step transition probability matrix of the underlying MC is given by

$$R = \sum_{a=0}^{2} \bigotimes_{k=0}^{2} P_{e_a}^{(k)}.$$

2.2. An SAN model for VBR traffic

In this section, we consider an SAN model of a wireless ATM system that accepts VBR calls. We assume that an arrival associated with VBR traffic can be either a new call or a handover just like CBR arrivals. Handover VBR requests have priority over new VBR calls and they, respectively, arrive with probabilities p_{vh} and p_{nh}. We do not consider multiple handover or new VBR call arrivals during a TDMA frame since the associated probabilities with these events are small.

A VBR connection is characterized by a state of high intensity and a state of low intensity. In the state of high intensity, the VBR source transmits data with its highest rate, whereas in the state of low intensity, its transmission rate is reduced. The reduced transmission rate in the low intensity state in fact means that at some instances of time the slot in a TDMA frame allocated to the VBR connection will not be used. Hence, slots reserved for VBR traffic in a TDMA frame can be used by ABR traffic in two ways. First, a

slot may have been reserved for VBR traffic for which there is no active VBR connection. Second, there is a VBR connection associated with the given slot, but the connection is in the low intensity state and nothing is transmitted during this TDMA frame.

A VBR connection moves from the state of high intensity to the state of low intensity with probability α_v and from the state of low intensity to the state of high intensity with probability β_v. In terms of $\lambda_v = \beta_v/(\alpha_v + \beta_v)$ and its square coefficient of variation S_{C_v}, we have $\beta_v = 2\lambda_v(1-\lambda_v)/(S_{C_v} + 1 - \lambda_v)$ and $\alpha_v = \beta_v(1-\lambda_v)/\lambda_v$ (see Section 2). For the low intensity state, we introduce the parameters p_{empty} and p_{busy}. The former is the probability of transition from the state when the slot allocated to the VBR connection is busy to the state when the slot is empty. The latter is the probability of transition from the empty state to the busy state. We assume that when a VBR connection (either a new call or a handover) is set up, it is in the high intensity state. On the other hand, the connection can terminate in any state of the VBR source. We also assume that when a VBR connection changes its state from high intensity to low intensity, it enters the busy state. The number of VBR calls that may terminate in a given TDMA frame depends on the number of active VBR calls and the duration of each VBR call is assumed to be a geometric process with parameter p_{vs}. VBR arrivals to the system are assumed to happen at the beginning of a TDMA frame, and state changes of a VBR connection after it is set up are assumed to take place at the end of a frame.

The performance measures of interest are the dropping probability of handover VBR calls and the blocking probability of new VBR calls. We model each slot reserved for VBR traffic in the current TDMA frame by a single automaton of four states. State 0 of the automaton corresponds to the case of an idle slot, i.e., the VBR connection is not active. State 1 corresponds to the state of high intensity, states 2 and 3 correspond to the state of low intensity. In particular, state 2 indicates that the slot is busy and state 3 indicates that it is empty.

Similar to CBR arrivals, in a given TDMA frame we can have at most two VBR requests arriving simultaneously. Therefore, we define the three events f_0, f_1, f_2 that correspond to, respectively, 0, 1, 2 VBR arrivals. Note that VBR calls arrive to the system but not to a particular slot. Hence, if a TDMA frame has V slots reserved for VBR traffic numbered from 0 to $V-1$, the transition probability matrix P_{f_b} that corresponds to event f_b, $b \in \{0, 1, 2\}$, is given by

$$P_{f_b} = \widetilde{\gamma}_b \bigotimes_{k=0}^{V-1} P_{f_b}^{(k)} = \widetilde{\gamma}_b P_{f_b}^{(0)} \otimes \left(\bigotimes_{k=1}^{V-1} P_{f_b}^{(k)} \right).$$

Here $P_{f_b}^{(k)}$ is the contribution of $\mathcal{A}^{(k)}$ (corresponding to the kth VBR slot) to event f_b, $\widetilde{\gamma}_b$ is the probability of b VBR arrivals during the current TDMA frame, and $\widetilde{\gamma}_0 = \bar{p}_{vn}\bar{p}_{vh}$, $\widetilde{\gamma}_1 = p_{vn}\bar{p}_{vh} + p_{vh}\bar{p}_{vn}$, and $\widetilde{\gamma}_2 = p_{vn}p_{vh}$. Thus, only one synchronizing event matrix of event f_b needs to be scaled by $\widetilde{\gamma}_b$. Hence, for convenience we define

$$P_{f_b}^{(k)} = \begin{cases} \widetilde{\gamma}_b P_b^{(k)}, & k = 0, \\ P_b^{(k)}, & 0 < k < V, \end{cases}$$

where $P_b^{(k)}$ is the transition probability matrix describing the evolution of the kth TDMA slot when there are b VBR arrivals.

The transition probability matrix $P_0^{(k)}$ is given by

$$P_0^{(k)} = \begin{bmatrix} 1 & & & \\ p_{\mathrm{vs}} & \bar{p}_{\mathrm{vs}}\bar{\alpha}_{\mathrm{v}} & \bar{p}_{\mathrm{vs}}\alpha_{\mathrm{v}} & \\ p_{\mathrm{vs}} & \bar{p}_{\mathrm{vs}}\beta_{\mathrm{v}} & \bar{p}_{\mathrm{vs}}\bar{\beta}_{\mathrm{v}}\bar{p}_{\mathrm{empty}} & \bar{p}_{\mathrm{vs}}\bar{\beta}_{\mathrm{v}}p_{\mathrm{empty}} \\ p_{\mathrm{vs}} & \bar{p}_{\mathrm{vs}}\beta_{\mathrm{v}} & \bar{p}_{\mathrm{vs}}\bar{\beta}_{\mathrm{v}}p_{\mathrm{busy}} & \bar{p}_{\mathrm{vs}}\bar{\beta}_{\mathrm{v}}\bar{p}_{\mathrm{busy}} \end{bmatrix}.$$

When a single VBR request arrives to the system, only one automaton among those that are in the idle state (i.e., state 0) should change its state. We choose the automaton that is in the idle state and that has the smallest index. If all V automata are in active states (i.e., there are V active VBR connections), the incoming VBR call is rejected. Observe that if an automaton is in one of the three active states (i.e., states 1–3), the transition probabilities out of the active state are the same as those for zero VBR arrivals (see rows 2–4 of matrix $P_0^{(k)}$). Thus, the transition probability matrix $P_1^{(k)}$ is given by

$$P_1^{(k)} = \begin{bmatrix} 1 - g_3(k)\bar{p}_{\mathrm{vs}} & g_3(k)\bar{p}_{\mathrm{vs}}\bar{\alpha}_{\mathrm{v}} & g_3(k)\bar{p}_{\mathrm{vs}}\alpha_{\mathrm{v}} & \\ p_{\mathrm{vs}} & \bar{p}_{\mathrm{vs}}\bar{\alpha}_{\mathrm{v}} & \bar{p}_{\mathrm{vs}}\alpha_{\mathrm{v}} & \\ p_{\mathrm{vs}} & \bar{p}_{\mathrm{vs}}\beta_{\mathrm{v}} & \bar{p}_{\mathrm{vs}}\bar{\beta}_{\mathrm{v}}\bar{p}_{\mathrm{empty}} & \bar{p}_{\mathrm{vs}}\bar{\beta}_{\mathrm{v}}p_{\mathrm{empty}} \\ p_{\mathrm{vs}} & \bar{p}_{\mathrm{vs}}\beta_{\mathrm{v}} & \bar{p}_{\mathrm{vs}}\bar{\beta}_{\mathrm{v}}p_{\mathrm{busy}} & \bar{p}_{\mathrm{vs}}\bar{\beta}_{\mathrm{v}}\bar{p}_{\mathrm{busy}} \end{bmatrix},$$

where

$$g_3(k) = \begin{cases} 1, & s\mathcal{A}^{(k)} = 0 \text{ and } \sum_{l=0}^{k-1} \tilde{s}\mathcal{A}^{(l)} = k, \\ 0 & \text{otherwise}, \end{cases} \qquad \tilde{s}\mathcal{A}^{(l)} = \begin{cases} 0, & s\mathcal{A}^{(l)} = 0, \\ 1 & \text{otherwise}. \end{cases}$$

The difference between events f_1 and f_2 is that when two VBR requests arrive, two automata among those that are in the idle state should change their states. Hence, we only have to redefine the function $g_3(k)$. The new function $g_4(k)$ should return 1 for the two automata that are in the idle state and that have the smallest indices

$$g_4(k) = \begin{cases} 1, & s\mathcal{A}^{(k)} = 0 \text{ and } \sum_{l=0}^{k-1} \tilde{s}\mathcal{A}^{(l)} = k \text{ or } \sum_{l=0}^{k-1} \tilde{s}\mathcal{A}^{(l)} = k-1, \\ 0 & \text{otherwise}. \end{cases}$$

Thus, the transition probability matrix $P_2^{(k)}$ is $P_1^{(k)}$ in which $g_3(k)$ is replaced by $g_4(k)$.

Similar to the basic SAN model, the stochastic one-step transition probability matrix of the underlying MC is given by

$$Q = \sum_{b=0}^{2} \bigotimes_{k=0}^{V-1} P_{f_b}^{(k)}.$$

2.3. The combined SAN model

In this section, we finalize the SAN model of the wireless ATM system by combining the SAN models for CBR and VBR calls considered in the previous two sections. The SAN model of the combined system consists of $(3 + V)$ automata. The first three are the automata of the basic SAN model and the last V are

the automata dedicated to VBR arrivals. We remark that the automata of the SAN that handle CBR and VBR arrivals are mutually independent. Hence, in the combined model the set of events that correspond to new call and handover arrivals is given by the Cartesian product $E_{\mathrm{CBR}} \times E_{\mathrm{VBR}}$, where $E_{\mathrm{CBR}} = \{e_0, e_1, e_2\}$ and $E_{\mathrm{VBR}} = \{f_0, f_1, f_2\}$. Therefore, we denote by s_{ab} the event of a CBR arrivals and b VBR arrivals. Then the synchronizing transition probability matrix of automaton $\mathcal{A}^{(k)}$, $k \in \{0, 1, \ldots, V+2\}$, and event s_{ab} for $a, b \in \{0, 1, 2\}$ is given by

$$P_{s_{ab}}^{(k)} = \begin{cases} P_{e_a}^{(k)}, & 0 \le k < 3, \\ P_{f_b}^{(k-3)}, & 3 \le k \le V+2 \end{cases}$$

under the following conditions.

