Load Transformation of Markovian Arrival Processes: Methods and Tool Support

Stephan Heckmüller, Michael Spork, Bernd E. Wolfinger

Dept. of Computer Science, TKRN, University of Hamburg
Vogt-Kölln-Str. 30, D-22527 Hamburg, Germany
E-mail: {heckmueller|2spork|wolfinger}@informatik.uni-hamburg.de

ABSTRACT

Loads in computer networks, i.e. sequences of requests, are modified within the protocol hierarchy of multiple nodes during the transmission process via an interconnected network such as the Internet. This alteration can be seen as a transformation which tends to be of high complexity in today's computer networks. A comprehensive knowledge of the affecting load transformation(s) can be of great help for the prediction of loads as seen at interfaces within the communication network based on information concerning load at the application layer (*primary load*). In the past we successfully developed realistic, analytical models for important types of load transformations. In particular we proposed load transformations for primary loads which can be validly described by BMAPs (*Batch Markovian Arrival Processes*). For a deeper understanding and simplified usage of these analytical transformation models the availability of a tool which allows for a systematic application of the models is highly advantageous. Accordingly this contribution presents a tool which admits the generation of process descriptions of loads which are transformed, possibly multiple times, in a simple and intuitive manner.

1. INTRODUCTION

The number of users and the diversity of distributed applications and computer network services in current networks typically is very large and has been increasing strongly and steadily over the past decade(s). This holds in particular for the Internet and for public mobile networks. As a consequence, the traffic induced into networks tends to be highly complex and very difficult to understand and analyze. However, sufficiently realistic models for traffic and load are indispensable in order to carry out meaningful and valid performance evaluations and predictions for computer and communication networks.

The global goal of this paper is to support the provisioning of realistic characterizations of load as seen at different interfaces within the network (e.g. at different service interfaces within a network's protocol hierarchy), where we define load as a sequence of requests handed over to a service provider. We assume that requests are characterized by the time instant at which they are handed over to the service provider and by a set of request attributes. Typically, load at an application-oriented interface can be observed more easily and be characterized by simpler models.

Evidently, load, i.e. the sequence of requests at a given interface, is changed on its way through the layers of the protocol stack – we call this manipulation and modification of requests *load transformation*. Thus, a load transformation is a mapping of the original sequence of requests SR_1 – which we call *primary load* (PL) – handed over at an 'upper' interface IF_1 onto a new sequence of requests SR_2 – which we call *secondary load* (SL) – observable at a 'lower layer' interface IF_2. Examples of load transforming mechanisms in computer networks may comprise, e.g., fragmentation, the delaying of packets at a router or traffic shaping mechanisms. Coarsely, load transformations can be distinguished into transformations which transform the requests per se and/or those ones which have an impact on the timing of the sequence of requests, e.g. if they change the request interarrival times (for more details, cf. Section 2).

An understanding of load transformations is highly important as it would allow one to come up with realistic load models for lower layer network interfaces just based on knowledge regarding load at a higher interface (without having to measure load at the lower interfaces – where observation could be hard or even impossible anyway). The obtained models can in turn be used for performance evaluation or for load prediction. Moreover, the proposed procedures also apply for cases where the system to be evaluated does not yet exist. That is we can use load transformation procedures during the design phase of a system as well.

So, the specific goal of this paper will be to elaborate techniques and algorithms – combined with dedicated tool support – which allow us to precisely take into account the effect of important classes of load transformations. Quite a few research activities have been carried out in the past for modeling the effect of load transformations [1, 22], which have focused on the transformation of requests per se (in particular on the transformation of request attributes, such as lengths of data units to be transmitted). Therefore, the challenging problem of finding realistic models for the transformation of request arrival processes is still largely unsolved. In order to describe the effect of a given load transformation it is important to make an adequate choice concerning the description technique of the arrival sequence at primary load level. We