Each TDMA frame of the combined system that gives CBR, VBR, and ABR service consists of C slots reserved for CBR traffic and V slots reserved for VBR traffic. ABR traffic can be pushed into any reserved, but unused slots. Hence, data packets can be transmitted in the idle slots among the C reserved for CBR traffic, in the idle slots among the V reserved for VBR traffic, and in those slots among the V that are in the empty state. Thus, the function FS should be redefined as follows:

$$FS(a,b) = [C - (s\mathcal{A}^{(1)} + a)]^+ + \left[V - \left(\sum_{k=3}^{V+2} \delta(s\mathcal{A}^{(k)} = 1) + \sum_{k=3}^{V+2} \delta(s\mathcal{A}^{(k)} = 2) + b \right) \right]^+,$$

where a and b are the indices of event s_{ab} and δ denotes the Kronecker delta.

Since data packets can use effectively a maximum of $(C+V)$ slots, each C in the matrix $P_{e_a}^{(2)}$ should be replaced by $(C+V)$. Furthermore, functions g_0, g_1, and g_2 should be redefined as follows:

$$g_0(l,a,b) = \begin{cases} p_d(FS(a,b) - l), & s\mathcal{A}^{(0)} = 1, 0 \le (FS(a,b) - l) \le 3, \\ 1, & s\mathcal{A}^{(0)} = 0, FS(a,b) = l, \\ 0 & \text{otherwise}, \end{cases}$$

$g_1(i,a,b) = \sum_{l=i}^{C+V} g_0(l,a,b)$, $g_2(j,a,b) = \sum_{l=i}^{3} g_0(-l,a,b)$, where a and b are the indices of event s_{ab}, $l \in \{-3, -2, \ldots, C+V\}$, $i \in \{0, 1, \ldots, C+V\}$, and $j \in \{0, 1, 2, 3\}$.

Finally, the underlying MC of the combined system is given by

$$P = \sum_{a=0}^{2} \sum_{b=0}^{2} \bigotimes_{k=0}^{V+2} P_{s_{ab}}^{(k)}.$$

Observe that each matrix $P_{s_{ab}}^{(k)}$ has equal row sums. In particular, for $a, b = 0, 1, 2$, each $P_{s_{ab}}^{(k)}$, $k \in \{0, 2\} \cup \{4, 5, \ldots, V+2\}$, has row sums of 1; each $P_{s_{ab}}^{(1)}$ has row sums of γ_a; and each $P_{s_{ab}}^{(3)}$ has row sums of $\widetilde{\gamma}_b$. Hence, from Proposition A1 in [30], each matrix $\bigotimes_{k=0}^{V+2} P_{s_{ab}}^{(k)}$ has row sums of $\gamma_a \widetilde{\gamma}_b$. Since $\sum_{a=0}^{2} \gamma_a = 1$ and $\sum_{b=0}^{2} \widetilde{\gamma}_b = 1$, P has row sums of 1.

Now, we comment on the dependencies among the automata of the SAN model. Evolution of $\mathcal{A}^{(0)}$, which corresponds to the data source, does not depend on the states of other automata and the events of the SAN model. Evolution of $\mathcal{A}^{(1)}$, which corresponds to the C TDMA slots reserved for CBR calls, depends on the events of the SAN model, but does not depend on the states of other automata. Evolution of VBR automata $\mathcal{A}^{(v)}$, $v = 3, 4, \ldots, V+2$, depends on the states of automata $k = 3, 4, \ldots, v-1$ and the

events of the SAN, but does not depend on $\mathcal{A}^{(0)}$, $\mathcal{A}^{(1)}$, $\mathcal{A}^{(2)}$. Finally, evolution of $\mathcal{A}^{(2)}$ corresponding to the data buffer depends on the states of all the other automata and the events of the SAN. The dependencies among automata show that parts of the model corresponding to CBR calls and VBR calls can be analyzed separately from each other, from $\mathcal{A}^{(0)}$, and from $\mathcal{A}^{(2)}$. On the other hand, analyses of performance indices related to data packets requires the consideration of the combined system, since arrival of data packets depends on the state of the data source and data packets can be transmitted in the TDMA slots reserved for CBR and VBR calls.

3. Analysis of a class of lumpable discrete-time SANs

The discrete-time SAN model of the combined system in Section 2 has $(3+V)$ automata and nine events. The first three automata, respectively, have 2, $(C+1)$, $(B+1)$ states and the last V automata each have four states giving us a global state space size of $n = 2(C+1)(B+1)4^V$. The automaton $\mathcal{A}^{(2)}$ that models the data buffer depends on all the other automata, and each automaton $\mathcal{A}^{(k)}$, $k \in \{4, 5, \ldots, V+2\}$, that models VBR traffic depends on the automata $\mathcal{A}^{(3)}$, $\mathcal{A}^{(4)}, \ldots, \mathcal{A}^{(k-1)}$. The probability matrices associated with the automata are all relatively dense except the ones that correspond to the data buffer when $s\mathcal{A}^{(0)} = 0$.

Now, we are in a position to consider an example and discuss its implications. We set $\lambda = 0.1$, $S_C = 1$, $(p_d(0), p_d(1), p_d(2), p_d(3)) = (0.05, 0.1, 0.25, 0.6)$ (amounting to an average of $\rho = 2.5$ packet arrivals during a TDMA frame), $(p_{\mathrm{n}}, p_{\mathrm{h}}, p_s) = C(5\times10^{-6}, 10^{-5}, 5\times10^{-6})$, $\lambda_{\mathrm{v}} = 0.5$, $S_{C_{\mathrm{v}}} = 10$, $(p_{\mathrm{empty}}, p_{\mathrm{busy}}) = (0.9, 0.1)$, $(p_{\mathrm{vn}}, p_{\mathrm{vh}}, p_{\mathrm{vs}}) = V(5\times10^{-6}, 10^{-5}, 5\times10^{-6})$. For the problem $(C, V, B) = (8, 2, 15)$ with at most two $(=M)$ CBR departures during a TDMA frame, we have $n = 4608$ and $nz = 1{,}618{,}620$ (number of nonzeros larger than 10^{-16} is 1,174,657). Here nz denotes the number of nonzeros in the underlying DTMC. In the problem $(C, V, B) = (12, 3, 15)$ with $M = 3$, we have $n = 26{,}624$ and $nz = 39{,}042{,}922$ (number of nonzeros larger than 10^{-16} is 19,979,730). For the larger problem $(C, V, B) = (16, 4, 15)$ with $M = 4$, we have $n = 139{,}264$, but are not able to determine its number of nonzeros in a reasonable amount of time. Hence, in this problem, we not only have state space explosion, but we also have a relatively dense global DTMC hindering performance analysis by conventional techniques. However, the situation is not hopeless.

3.1. Ordinary lumpability

Let $A = \bigotimes_{k=0}^{N-1} A^{(k)}$, where each $A^{(k)}$, $k = 0, 1, \ldots, N-1$, is a square matrix of order n_k. Similar to the global state of an SAN, to row i of A corresponds the vector $(rA^{(0)}, rA^{(1)}, \ldots, rA^{(N-1)})$ (i.e., $i \leftrightarrow (rA^{(0)}, rA^{(1)}, \ldots, rA^{(N-1)})$), where $rA^{(k)}$ denotes the row index of $A^{(k)}$, and to column j of A corresponds the vector $(cA^{(0)}, cA^{(1)}, \ldots, cA^{(N-1)})$, where $cA^{(k)}$ denotes the column index of $A^{(k)}$. From the definition of tensor product [10, p. 117], for any $m \in \{1, 2, \ldots, N-1\}$ the matrix A can be partitioned into K^2 blocks of the same order as

$$A = \begin{pmatrix} A_{11} & A_{12} & \ldots & A_{1K} \\ A_{21} & A_{22} & \ldots & A_{2K} \\ \vdots & \vdots & \ddots & \vdots \\ A_{K1} & A_{K2} & \ldots & A_{KK} \end{pmatrix}, \tag{3}$$

where $K = \prod_{k=0}^{m-1} n_k$,

$$A_{ij} = \xi_{ij} \bigotimes_{k=m}^{N-1} A^{(k)} = \left(\prod_{k=0}^{m-1} A^{(k)}(rA^{(k)}, cA^{(k)}) \right) \bigotimes_{k=m}^{N-1} A^{(k)}, \tag{4}$$

$i \leftrightarrow (rA^{(0)}, rA^{(1)}, \ldots, rA^{(m-1)})$, and $j \leftrightarrow (cA^{(0)}, cA^{(1)}, \ldots, cA^{(m-1)})$. Now, let us assume that the matrices $A^{(k)}$ may have functional elements such that the value of the function depends on the row index of A.

Theorem 1. *Each block A_{ij} in Eq. (3) of the partitioning corresponding to $m \in \{1, 2, \ldots, N-1\}$ and an ordering of the matrices $A^{(k)}$, $k = 0, 1, \ldots, N-1$, has equal row sums if $A^{(l)}$, $l = m, m+1, \ldots, N-1$, has equal row sums and for any pair (k, l), $k \in \{0, 1, \ldots, m-1\}$ and $l \in \{m, m+1, \ldots, N-1\}$, functional elements of $A^{(k)}$ do not depend on $A^{(l)}$.*

Proof. We must show in Eq. (3) that $A_{ij}u = \alpha_{ij}u$ for $i, j = 1, 2, \ldots, K$, where α_{ij} is a constant value that depends on i, j, m, and u represents the column vector of 1's with appropriate length. We are dropping m from α_{ij} since m is fixed for the chosen partitioning. The value $A^{(k)}(rA^{(k)}, cA^{(k)})$ in Eq. (4), where $k \in \{0, 1, \ldots, m-1\}$, may be a function of $rA^{(l)}$ for some $l \in \{0, 1, \ldots, m-1\}$. However, from the statement of the theorem, $A^{(k)}(rA^{(k)}, cA^{(k)})$ does not depend on $rA^{(l)}$ for any $l \in \{m, m+1, \ldots, N-1\}$. Hence, for the particular mapping $i \leftrightarrow (rA^{(0)}, rA^{(1)}, \ldots, rA^{(m-1)})$, $A^{(k)}(rA^{(k)}, cA^{(k)})$ and consequently ξ_{ij} in Eq. (4) are well-defined constants.

From the statement of the theorem, each $A^{(l)}$, $l = m, m+1, \ldots, N-1$, has equal row sums. Hence, $A^{(l)}u = \nu^{(l)}u$ for $l = m, m+1, \ldots, N-1$ and for some constant value $\nu^{(l)}$ that depends only on l. Then, from Eq. (4)

$$\left(\xi_{ij} \bigotimes_{l=m}^{N-1} A^{(l)} \right) u = \xi_{ij} \bigotimes_{l=m}^{N-1} (A^{(l)} u_{n_l}) = \xi_{ij} \bigotimes_{l=m}^{N-1} (\nu^{(l)} u_{n_l}) = \left(\xi_{ij} \prod_{l=m}^{N-1} \nu^{(l)} \right) u = \alpha_{ij} u,$$

where u_{n_l} denotes the column vector of n_l 1's. □

Observe that the requirement of Theorem 1 regarding functional dependencies among the matrices $A^{(k)}$ is satisfied if the directed graph $G(\mathcal{V}, \mathcal{E})$, in which vertex $v_k \in \mathcal{V}$ represents $A^{(k)}$ and the edge $(v_k, v_l) \in \mathcal{E}$ when elements in $A^{(k)}$ functionally depend on $rA^{(l)}$, has more than one strongly connected component (SCC). Next, we state a result that extends Theorem 1 to a square matrix given as the sum of E tensor products.

Corollary 1. *If there exists the same value of m for which each tensor product $\bigotimes_{k=0}^{N-1} B_e^{(k)}$ in $B = \sum_{e=0}^{E-1} \bigotimes_{k=0}^{N-1} B_e^{(k)}$, where $B_e^{(k)}$ is of order n_k for $e = 0, 1, \ldots, E-1$, satisfies the conditions of Theorem 1, then each block B_{ij}, $i, j = 1, 2, \ldots, K$, in the partitioning of B specified by m as in Eq. (3) has equal row sums.*

When B is a stochastic matrix that satisfies the conditions of Corollary 1, B is ordinarily lumpable.

We remark that the application of Corollary 1 to discrete-time SANs whose descriptors are given by Eq. (2) is straightforward. Therefore, we concentrate on Eq. (1). This equation can be considered as a sum of the terms $P_L = \bigotimes_{k=0}^{N-1} L^{(k)}$, $P_R = \sum_{s=0}^{S-1} \bigotimes_{k=0}^{N-1} R_s^{(k)}$ and $P_N = -\sum_{s=0}^{S-1} \bigotimes_{k=0}^{N-1} N_s^{(k)}$. Note that P_N is a

diagonal matrix and influences only the diagonal elements of P. Hence, if we show that each off-diagonal block of a partitioning of P has equal row sums, then each diagonal block of the same partitioning will also have equal row sums since P is a stochastic matrix.

Consider now the term P_L. Recall that each $L^{(k)}$, $k = 0, 1, \ldots, N-1$, has equal row sums. Hence, Theorem 1 applies to P_L if the digraph $G(\mathcal{V}, \mathcal{E})$ associated with the matrices $L^{(k)}$ has more than one SCC. Assuming that for some m the conditions of Theorem 1 are satisfied by the matrices $L^{(k)}$, the conditions of Corollary 1 for the same m must be satisfied by the matrices $R_s^{(k)}$. Unfortunately, synchronizing transition probability matrices need not have equal row sums. Furthermore, the condition regarding functional dependencies may not hold for them either. For instance, the discrete-time SAN model that appears in [30] which has a descriptor as in Eq. (1) does not satisfy Corollary 1 due to the requirement regarding functional dependencies among automata. The example of the SAN model therein has two automata whose matrices satisfy the equal row sums property. However, functional transitions of the third and the fourth automata depend on the states of the first and the second automata. On the other hand, our research on continuous-time SANs whose descriptors have a very similar form to that in Eq. (1) have revealed that some of the existing SAN models are ordinarily lumpable. For instance, the underlying MCs of the three queues problem [15] and the mass storage problem [12] are ordinarily lumpable. We remark that this has not been noticed before.

A particular case that satisfies the conditions of Corollary 1 for a descriptor of the form in Eq. (1) is when a subset of automata do not participate in the synchronizing events of the SAN and the remaining automata do not functionally depend on the subset. For instance, if $\mathcal{A}^{(l)}$ is not involved in any of the synchronizing events, that is $R_s^{(l)} = L^{(l)}$ for $s = 0, 1, \ldots, S-1$, and if for any $k \in \{0, 1, \ldots, N-1\}$, $k \neq l$, $\mathcal{A}^{(k)}[\mathcal{A}^{(l)}]$ does not hold, then the SAN is ordinarily lumpable with respect to the states of $\mathcal{A}^{(l)}$. However, note that not being involved in any of the synchronizing events, does not mean that $\mathcal{A}^{(l)}$ is independent of the other automata and can be analyzed separately. Functional transitions may very well be present in the matrices of $\mathcal{A}^{(l)}$ with values that depend on the states of other automata.

Consider now the application of Corollary 1 to the SAN model of Section 2.3. First, we remark that each $P_{s_{ab}}^{(k)}$, $k = 0, 1, \ldots, V+2$ and $a, b = 0, 1, 2$, has equal row sums. Second, for any ordering of automata in which the data buffer automaton is placed in the last position and the VBR automata are placed in any position other than the last as long as they are ordered according to increasing index among themselves, there exists $m \in \{1, 2, \ldots, V+2\}$ for which the requirements of Corollary 1 are satisfied. Thus, P of the combined SAN model is ordinarily lumpable and there are $(V+2)$ lumpable partitionings for each ordering of automata in which the data buffer automaton is placed in the last position and VBR automata are ordered according to increasing index among themselves. Examples of discrete-time SANs that are ordinarily lumpable and have descriptors as in Eq. (2) can be also found in [19,38]. Matrices of the automata of the SAN introduced in [29] satisfy the equal row sums property but each automaton functionally depends on the others. Hence, Corollary 1 does not apply and the SAN is not lumpable with respect to any partitioning in Eq. (3).

The ordinary lumpability of SANs discussed in this section is a special case of exact aggregation considered in [3] where a subset of states of an automaton that satisfy lumpability conditions is aggregated. However, in contrast to the aggregation of SANs discussed in [3], Corollary 1 applies to SANs whose descriptors are composed of generalized tensor products. But most importantly, in the partitioning of Eq. (3), all blocks are of the same order and can be easily obtained from the tensor representation of the SAN descriptor. We take advantage of this in the efficient AID algorithm for ordinarily lumpable SANs which is introduced in the next subsection.

3.2. Aggregation–iterative disaggregation

Assuming that P is ordinarily lumpable with respect to the partitioning in (3) and is irreducible, we propose Algorithm 1 to compute its stationary probability vector π that satisfies $\pi P = \pi$, $\|\pi\| = 1$. At the end of this subsection, we discuss how to proceed with Algorithm 1 when the MC to be analyzed is reducible.

Algorithm 1 (AID algorithm for lumpable discrete-time SANs).

1. Let $\pi^{(0)} = (\pi_1^{(0)}, \pi_2^{(0)}, \ldots, \pi_K^{(0)})$ be a given initial approximation of π. Set it $= 1$.
2. Aggregation:
   (a) Compute the lumped matrix L of order K with ijth element $l_{ij} = e_1^T(P_{ij}u)$.
   (b) Solve the singular system $\tau(I - L) = 0$ subject to $\|\tau\|_1 = 1$ for $\tau = (\tau_1, \tau_2, \ldots, \tau_K)$.
3. Disaggregation:
   (a) Compute the row vector
   $$z^{(\mathrm{it})} = \left(\tau_1 \frac{\pi_1^{(\mathrm{it}-1)}}{\|\pi_1^{(\mathrm{it}-1)}\|_1}, \tau_2 \frac{\pi_2^{(\mathrm{it}-1)}}{\|\pi_2^{(\mathrm{it}-1)}\|_1}, \ldots, \tau_K \frac{\pi_K^{(\mathrm{it}-1)}}{\|\pi_K^{(\mathrm{it}-1)}\|_1} \right).$$
   (b) Solve the K nonsingular systems of which the ith element is given by
   $$\pi_i^{(\mathrm{it})}(I - P_{ii}) = b_i^{(\mathrm{it})}$$
   for $\pi_i^{(\mathrm{it})}$, $i = 1, 2, \ldots, K$, where $b_i^{(\mathrm{it})} = \sum_{j>i} z_j^{(\mathrm{it})} P_{ji} + \sum_{j<i} \pi_j^{(\mathrm{it})} P_{ji}$.
4. Test $\pi^{(\mathrm{it})}$ for convergence. If the desired accuracy is attained, then stop and take $\pi^{(\mathrm{it})}$ as the stationary probability vector of P. Else set it $=$ it $+ 1$ and go to step 3.

Algorithm 1, which we name as AID, is a modified form of Koury–McAllister–Stewart's (KMS) IAD algorithm [34]. To the best of our knowledge, the implementation of the IAD algorithm of KMS for SANs have not appeared in the literature. The existing aggregation–disaggregation algorithm for SANs discussed in [4] utilizes a different approach in which aggregation at each iteration is done with respect to the states of an automaton chosen adaptively. We remark also that in the experiments of Buchholz [4] the disaggregation phase of the algorithm is a power iteration, which is inferior to BGS since BGS is a preconditioned power iteration in which the preconditioning matrix is the block lower-triangular part of the coefficient matrix. Recent results [14] on the computation of the stationary vector of MCs show that IAD and BGS with judiciously chosen partitionings mostly outperform incomplete LU (ILU) preconditioned projection methods. Furthermore, BGS, which forms the disaggregation phase of IAD, when used with partitionings having blocks of equal order is likely to outperform IAD when the problem at hand is not ill-conditioned. Therefore, we propose AID rather than BGS for discrete-time SANs since the partitioning in (3) is a balanced one with equal order of blocks and the aggregate matrix needs to be formed only once due to lumpability. In this way, we not only use a balanced partitioning but we also incorporate to our algorithm the gain obtained from being able to exactly solve the coupling matrix in IAD (i.e., lumped matrix).