SMCTools 2008, October 20, 2008, Athens, GREECE

Digital Object Identifier: 10.4108/ICST.VALUETOOLS2008.4417
http://dx.doi.org/10.4108/ICST.VALUETOOLS2008.4417

focus on primary load models described in terms of BMAPs (*Batch Markovian Arrival Processes*), because BMAPs have already been successfully applied to model a large range of load scenarios.
As one of its main contributions this paper will present a tool which allows to specify in an intuitive way the proposed load transformation procedures on $BMAPs$, being used as primary load models. By means of that tool it is possible to systematically generate process descriptions of the transformed loads which is highly advantageous for a deeper understanding of the underlying systems' behaviour. Moreover the generated arrival processes can be used for the analytical and simulative performance evaluation of the systems under consideration. As the transformation processes encountered in computer networks tend to be highly complex an important requirement to the tool was, allowing to specify complex sequences of transfomations by a graphical editing process. Moreover because of the wide variety of transformation types the tool is expandable via a plugin mechanism. Due to this high degree of flexibility other functions, e.g visualization or state reduction functionality, can be incorporated as well. Up to our knowledge no tool for transformation and manipulation of Markovian Arrival Processes with the mentioned properties exists yet.
The concept of load transformation has been successfully applied to solve a number of problems: In [22] the distribution of packet lengths for fragmented loads was derived. The authors applied these results to $MPEG$ video streams. In [6] analytical transformations based on $BMAPs$ were given for fragmentation and sliding window procedures. Additionally, we proposed analytical transformations based on $BMAPs$ with which we are able to incorporate the influence of loss processes occuring at wireless links [7]. Besides their direct use for load modeling these results can be used to compute an optimal maximum fragment length with respect to throughput maximization.
Moreover, the concept of load transformation is closely related to that of departure processes, where the transformed load is described by means of process specifications. In [24] the authors give approximative descriptions of the departure process from a $BMAP/MAP/1$-Queue. The authors assume infinite capacity of the queue and give approximations, whose complexity depends on the lag up to which the correlation of the queue length is captured.
Batch Markovian Arrival Processes (BMAP) were first introduced with alternative notation in [17]. The author uses the term *versatile Markovian Point Process*. BMAPs in the formulation used in this paper were introduced in [15]. Since then considerable effort has been made to investigate *Batch Markovian Arrival Processes*. Recent works include for example the blocking probability of a $BMAP/G/1$ Queueing System [2], the departure process from a $BMAP/G/1$ Queue [5] or the workload distribution for systems with threshold [14]. Here the server is assumed to stay idle until workload, i.e. the number of requests waiting for service, exceeds a certain threshold. Other works extend the $BMAP/G/1$ Queueing System: In [11] such an extension of $BMAP/G/1$ is elaborated in order to allow the arrival stream to depend on the system's state, i.e. on the number of customers in the system.
In [13] BMAPs are used to model network traffic at IP-level. The authors give procedures to match model parameters with measured data. Several modeling techniques for video traffic using $BMAP$s were proposed. In [10] video traffic is successfully modeled by discrete $BMAP$s in order to investigate the queueing behaviour of the video stream. The authors use the results for buffer dimensioning. Moreover, adequate models for voice data transmission can based on constructed from the on-off models typically used in literature (cf. e.g. [20]). There are nonetheless cases where modeling of primary load by means of BMAPs might not be the best choice, e.g. when the load generation process at primary load level is influenced by the state of the computer network.
Though we do not know of any tools with directly comparable functionality, several remotely related tools have been published, e.g. for model checking of markov chains [9] or for the analysis and simulation of Petri nets [18].
The rest of this paper is organized as follows: In section 2 we describe the concept of load transformation in general and its application to Markovian Arrival Processes. We propose a load transformation modeling the effect a token bucket regulator has on Markovian Arrival Processes. In section 3 we present our newly designed tool, which allows for graphical composition of load transformations of Markovian Arrival Processes. By using this tool we evaluate the usefulness and accuracy of the proposed methods in section 4.

2. LOAD TRANSFORMATIONS

Modern computer networks, which are organized in layers and interconnected systems, constitute complex networks of service stations. In such a network each station influences the load characteristics seen by subsequent stations. This alteration of load characteristics is called, in our terminology, transformation of primary load to secondary load (cf. figure 1 where process P could correspond to ≥ 1 adjacent protocol layers in its functionality). In the context of computer networks load transformations are for example packet fragmentation, queueing delay or traffic shaping.
Before introducing load transformation in a formally rigouros manner, we first need to define load as shown in definition 1 [21].