In Algorithm 1, the lumped matrix L of order $K = \prod_{k=0}^{m-1} n_k$ is computed at the outset and solved once for its stationary vector τ. In step 2(a) of the algorithm, the elements of the vector $P_{ij}u$ are all equal; hence,

we choose its first element. See the product $e_1^T(P_{ij}u)$, where e_1 is the column vector of length $\prod_{k=m}^{N-1} n_k$ in which the first element is equal to 1 and the other elements are zero. Note that the lumped matrix L is also lumpable if $m > 1$. These hint at the solver to be chosen at the aggregation phase to compute τ. Assuming that L is dense, one may opt for a direct solver such as Gaussian elimination (GE) (or the method of Grassmann–Taksar–Heyman (GTH) if L is relatively ill-conditioned) when K is on the order of hundreds. Else one may use AID with a lumped matrix of order $\prod_{k=0}^{m'-1} n_k$, where $1 < m' < m$, or IAD with a nearly completely decomposable (NCD) [26] partitioning if L is relatively ill-conditioned. In any case, sufficient space must be allocated to store L while executing step 2 of Algorithm 1.

As for the disaggregation phase (i.e., a BGS iteration), we need to look into how the right-hand sides $b_i^{(\mathrm{it})}$ at iteration "it" are computed. First, we remark that it is possible to represent each block P_{ij} of the partitioning given by Eq. (3) as

$$P_{ij} = \sum_{e=0}^{E-1} \xi_{ij}^{(e)} T_i^{(e)}, \quad \text{where } T_i^{(e)} = \bigotimes_{k=m}^{N-1} P_e^{(k)}$$

(cf. Eq. (4)). Second, we have

$$\begin{aligned} b_i^{(\mathrm{it})} &= \sum_{j>i} z_j^{(\mathrm{it})} \left(\sum_{e=0}^{E-1} \xi_{ji}^{(e)} T_j^{(e)} \right) + \sum_{j<i} \pi_j^{(\mathrm{it})} \left(\sum_{e=0}^{E-1} \xi_{ji}^{(e)} T_j^{(e)} \right) \\ &= \sum_{j>i} \sum_{e=0}^{E-1} \xi_{ji}^{(e)} (z_j^{(\mathrm{it})} T_j^{(e)}) + \sum_{j<i} \sum_{e=0}^{E-1} \xi_{ji}^{(e)} (\pi_j^{(\mathrm{it})} T_j^{(e)}) \end{aligned}$$

for $i = 1, 2, \ldots, K$. Since $T_j^{(e)}$ is composed of $(N-m)$ tensor products, the vector–matrix multiplications $z_j^{(\mathrm{it})} T_j^{(e)}$ and $\pi_j^{(\mathrm{it})} T_j^{(e)}$ turn out to be expensive operations even though they are performed using the efficient vector–tensor product multiplication algorithm. Furthermore, they are performed a total of $K(K-1)E$ times during each iteration and constitute the bottleneck of the iterative solver. However, this situation can be improved at the cost of extra storage as we next discuss.

The subvectors $z_j^{(\mathrm{it})} T_j^{(e)}$ and $\pi_j^{(\mathrm{it})} T_j^{(e)}$ in the two summations appear in the computation of multiple $b_i^{(\mathrm{it})}$. Therefore, at iteration "it", these subvectors of length $\prod_{k=m}^{N-1} n_k$ can be computed and stored when they are encountered for the first time for a specific pair of j and e, and then they can be scaled by $\xi_{ji}^{(e)}$ whenever necessary.

For example, when

$$b_1^{(\mathrm{it})} = \sum_{j>1} \sum_{e=0}^{E-1} \xi_{j1}^{(e)} (z_j^{(\mathrm{it})} T_j^{(e)})$$

is sought, we can compute and store each $z_j^{(\mathrm{it})} T_j^{(e)}$ for $j = 3, 4, \ldots, K$ and $e = 0, 1, \ldots, E-1$. The $z_2^{(\mathrm{it})} T_2^{(e)}$ for $e = 0, 1, \ldots, E-1$ are computed but not stored. Then when

$$b_2^{(\mathrm{it})} = \sum_{j>2} \sum_{e=0}^{E-1} \xi_{j2}^{(e)} (z_j^{(\mathrm{it})} T_j^{(e)}) + \sum_{e=0}^{E-1} \xi_{12}^{(e)} (\pi_1^{(\mathrm{it})} T_1^{(e)})$$

is to be computed at the next step, we can scale each stored subvector $z_j^{(\mathrm{it})} T_j^{(e)}$ by $\xi_{j2}^{(e)}$ for $j = 3, 4, \ldots, K$ and $e = 0, 1, \ldots, E-1$. At this step, we also need to compute and store $\pi_1^{(\mathrm{it})} T_1^{(e)}$ for $e = 0, 1, \ldots, E-1$. But these E vectors can overwrite $z_3^{(\mathrm{it})} T_3^{(e)}$ for $e = 0, 1, \ldots, E-1$, which are no longer required. At the next step, when $i = 3$, we need to scale $z_j^{(\mathrm{it})} T_j^{(e)}$ by $\xi_{j3}^{(e)}$ for $j = 4, 5, \ldots, K$ and $e = 0, 1, \ldots, E-1$, and we need to scale $\pi_1^{(\mathrm{it})} T_1^{(e)}$ by $\xi_{j1}^{(e)}$ for $e = 0, 1, \ldots, E-1$. At this step, we also need to compute and store $\pi_2^{(\mathrm{it})} T_2^{(e)}$ for $e = 0, 1, \ldots, E-1$. These E vectors can overwrite $z_4^{(\mathrm{it})} T_4^{(e)}$ for $e = 0, 1, \ldots, E-1$, which are no longer required, and so on. All of this improvement comes at the expense of $E(K-2)$ vectors of length $\prod_{k=m}^{N-1} n_k$, that is roughly E vectors of length n.

There is a tradeoff between the density of P and the advantage that can be gained by storing additional E vectors of length n. In order to improve the performance of Algorithm 1 in step 3(b), the diagonal blocks can be generated and factorized once in the outset and the LU factors can be stored in core memory and then used at each iteration. Hence, when P is relatively dense, the memory required for the lumped matrix in the aggregation phase and for the LU factors in the disaggregation phase is likely to dominate the memory required for the E vectors of length n. On the other hand, when P and blocks of the partitioning are relatively sparse, a smaller number of multiplications are performed when computing the products $z_j^{(\mathrm{it})} T_j^{(e)}$ and $\pi_j^{(\mathrm{it})} T_j^{(e)}$. Hence, computing and storing additional E vectors of length n may not be needed and a straightforward implementation of the algorithm can be used.

In summary, the proposed solver is limited by $\max(K^2, (E+2)n)$ amount of double precision storage assuming that the lumped matrix is stored in two dimensions. The two vectors of length n are used to store the previous and current approximations of the solution.

So far, we have implicitly assumed that the underlying DTMC of the given discrete-time SAN is irreducible. This need not be the case. In fact, we have implemented a state classification (SC) algorithm that classifies the states in the global state space of an SAN into recurrent and transient subsets following [35, pp. 25–26]. In doing this, we use an SCC search algorithm on a digraph with edges whose presence can be taken into consideration on the fly. The SC algorithm is based on an algorithm that finds SCCs of a digraph using depth first search (DFS). The main idea of DFS is to explore all the vertices of the digraph in a recursive manner. Whenever an unvisited vertex is encountered, the algorithm starts exploring its adjacent vertices. When searching for the next unvisited vertex, SC uses the multidimensional representation of the global state of an SAN. A detailed description of the SC algorithm can be found in [21]. We do not consider the case in which there is more than one recurrent class since it hints at a modeling problem. Note that in the presence of transient states, AID can be used if the lumped matrix corresponding to the ordinarily lumpable partitioning is irreducible. One can show that if each subset of states in the lumpable partitioning has at least one recurrent state, then the lumped matrix is irreducible. Finally, we remark that the inhibition of transient states is important in removing redundant computation from iterative solvers.

4. Numerical results

We implemented Algorithm 1 in C++ as part of the software package PEPS [31]. We timed the solver on a Pentium III with 128 MB of RAM under Linux although the experimental framework in most problems could fit into 64 MB. We order the automata as $\mathcal{A}^{(3)}, \mathcal{A}^{(4)}, \ldots, \mathcal{A}^{(V+2)}, \mathcal{A}^{(1)}, \mathcal{A}^{(0)}, \mathcal{A}^{(2)}$ and choose $m = N-2$ for the partitioning in Eq. (3). Hence, each of the K nonsingular systems to be

solved in step 3(b) of Algorithm 1 is of order $2(B+1)$ and the lumped matrix to be solved in step 2 of Algorithm 1 is of order $K = 4^V(C+1)$.

In each experiment, we use a tolerance of 10^{-8} on the approximate error, $\epsilon^{(\mathrm{it})} = \|\pi^{(\mathrm{it})} - \pi^{(it-1)}\|_2$, in step 4 of Algorithm 1. We remark that the approximate residual, $\eta^{(\mathrm{it})} = \|\pi^{(\mathrm{it})} - \pi^{(\mathrm{it})}P\|_2$, turns out to be less than the approximate error upon termination in all our experiments. Furthermore, for all combinations of the integer parameters we considered, there is sufficient space to factorize in sparse format (that is, to apply sparse GE to) the K diagonal blocks in step 3(b) of Algorithm 1 at the outset. Hence, we use sparse forward and back substitutions to solve the K nonsingular systems at each iteration of Algorithm 1. If this had not been the case, we would suggest using point GS as discussed in [37] with a maximum number of 100 iterations and a tolerance of 10^{-3} on the approximate error for the K nonsingular systems at each iteration of Algorithm 1 (see [14]). Regarding the solver for the lumped matrix formed in step 2 of Algorithm 1, we use GTH as discussed in [13] when K is on the order of hundreds. When the lumped matrix is of considerable size and density with a number of nonzeros on the order of millions, we solve it using sparse IAD as discussed in [13] with a tolerance of 10^{-12} on its approximate error. In doing this, when the lumped matrix to be solved is NCD with a small degree of coupling [14], hence ill-conditioned, we employ an NCD partitioning. Otherwise we take advantage of the fact that the lumped matrix is also lumpable (see Section 3.2) and try to use a balanced partitioning by separating the first $(N-2)$ automata in the chosen ordering into two subsets.

As for the performance measures of interest, the dropping probability of handover CBR calls is given by $P_{\mathrm{c_{drop}}} = \|\pi_{s\mathcal{A}^{(1)}=C}\|_1$, whereas the blocking probability of new CBR calls is given by $P_{\mathrm{c_{block}}} = \|\pi_{s\mathcal{A}^{(1)}\geq C-1}\|_1$. By the notation $\|\pi_{\mathrm{condition}}\|_1$, we mean the sum of the stationary probabilities of all states that satisfy *condition*. Similarly, the dropping probability of handover VBR calls is given by $P_{\mathrm{v_{drop}}} = \|\pi_{s\mathcal{A}^{(k)}\neq 0\,\forall k\in\{3,4,\ldots,V+2\}}\|_1$ and the blocking probability of new VBR calls is given by $P_{\mathrm{v_{block}}} = \|\pi_{s\mathcal{A}^{(k)}=0 \text{ for only one } k\in\{3,4,\ldots,V+2\}}\|_1$. Finally, the blocking probability of data packets is given by

$$P_a = P[\text{zero empty slots}] + \frac{(p_d(2)+2p_d(3))P[\text{one empty slot}] + p_d(3)P[\text{two empty slots}]}{\rho},$$

where

$$P[\text{zero empty slots}] = \|\pi_{FS(0,0)=0\wedge s\mathcal{A}^{(2)}=B}\|_1,$$
$$P[\text{one empty slot}] = \|\pi_{(FS(0,0)=1\wedge s\mathcal{A}^{(2)}=B)\vee(FS(0,0)=0\wedge s\mathcal{A}^{(2)}=B-1)}\|_1,$$
$$P[\text{two empty slots}] = \|\pi_{(FS(0,0)=2\wedge s\mathcal{A}^{(2)}=B)\vee(FS(0,0)=1\wedge s\mathcal{A}^{(2)}=B-1)\vee(FS(0,0)=0\wedge s\mathcal{A}^{(2)}=B-2)}\|_1.$$

We remark that $P_{\mathrm{c_{drop}}}$ and $P_{\mathrm{c_{block}}}$ are independent of ABR and VBR traffics. Similarly, $P_{\mathrm{v_{drop}}}$ and $P_{\mathrm{v_{block}}}$ are independent of ABR and CBR traffic. Hence, when $C = V = M$ and the real valued parameters for CBR and VBR traffic are the same, we have $P_{\mathrm{c_{drop}}} = P_{\mathrm{v_{drop}}}$ and $P_{\mathrm{c_{block}}} = P_{\mathrm{v_{block}}}$. In Fig. 1, we set $(p_{\mathrm{n}}, p_{\mathrm{h}}, p_s) = C(5\times10^{-6}, 10^{-5}, 5\times10^{-6})$, $M = C$ and plot $P_{\mathrm{c_{block}}}$ and $P_{\mathrm{c_{drop}}}$ vs. C. We verify that $P_{\mathrm{c_{block}}}$ is larger than $P_{\mathrm{c_{drop}}}$ and that larger C implies smaller $P_{\mathrm{c_{block}}}$ and $P_{\mathrm{c_{drop}}}$.

Next we consider the three problems $(C, V, B) \in \{(8,2,15), (12,3,15), (16,4,15)\}$ that are, respectively, named *small*, *medium*, and *large* (see Section 3). We set $M = V$, $(p_d(0), p_d(1), p_d(2), p_d(3)) = (0.05, 0.1, 0.25, 0.6)$, $(p_{\mathrm{n}}, p_{\mathrm{h}}, p_s) = C(5\times10^{-6}, 10^{-5}, 5\times10^{-6})$, and $(p_{\mathrm{vn}}, p_{\mathrm{vh}}, p_{\mathrm{vs}}) = V(5\times10^{-6}, 10^{-5}, 5\times10^{-6})$. We choose (α, β) so as to satisfy $\lambda \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$ and $S_C \in \{1, 10\}$ (see Section 2). As for the VBR arrival processes, we set $(\alpha_{\mathrm{v}}, \beta_{\mathrm{v}})$ so as to have $\lambda_{\mathrm{v}} = 0.5$ and $S_{C_{\mathrm{v}}} = 10$ (see Section 2.2). Finally, we set $(p_{\mathrm{empty}}, p_{\mathrm{busy}}) = (0.9, 0.1)$ so that the rate of each VBR arrival process in

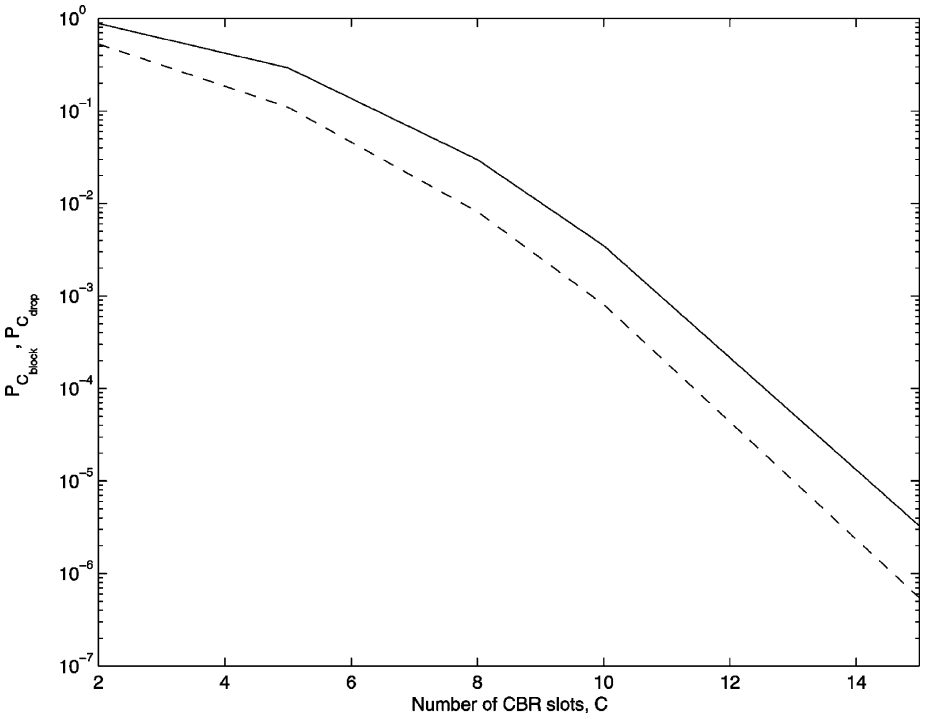

Fig. 1. $P_{c_{block}}$ (solid) and $P_{c_{drop}}$ (dashed) vs. C.

the low intensity state is 10% that of its high intensity state. In Fig. 2, we plot P_a vs. λ for the problems *small*, *medium*, and *large* when (a) $S_C = 1$, (b) $S_C = 10$. We observe that P_a increases with λ and S_C though the increase with S_C happens slowly.

Note that the lumped matrices of the *small* problems are all the same. The same argument follows for the lumped matrices of the *medium* and *large* problems. This is simply because the parameters that we alter in the experiments of Fig. 2 are only those of $s\mathcal{A}^{(0)}$, which happens to be among the last two automata in the chosen ordering of automata. Even though α and β change, the row sums of the blocks P_{ij} in Eq. (3) are the same because $P^{(0)}_{s_{ab}}u = u$ for all $a, b \in \{0, 1, 2\}$.

Since data packets can use VBR slots when a VBR connection is active but in the low intensity state, we expect P_a to depend weaker on p_{vs} than on p_s. To that effect, we consider the problem $(C, V, B) = (4, 4, 15)$. We set $(p_d(0), p_d(1), p_d(2), p_d(3)) = (0.05, 0.1, 0.25, 0.6)$, $\lambda = 0.5$, and choose $S_C \in \{1, 10\}$. For the VBR arrival processes, we set $(\alpha_{\text{v}}, \beta_{\text{v}})$ so as to have $\lambda_{\text{v}} = 0.5$ and $S_{C_{\text{v}}} = 10$, and we set $(p_{\text{empty}}, p_{\text{busy}}) = (0.9, 0.1)$. Furthermore, we set $(p_{\text{n}}, p_{\text{h}}) = (p_{\text{vn}}, p_{\text{vh}}) = C(5 \times 10^{-6}, 10^{-5})$ and $M = 4$. In Fig. 3, we plot P_a vs. $p_{\text{vs}} = iV \times 10^{-6}$, $i \in \{1, 3, 5, 7, 9\}$, for fixed $p_s = 5C \times 10^{-6}$ using a dashed curve and we plot P_a vs. $p_s = iC \times 10^{-6}$, $i \in \{1, 3, 5, 7, 9\}$, for fixed $p_{\text{vs}} = 5V \times 10^{-6}$ using a solid curve on the same graph when (a) $S_C = 1$, (b) $S_C = 10$.

In the last set of experiments, we again consider the problem $(C, V, B) = (4, 4, 15)$. We set $(p_d(0), p_d(1), p_d(2), p_d(3)) = (0.05, 0.1, 0.25, 0.6)$, $\lambda = 0.5$, and choose $S_C \in \{1, 10\}$. We set $(p_{\text{n}}, p_{\text{h}}, p_s) = (p_{\text{vn}}, p_{\text{vh}}, p_{\text{vs}}) = C(10^{-6}, 5 \times 10^{-5}, 10^{-6})$, $M = 4$, and $S_{C_{\text{v}}} = 10$. In Fig. 4, we plot P_a vs. $\lambda_{\text{v}} \in \{0.1, 0.3, \ldots, 0.9\}$ when (a) $(p_{\text{empty}}, p_{\text{busy}}) = (0.9, 0.1)$ and (b) $(p_{\text{empty}}, p_{\text{busy}}) = (0.5, 0.5)$. We observe