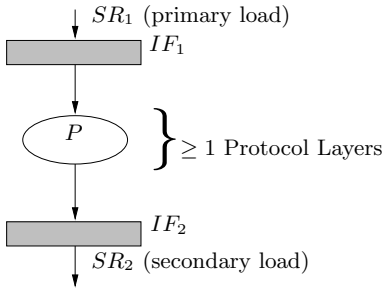


Figure 1: Load transformation between interfaces IF_1 and IF_2 by a request transforming process P

DEFINITION 1. *Load $L = L(E, S, IF, T)$ is defined as the sequence of requests, which are provided to the service system S during interval T by its environment E. Requests are passed via interface IF, which separates the service system from its environment.* ◇

Based on Definition 1 we define four types of transformations

Digital Object Identifier: 10.4108/ICST.VALUETOOLS2008.4417
http://dx.doi.org/10.4108/ICST.VALUETOOLS2008.4417

which we subsequently use to describe the effect which a given transformation has onto a given load.

1. We define request transformation as a function T_R, which maps the sequence of primary load requests $R^p = (r_1^p, ..., r_N^p)$ to the sequence of secondary load requests $R^s = (r_1^s, ..., r_K^s)$ for a given transformation.

$$T_R : R^p \rightarrow R^s$$

2. Transformation of timing is defined as a function mapping interarrival times of primary load $T^p = (t_1^p, ..., t_N^p)$ to interarrival times of secondary load $T^s = (t_1^s, ..., t_K^s)$

$$T_T : T^p \rightarrow T^s$$

3. If the transformation of request attributes cannot be achieved independently of timing, we additionally use

$$T_R' : R^p \times T^p \rightarrow R^s$$

4. Likewise, if the transformation of timing cannot be achieved independently of request attributes, we use

$$T_T' : R^p \times T^p \rightarrow T^s$$

Case 1 corresponds, e.g., to the influence of fragmentation on the distribution of packet lengths, whereas case 2 could be used to capture the influence of variable delay experienced in a router. Case 3 may be applied when the interarrival times of packets influence the loss probability of packets, so that the packet lengths at secondary load level depend on the packet lengths and the interarrival times at primary load level. Analogously, case 4 can be used to model the influence of packet lengths on the interarrival times at secondary load level.

Although it is possible to transform concrete arrival sequences in a straightforward algorithmic (e.g. simulative) manner, this approach bears some disadvantages. First the secondary load obtained via transformation of concrete arrival sequences is not suitable for analytic methods like queueing theory, which makes it difficult to obtain performance measures, e.g. waiting time or blocking probability. Second, even if we use a stochastic model for primary load, the obtained description of secondary load would only be a realization of the transformed stochastic process.

On the other hand analytically tractable models often have strong restrictions concerning their modeling power. In order to circumvent the disadvantages of both worlds as far as possible, we chose to model primary load by means of *Batch Markovian Arrival Processes* (BMAP) (see definition 2 [16]).

Definition 2. *Let the* Batch Markovian Arrival Process *(BMAP) be defined as a markovian arrival process with infinitesimal generator matrix:*

$$\mathbf{Q} = \begin{pmatrix} D_0 & D_1 & D_2 & D_3 & \dots \\ 0 & D_0 & D_1 & D_2 & \dots \\ 0 & 0 & D_0 & D_1 & \dots \\ \vdots & & \ddots & & \ddots \end{pmatrix}$$

With $D_{0,...,\infty}$ being $(m \times m)$ matrices defined by:

$$(D_0)_{ii} = -\lambda_i, \quad 1 \leq i \leq m$$

$$(D_0)_{ij} = \lambda_i p_i(0,j), \quad 1 \leq i,j \leq m \wedge i \neq j$$

$$(D_k)_{ij} = \lambda_i p_i(k,j), \quad 1 \leq i,j \leq m \wedge k > 0$$

$$\sum_{\substack{j=1 \\ j \neq i}}^{m} p_i(0,j) + \sum_{k=1}^{\infty}\sum_{j=1}^{m} p_i(k,j) = 1, \quad 1 \leq i \leq m$$

◇

Here $p_i(k,j)$ denotes the probability that – given the process is in state i – a transition to state j occurs with a batch arrival of size k. The overall rate with which the process leaves state i is given by λ_i.