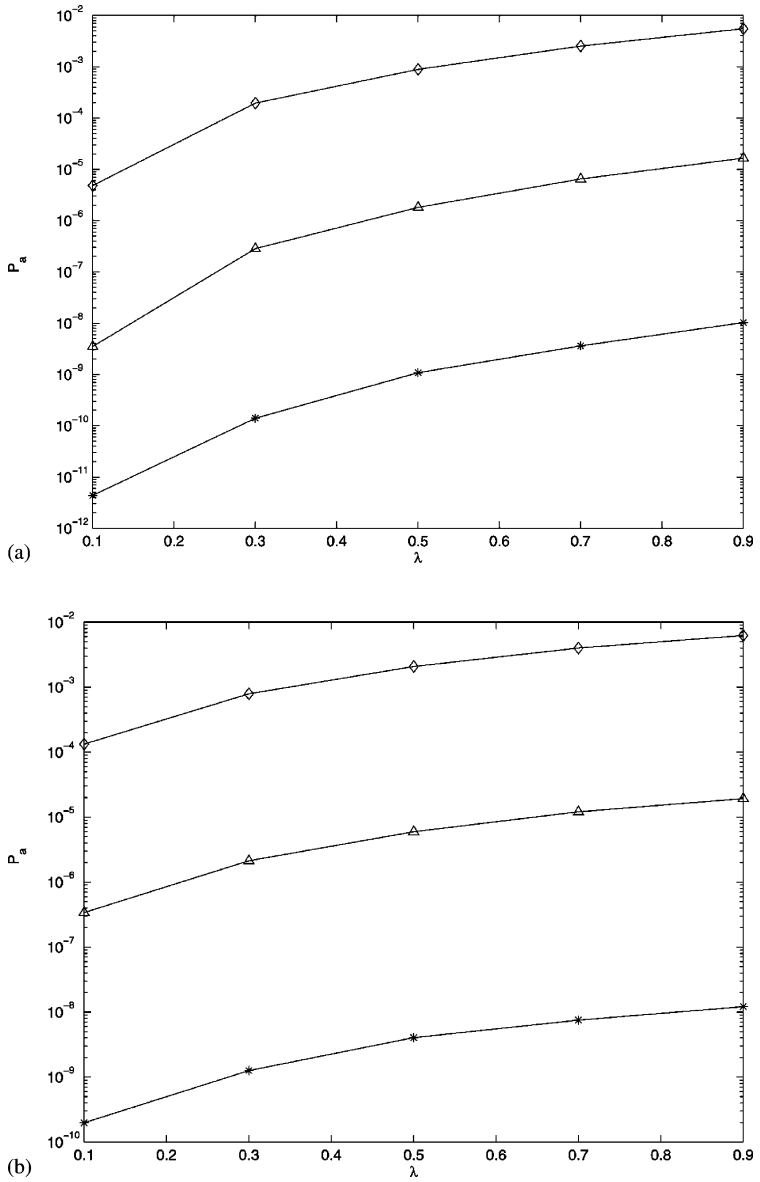

Fig. 2. P_a vs. λ for (◇), (△), and (∗) as small, medium, and large, respectively, when (a) $S_C = 1$, (b) $S_C = 10$.

that when the VBR arrival process behaves more like the CBR arrival process as in part (b), P_a is larger. Furthermore, the increase in P_a with respect to λ_v is smoother for larger S_C.

The underlying DTMCs of the SAN models in Figs. 1, 3 and 4 are irreducible. Those of Fig. 2 are reducible with a single subset of recurrent states (see the end of Section 3). When a reducible discrete-time SAN has a single subset of recurrent states and each subset of the partition in Eq. (3) includes at least one recurrent state (which is the case in the problems of Fig. 2), the lumped matrix computed in step

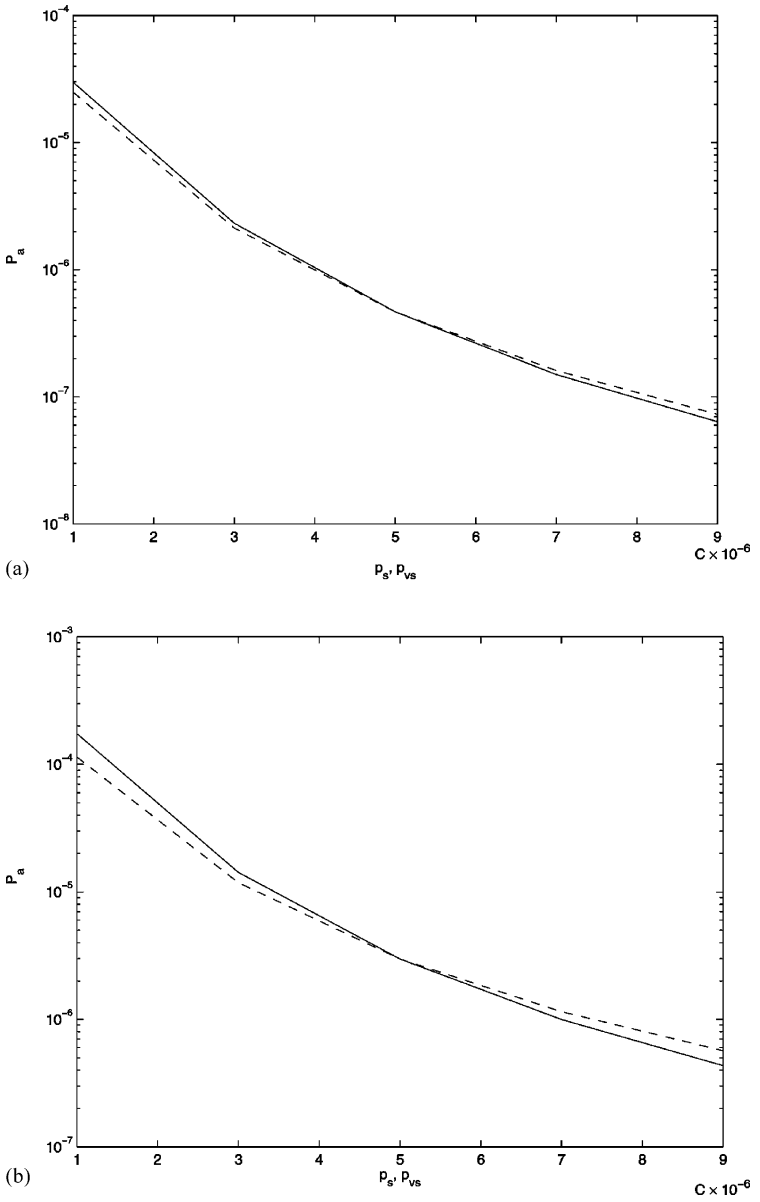

Fig. 3. P_a vs. p_{vs} ($p_s = 5C \times 10^{-6}$, dashed), P_a vs. p_s ($p_{vs} = 5V \times 10^{-6}$, solid) for $(C, V, B) = (4, 4, 15)$ and $\lambda = 0.5$ when (a) $S_C = 1$, (b) $S_C = 10$.

2 of Algorithm 1 is irreducible. With such a partitioning, if one starts in step 1 with an initial approximation having zero elements corresponding to transient states, successive approximations computed by Algorithm 1 will have zero elements corresponding to transient states as well. This simply follows from the fact that in step 3(a) the nonzero structure of z is the same as that of the previous approximation. Consequently, in step 3(b) each computed b_i has zero elements corresponding to transient states. Hence,

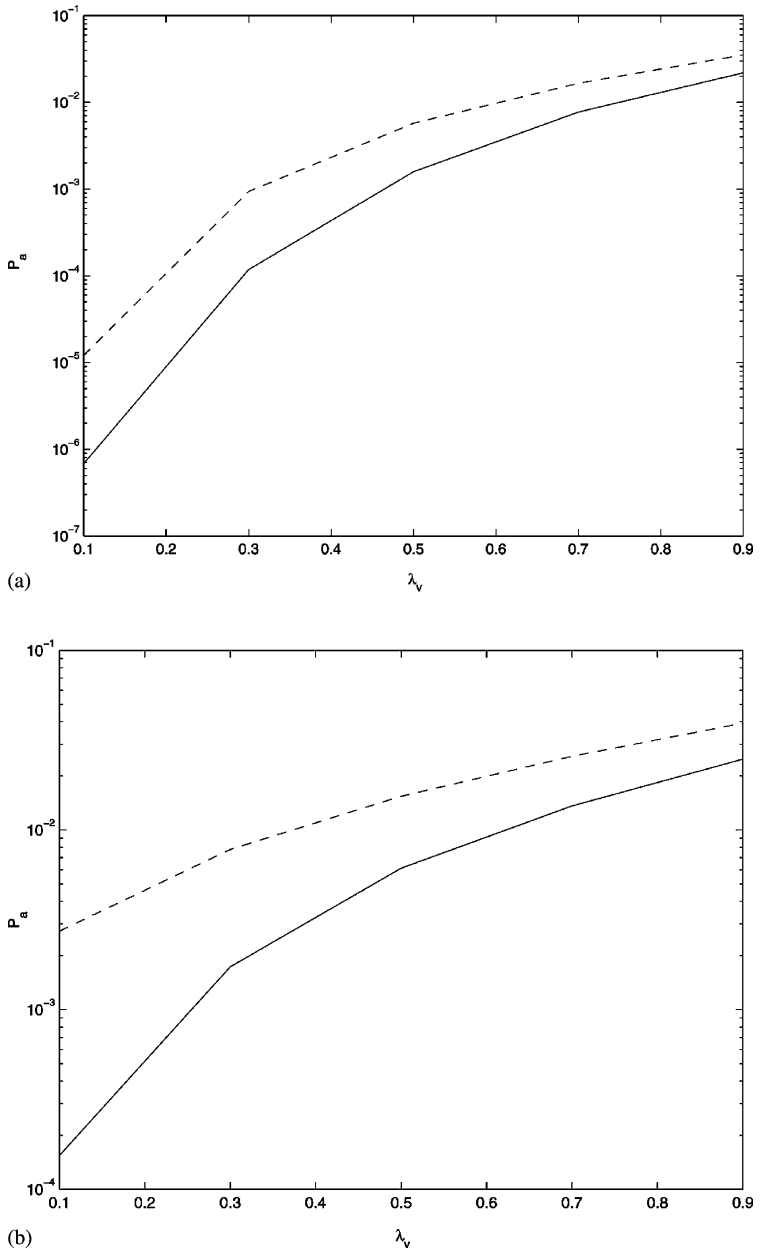

Fig. 4. P_a vs. λ_v when $\lambda = 0.5$, $S_C = 1$ (solid), $S_C = 10$ (dashed) for $(C, V, B) = (4, 4, 15)$ and $S_{C_v} = 10$ when (a) $(p_{\text{empty}}, p_{\text{busy}}) = (0.9, 0.1)$, (b) $(p_{\text{empty}}, p_{\text{busy}}) = (0.5, 0.5)$.

the solutions of the K nonsingular systems at step 3(b) have zero elements corresponding to transient states.

In the problems of Fig. 2, we start with a positive initial approximation just like the other problems and observe that all elements that correspond to transient states become zero at the second iteration in

Table 1
Timing results in seconds and number of AID iterations for Fig. 2

Problem	$\lambda = 0.1$		$\lambda = 0.3$		$\lambda = 0.5$		$\lambda = 0.7$		$\lambda = 0.9$	
	Time	$it/\eta^{(it)}$	Time	$it/\eta^{(it)}$	Time	$it/\eta^{(it)}$	Time	$it/\eta^{(it)}$	Time	$it/\eta^{(it)}$
Small, $S_C = 1$										
AID	8	18	9	21	7	17	8	20	11	27
BGS	15	5.8	17	4.1	14	3.3	16	3.6	22	5.0
Small, $S_C = 10$										
AID	8	18	8	19	8	19	14	35	35	89
BGS	15	6.0	15	4.3	15	3.7	28	4.0	70	4.6
Medium, $S_C = 1$										
AID	69	9	76	10	76	10	83	11	133	18
BGS	68	6.2	75	4.1	74	3.5	82	3.1	132	3.5
Medium, $S_C = 10$										
AID	140	19	162	22	90	12	297	41	789	110
BGS	139	5.1	160	3.8	89	3.3	297	2.9	790	3.1
Large, $S_C = 1$										
AID	774	4	939	5	778	4	1605	9	3257	19
AID*	332	4	386	5	330	4	602	9	1146	19
BGS	236	75.5	290	35.8	236	57.7	508	5.5	1051	2.6
Large, $S_C = 10$										
AID	3420	20	3904	23	941	5	7558	45	20,303	122
AID*	1204	20	1363	23	384	5	2559	45	6745	122
BGS	1103	3.8	1264	2.9	290	33.4	2459	2.1	6630	2.1

the *small* and *medium* problems and at the third iteration in the *large* problem. Once the elements that correspond to transient states in an approximate solution become zero, they remain zero by the argument in the previous paragraph. This is also confirmed experimentally. Hence, there is no need to run the time consuming SC algorithm for each of the 30 experiments in Fig. 2, since the matrices in each of the three problems have the same nonzero structure.

Regarding the solution times of numerical experiments, we provide a representative group of results in Table 1 which are for the problems of Fig. 2. We remark that among *small*, *medium*, and *large*, the DTMC of only the first can be stored on the target architecture (see Section 3). The degree of coupling, $\|P - \mathrm{diag}(P_{11}, P_{22}, \ldots, P_{KK})\|_\infty$, associated with the partitioning in Eq. (3) for the three problems is, respectively, 0.9909, 0.9991, 0.9999 in four decimal digits of precision. Note that $\mathrm{diag}(P_{11}, P_{22}, \ldots, P_{KK})$ is a block diagonal matrix with the blocks P_{ii} placed along the main diagonal. Thus, none of the lumpable partitionings we consider in Algorithm 1 is NCD. However, the smallest degree of coupling we find for each of the 10 *small* problems using the algorithm in [11] is on the order of 10^{-5}. The same value for each of the 10 *medium* problems is also on the order of 10^{-5}. We are not able to determine the value for the *large* problem due its order and density (Section 3), but it is very likely that again we will have a value on the order of 10^{-5}. This all means that although the lumpable partitionings we consider for the problems in Fig. 2 are not NCD partitionings, there exist highly NCD partitionings for each one, and therefore

they are all very ill-conditioned. Nevertheless, we are fortunate that Algorithm 1 does not require NCD partitionings for convergence [25].

We compare the performance of AID with the recursive implementation of BGS discussed in [37]. Due to the high NCDness of the DTMCs at hand, we observed very slow convergence of BGS. For instance, after executing 1000 iterations of BGS for the *small* problem with $S_C = 1$ and $\lambda = 0.1$, the approximate error, $\epsilon^{(\mathrm{it})}$, was greater than 10^{-5} and the approximate residual, $\eta^{(\mathrm{it})}$, was greater than 10^{-6}. In other words, for the chosen tolerance of 10^{-8} on the approximate error, it is not possible to solve the DTMCs of interest using BGS in a reasonable amount of time. Taking this into account, the motivation of our experiments with BGS is to compare its running time with that of AID for the number of iterations it takes AID to converge to the specified tolerance in a particular experiment.

In all experiments, AID and BGS use the same ordering of automata and the same partitioning of P. Furthermore, the diagonal blocks are generated and factorized using the same routines. The timing results of the experiments reported in Table 1 are in seconds. Note that the timing results for AID include time spent to generate and solve the lumped matrix, and timing results for AID and BGS include time spent to generate and factorize diagonal blocks of the partitioning. In column it/$\eta^{(\mathrm{it})}$, we either give the number of iterations, "it", for AID to converge or the scaled approximate error, $\eta^{(\mathrm{it})} \times 10^6$, for BGS after it performs "it" iterations in the corresponding experiment.

When analyzing the problems of Fig. 2 with AID, we solve the lumped matrix of the *small* problem ($n_L = 144$ and $nz_L = 7644$) using GTH in nearly 0 s. We solve the lumped matrix of the *medium* problem ($n_L = 832$, $nz_L = 188{,}094$) in 0.5 s and 16 iterations using sparse IAD with an NCD partitioning of four blocks (with orders varying between 117 and 351) and a degree of coupling on the order of 10^{-5}. We solve the lumped matrix of the *large* problem ($n_L = 4352$ and $nz_L = 3{,}980{,}512$) in 59.4 s and 22 iterations using sparse IAD with an NCD partitioning of 16 blocks (with orders varying between 17 and 1377) and a degree of coupling on the order of 10^{-4}.

It is interesting to note that when the size of the problem increases, the time AID spends in an iteration increases much faster than that of BGS. Thus, in the experiments with the *small* problem, an iteration of AID takes almost half of the time as that of BGS. In the experiments with the *medium* problem, an iteration of AID takes almost the same amount of time as that of BGS. In the experiments with the *large* problem, an iteration of AID takes almost three times that of BGS. Recall that the order of blocks in the partitionings of the three problems is the same and equal to $2(B + 1) = 32$. Hence, when the size of the problem increases, the number of blocks increases as well. In other words, when the size of the problem is relatively large and the order of blocks is relatively small, at each iteration of AID a large number of multiplications are performed when computing the products $z_j^{(\mathrm{it})} T_j^{(e)}$ and $\pi_j^{(\mathrm{it})} T_j^{(e)}$ (see step 3(b) of Algorithm 1). Therefore, the implementation of the disaggregation phase of AID discussed in Section 3.2 is not suitable for highly unbalanced partitionings in which the blocks are relatively small compared to P.

Since the disaggregation phase of AID is a BGS iteration, for unbalanced partitionings the recursive implementation of BGS for SANs is recommended. The results of experiments with the *large* problem for AID*, which is AID with the BGS iteration implemented as in [37], appear in Table 1. Observe that the time spent by AID* in each iteration is slightly larger than the corresponding time of BGS; AID* spends the extra time in step 3(a) of Algorithm 1 when computing $z^{(\mathrm{it})}$.

Regarding the number of iterations taken by Algorithm 1 to converge, the highest values are encountered when $\lambda \in \{0.7, 0.9\}$ and $S_C = 10$. They are iteration numbers greater than or equal to 35, and are for the cases in which α and β are highly unbalanced. As a result, the corresponding solution times are

Table 2
Memory requirements for AID in the experiments of Table 1

Problem	Aggregation		Disaggregation			
	nz_L	max_size	nz_{LU}	LU_size	$n(E+2)$	total_size
Small	7644	0.16	56,112	0.64	0.39	1.03
Medium	188,094	4.31	285,912	3.27	2.23	5.50
Large	3,980,512	91.11	1,278,000	14.63	11.69	26.32

considerably larger than those of other combinations of λ and S_C. If we exclude these six cases, the *small* problem can be solved within 11 s, the *medium* problem within 162 s, and the *large* problem, using AID*, within 1363 s. On the other hand, the smallest time to obtain a solution for the *small*, *medium*, and *large* problems is, respectively, 8, 69, and 332 s. In general, the solution times are very satisfactory if we keep in mind the number of nonzeros of the underlying DTMC.

The amount of core memory required to solve each of the three problems using AID is given in Table 2. The memory requirements of the aggregation phase mainly consist of space to store the lumped matrix. In column nz_L we give the number of nonzero elements in the lumped matrix for each problem and in column max_size we give the maximum amount of memory in megabytes required in the aggregation phase for each problem. As we already mentioned, the lumped matrix of the *small* problem is stored as a square matrix and solved using GTH. Hence, the maximum amount of memory for the *small* problem in the aggregation phase is roughly n_L^2 double precision memory locations. The lumped matrices of the *medium* and *large* problems are solved using IAD with an NCD partitioning. When doing this, we first find an NCD partitioning of the lumped matrix, permute the matrix to the block diagonal form in which all off-diagonal blocks have elements that are smaller than the user specified decomposability parameter [11] and then apply IAD to the permuted matrix. When permuting the lumped matrix, a temporary copy of the matrix is created. Hence, the maximum amount of memory needed in this case is two times that is needed to store the lumped matrix in sparse format.

The requirements of core memory in the disaggregation phase mainly consist of space to store the LU factors of the diagonal blocks and $(E+2)$ vectors of length n that are used in step 3(b) of Algorithm 1. In columns nz_{LU} and LU_size we provide the number of nonzero elements in the LU factors of the diagonal blocks and the corresponding amount of memory in megabytes required to store these LU factors. Column $n(E+2)$ corresponds to the amount of memory in megabytes required to store the $(E+2)$ vectors of length n. Finally, the values in column total_size is the sum of the corresponding values in columns LU_size and $n(E+2)$. Observe that when the implementation AID* is used, the memory requirements of the disaggregation phase are the same as for BGS and mainly consist of space for the LU factors of the diagonal blocks and two vectors of length n.

After the aggregation phase is completed, the memory allocated for the lumped matrix is freed and can be used in the disaggregation phase. Hence, the amount of core memory required for AID is equal to the maximum of the corresponding values in columns max_size and total_size. Thus, the maximum amount of core memory required to solve the *small* and *medium* problems corresponds to the memory requirements of the disaggregation phase of AID, whereas for the *large* problem it corresponds to the memory requirements of the aggregation phase. Observe that for the chosen partitioning of automata, the

amount of memory to store $(E + 2)$ vectors of length n does not exceed the amount of memory to store LU factors of the diagonal blocks.

5. Conclusion

In this paper, we give a discrete-time SAN model of a system in mobile communications. We remodel the SAN introduced in [38] so that the new model is scalable with respect to the number of events. Furthermore, we extend the model by service for VBR calls. We also address the difficulties associated with the analysis of the DTMC underlying the particular SAN, namely its relatively high density and its high NCDness. Using properties of generalized tensor products, we state sufficient conditions under which discrete-time SANs are ordinarily lumpable with respect to the partitioning induced by the tensor representation. For the particular class of ordinarily lumpable discrete-time SANs, we introduce an efficient AID algorithm that takes advantage of lumpable partitionings. Using the proposed algorithm, we analyze the model for various combinations of its parameters. When the blocks in a lumpable partitioning are relatively small with respect to the size of the DTMC, we suggest using the recursive implementation of BGS [37] in the disaggregation phase of AID. Future work may focus on forecasting better orderings of automata and faster converging partitionings based on NCD analyses of DTMCs underlying SAN models.

Acknowledgements

We thank the anonymous referees for their detailed remarks, which led to an improved manuscript.

References

[1] P. Buchholz, Exact and ordinary lumpability in finite Markov chains, J. Appl. Probab. 31 (1994) 59–75.
[2] P. Buchholz, Hierarchical Markovian models: symmetries and reduction, Perform. Eval. 22 (1995) 93–110.
[3] P. Buchholz, Equivalence relations for stochastic automata networks, in: W.J. Stewart (Ed.), Computations with Markov Chains, Kluwer Academic Publishers, Boston, MA, 1995, pp. 197–215.
[4] P. Buchholz, An aggregation/disaggregation algorithm for stochastic automata networks, Probab. Eng. Inform. Sci. 11 (1997) 229–253.
[5] P. Buchholz, Projection methods for the analysis of stochastic automata networks, in: B. Plateau, W.J. Stewart, M. Silva (Eds.), Numerical Solution of Markov Chains, Prensas Universitarias de Zaragoza, Zaragoza, Spain, 1999, pp. 149–168.
[6] P. Buchholz, Exact performance equivalence: an equivalence relation for stochastic automata, Theoret. Comput. Sci. 215 (1999) 263–287.
[7] P. Buchholz, Multilevel solutions for structured Markov chains, SIAM J. Matrix Anal. Appl. 22 (2000) 342–357.
[8] P. Buchholz, G. Ciardo, S. Donatelli, P. Kemper, Complexity of memory-efficient Kronecker operations with applications to the solution of Markov models, INFORMS J. Comput. 12 (2000) 203–222.
[9] R.H. Chan, W.K. Ching, Circulant preconditioners for stochastic automata networks, Numer. Math. 87 (2000) 35–57.
[10] M. Davio, Kronecker products and shuffle algebra, IEEE Trans. Comput. C-30 (1981) 116–125.
[11] T. Dayar, Permuting Markov chains to nearly completely decomposable form, Technical Report BU-CEIS-9808, Department of Computer Engineering and Information Science, Bilkent University, Ankara, Turkey, August 1998. ftp://ftp.cs.bilkent.edu.tr/pub/tech-reports/1998/BU-CEIS-9808.ps.z.
[12] T. Dayar, O.I. Pentakalos, A.B. Stephens, Analytical modeling of robotic tape libraries using stochastic automata, Technical Report TR-97-189, CESDIS, NASA/GSFC, Greenbelt, MD, January 1997.

[13] T. Dayar, W.J. Stewart, On the effects of using the Grassmann–Taksar–Heyman method in iterative aggregation–disaggregation, SIAM J. Sci. Comput. 17 (1996) 287–303.
[14] T. Dayar, W.J. Stewart, Comparison of partitioning techniques for two-level iterative solvers on large, sparse Markov chains, SIAM J. Sci. Comput. 21 (2000) 1691–1705.
[15] P. Fernandes, B. Plateau, W.J. Stewart, Efficient descriptor–vector multiplications in stochastic automata networks, J. ACM 45 (1998) 381–414.
[16] P. Fernandes, B. Plateau, W.J. Stewart, Optimizing tensor product computations in stochastic automata networks, Oper. Res. 32 (3) (1998) 325–351.
[17] J.-M. Fourneau, Stochastic automata networks: using structural properties to reduce the state space, in: B. Plateau, W.J. Stewart, M. Silva (Eds.), Numerical Solution of Markov Chains, Prensas Universitarias de Zaragoza, Zaragoza, Spain, 1999, pp. 332–334.
[18] J.-M. Fourneau, H. Maisonniaux, N. Pekergin, V. Véque, Performance evaluation of a buffer policy with stochastic automata networks, in: Proceedings of the IFIP Workshop on Modelling and Performance Evaluation of ATM Technology, vol. C-15, La Martinique, IFIP Transactions, North-Holland, Amsterdam, 1993, pp. 433–451.
[19] J.-M. Fourneau, L. Kloul, N. Pekergin, F. Quessette, V. Véque, Modelling buffer admission mechanisms using stochastic automata networks, Rev. Ann. Télécommun. 49 (5–6) (1994) 337–349.
[20] J.-M. Fourneau, F. Quessette, Graphs and stochastic automata networks, in: W.J. Stewart (Ed.), Computations with Markov Chains, Kluwer Academic Publishers, Boston, MA, 1995, pp. 217–235.
[21] O. Gusak, T. Dayar, J.-M. Fourneau, Stochastic automata networks and near complete decomposability, Technical Report BU-CE-0016, Department of Computer Engineering, Bilkent University, Ankara, Turkey, October 2000. http://www.cs.bilkent.edu.tr/tech-reports/2000/BU-CE-0016.ps.z.
[22] O. Gusak, T. Dayar, J.-M. Fourneau, Stochastic automata networks and near complete decomposability, SIAM J. Matrix Anal. Appl. 23 (2001) 581–599.
[23] J. Hillston, Compositional Markovian modelling using a process algebra, in: W.J. Stewart (Ed.), Computations with Markov Chains, Kluwer Academic Publishers, Boston, MA, 1995, pp. 177–196.
[24] J.R. Kemeny, J.L. Snell, Finite Markov Chains, Van Nostrand, New York, 1960.
[25] I. Marek, P. Mayer, Convergence analysis of an iterative aggregation–disaggregation method for computing stationary probability vectors of stochastic matrices, Numer. Linear Algebra Appl. 5 (1998) 253–274.
[26] C.D. Meyer, Stochastic complementation, uncoupling Markov chains, and the theory of nearly reducible systems, SIAM Rev. 31 (1989) 240–272.
[27] B. Plateau, De l'évaluation du parallélisme et de la synchronisation, Thèse d'état, Université Paris Sud Orsay, 1984.
[28] B. Plateau, On the stochastic structure of parallelism and synchronization models for distributed algorithms, in: Proceedings of the ACM SIGMETRICS Conference on Measurement and Modelling of Computer Systems, Austin, TX, 1985, pp. 147–154.
[29] B. Plateau, K. Tripathi, Performance analysis of synchronization for two communicating processes, Perform. Eval. 8 (1988) 305–320.
[30] B. Plateau, K. Atif, Stochastic automata network for modeling parallel systems, IEEE Trans. Software Eng. 17 (1991) 1093–1108.
[31] B. Plateau, J.-M. Fourneau, K.-H. Lee, PEPS: a package for solving complex Markov models of parallel systems, in: R. Puigjaner, D. Ptier (Eds.), Modeling Techniques and Tools for Computer Performance Evaluation, Palma de Majorca, Spain, 1988, pp. 291–305.
[32] B. Plateau, J.-M. Fourneau, A methodology for solving Markov models of parallel systems, J. Parallel Distrib. Comput. 12 (1991) 370–387.
[33] M. Siegle, Structured Markovian performance modeling with automatic symmetry exploitation, in: Short Papers and Tool Descriptions of the 7th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation, Vienna, Austria, 1994, pp. 77–81.
[34] G.W. Stewart, W.J. Stewart, D.F. McAllister, A two-stage iteration for solving nearly completely decomposable Markov chains, in: G.H. Golub, A. Greenbaum, M. Luskin (Eds.), Recent Advances in Iterative Methods, IMA Vol. Math. Appl. 60, Springer, New York, 1994, pp. 201–216.
[35] W.J. Stewart, Introduction to the Numerical Solution of Markov Chains, Princeton University Press, Princeton, NJ, 1994.
[36] W.J. Stewart, K. Atif, B. Plateau, The numerical solution of stochastic automata networks, Eur. J. Oper. Res. 86 (1995) 503–525.

[37] E. Uysal, T. Dayar, Iterative methods based on splittings for stochastic automata networks, Eur. J. Oper. Res. 110 (1998) 166–186.
[38] V. Vèque, J. Ben-Othman, MRAP: a multiservices resource allocation policy for wireless ATM network, Comput. Networks ISDN Syst. 29 (1998) 2187–2200.

Oleg Gusak was born in Sumy, Ukraine, in October 1973. He received his M.S. degree in computer science from Kharkov State Technical University of Radio Electronics, Kharkov, Ukraine, in June 1995, and Ph.D. degree in computer engineering from Bilkent University, Ankara, Turkey, in July 2001. In 1995–1997, he was working as a software engineer of Scientific-Research Institute of Automated Control Systems of Gas Pipelines (Concern UkrGasProm). Since October 2001, he is a Visiting Scholar at the School of Interdisciplinary Computing and Engineering in the University of Missouri–Kansas City. His present research interests include performance modeling and analysis of distributed systems and computer networks, wireless networks, optical networks, numerical solutions of SANs.

Tuğrul Dayar received his B.S. degree in computer engineering from Middle East Technical University, Ankara, Turkey, in 1989, and the M.S. and Ph.D. degrees in computer science from North Carolina State University, Raleigh, NC, in 1991 and 1994, respectively. Since 1995, he has been with the Department of Computer Engineering at Bilkent University, Ankara, Turkey, where he is now an Associate Professor. His research interests are in the areas of performance modeling and analysis, numerical linear algebra for stochastic matrices, scientific computing, and computer networks. He is a member of Upsilon Pi Epsilon, IEEE Computer Society, ACM Special Interest Group on Measurement and Evaluation, SIAM Activity Group on Linear Algebra, and AMS.

Jean-Michel Fourneau is a Professor of Computer Science at the University of Versailles St. Quentin, France. He was formerly with Ecole Nationale des Telecommnications, Paris, and University of Paris XI Orsay as an Assistant Professor. He graduated in statistics and economy from Ecole Nationale de la Statistique et de l'Administation Economique, Paris, and he obtained his Ph.D. and his habilitation in computer science at Paris XI Orsay in 1987 and 1991, respectively. He is a member of IFIP WG7.3. He is the head of the performance evaluation team in PRiSM laboratory at Versailles University and his recent research interests are algorithmic performance evaluation, SANs, G-networks, stochastic bounds, and application to high speed networks, all optical networks and architecture.