The class of $BMAP$ contains a number of stochastic arrival processes which are typically used in load modeling. One of this process classes is the Markov Modulated Poisson Processes ($MMPP$) which is given by an infinitesimal generator matrix Q and a vector of rates r. Here arrivals occur at a rate specified by the i^{th} entry of r if the process is in state i. For further usage we define the conversion of a $MMPP$ to a $BMAP$ as given by Equ. (1) [16]. Here $diag(r)$ represents a diagonal matrix with values given by the vector of rates r.

$$D_{i\geq 0} = \begin{cases} Q - diag(r), & i = 0 \\ diag(r), & i = 1 \\ 0, & i > 1 \end{cases} \quad (1)$$

In prior works we gave transformation procedures operating on BMAPs for fragmentation processes [6] and for losses on wireless links [7]. These procedures allow to derive the secondary load description, resulting from the particular transformation, as a $BMAP$ again. Thus, we model load transformations as functions mapping BMAPs to BMAPs:

$$T_{BMAP} : \mathcal{BMAP}^p \rightarrow \mathcal{BMAP}^s \quad (2)$$

In order to describe load aggregation by means of the methods used here we additionally define

$$T_{BMAP} : \mathcal{BMAP}^p \times \mathcal{BMAP}^p \rightarrow \mathcal{BMAP}^s \quad (3)$$

where the aggregation of two BMAPs D^1 and D^2 is given by Equ. (4) where $\oplus$ denotes the Kronecker sum [16].

$$D_i = D_i^1 \oplus D_i^2, \quad i \in \{0, ..., \infty\} \quad (4)$$

We developed transformation procedures based on $BMAPs$ for a number of scenarios. As mentioned above we successfully applied the proposed method to fragmentation, sliding window procedures and the impact of losses induced by wireless links. Moreover we developed a transformation procedure modeling the effect a token bucket module has on arrival sequences which we will describe in this article. Evidently the description of departures from a queue by means of markovian arrival processes can be seen as a result of a load transformation, too.

Putting these results together we are able to describe more complex scenarios as a composition of multiple transformations (possibly of different type). As we only consider transformations mapping to a single $BMAP$ the evaluation of the composed transformations resembles a tree structure with the final result as the root. Moreover as aggregation is considered, too, multiple partially independent paths may occur. We will later on make use of these two facts in the design of the transformer tool.

Digital Object Identifier: 10.4108/ICST.VALUETOOLS2008.4417
http://dx.doi.org/10.4108/ICST.VALUETOOLS2008.4417

Due to the restricted space we cannot describe all transformation procedures developed yet. For this reason we illustrate the concept by presenting a load transformation modeling the impact of a token bucket mechanism. The token bucket mechanism is widely used for traffic shaping and policing in computer networks especially in Integrated Service (IntServ) resp. Differentiated Service (DiffServ) environments. In a token bucket module for each served request (e.g. packet) one token has to be present in the system. Tokens arrive periodically and can be accumulated up to maximum value n_t (cf. [19]). The maximum number of tokens queueable in the system and the rate with which they arrive determine the influence the module has on arrival sequences. Moreover a token bucket module can be used as a dropping device or as traffic shaper. In the former case arriving requests are dropped with a given probability if no token is present at the specific arrival instant. In the latter case queueing of requests is allowed up to maximum queue length n_a which leads to a shaping of the arrival sequence.
In a token bucket module tokens are assumed to arrive with constant interarrival time. Here for the purpose of tractability we assume that tokens arrive according to the Markovian Arrival Process $MAP^T = (D_0^T, D_1^T)$. We discuss the approximative character of the description given here in section 4. Furthermore the arrival of requests is assumed to be given by $MAP^P = (D_0^P, D_1^P)$.
The key idea behind the transformation procedure is that the behaviour of the system (with respect to departures) is determined by three factors: the state of the arrival process of requests, the state of the arrival process of tokens and the number of tokens resp. requests present in the token bucket module. For that reason we define the state of the token bucket module as $k \in \{-n_a, ..., n_t\}$ where states $k < 0$ represent backlogged requests and states with $k > 0$ represent accumulated tokens. Both state spaces can now be combined by means of Kronecker operations in such a way that the behaviour of the system when the token bucket module is in state k can completely be described. In order to do so we define the following block matrices based on the description of the arrival process of requests MAP^P resp. tokens MAP^T:

$$L = D_0^P \oplus D_0^T \tag{5}$$

$$\hat{L} = I(m_t) \otimes D_0^P \tag{6}$$

$$T = D_1^T \otimes I(m_a) \tag{7}$$

$$A = I(m_t) \otimes D_1^P \tag{8}$$

Here m_a resp. m_t denotes the size of the state space of arrival process of requests resp. tokens and $I(n)$ is an identity matrix of size $n \times n$. Moreover, $\otimes$ denotes the Kronecker product. The matrices defined above bear resemblance to those matrices used in the description of departure processes from queues (see e.g. [24]). This results from similarities that exist between queues and the token bucket mechanism: Depending on the state of the token bucket module either tokens or requests are queued so that the system can be seen as a double sided queue.
The semantic of the four matrices is as follows: Matrix L represents transitions of the system which do not change the number of tokens resp. requests present in the system, i.e. only transitions without arrivals are considered here. Matrix $\hat{L}$ only considers transitions in the arrival process of requests. This matrix is used to describe the system's behaviour when the maximum number of tokens is accumulated. Furthermore, matrix T describes the rate with which the arrival of tokens changes the system's state from state k to $k+1$. Analogously matrix A describes the rate with which arriving requests change the system's state from k to $k-1$ (in case the number of backlogged requests has not reached its maximum value yet). Based on these semantics the block matrices can be combined as shown in Proposition 1 in order to describe the system's behaviour as a whole.

PROPOSITION 1. *Let the Markovian Arrival Process describing the arrival process at primary load level be given by $MAP^P = (D_0^P, D_1^P)$. Tokens arrive according to the Markovian Arrival Process $MAP^T = (D_0^T, D_1^T)$.*
Then the departure process from the Token Bucket regulator is given by $MAP^S = (D_0^S, D_1^S)$ of size $m = m_a \cdot m_t \cdot (n_a + n_t + 1)$ with Matrices

$$D_0^S = \begin{pmatrix} L+A & 0 & 0 & \dots & & & 0 \\ A & L & 0 & \dots & & & 0 \\ \vdots & & \ddots & & & & \vdots \\ 0 & \cdots & A & L & T & 0 & \cdots \\ 0 & \cdots & 0 & 0 & L & T & \cdots \\ \vdots & & \ddots & & & & \vdots \\ 0 & \dots & & & & 0 & \hat{L} \end{pmatrix}$$

$$D_1^S = \begin{pmatrix} 0 & T & 0 & \dots & & & 0 \\ 0 & 0 & T & \dots & & & 0 \\ \vdots & & \ddots & & & & \vdots \\ 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ 0 & \cdots & & A & 0 & \cdots & 0 \\ \vdots & & \ddots & & & & \vdots \\ 0 & \dots & & & 0 & A & 0 \end{pmatrix}$$

◇

Here n_t denotes the maximum number of tokens that can be accumulated and n_a is the maximum queue length. The state space can be separated into $(n_a + n_t + 1)$ subsets corresponding to the number of tokens resp. requests present in state k. We enumerate these subsets by $Z = \{-n_a, ..., n_t\}$. Here a negative number $-l$ refers to subsets representing states with l backlogged requests, whereas a positive number k represents subsets with k tokens present in the system. In states $-l$ the system behaves essentially like a queue where token arrivals correspond to service completion. For states k the roles change and in a way the tokens behave like queued requests while arrivals have the effect of service completion on the queued tokens. To summarize, the process under consideration can be seen as "double sided queue". We describe the properties of the departure process from a token bucket module in a detailed manner. As the number of the states might exceed a maximum level tolerable for subsequent analysis we additionally proposed a method for the reduction of the state space size with which state space subsets can be aggregated (cf. [8] for details). Additionally standard state space reduction techniques (e.g. [4]) can be used in cases where the state space grows too large. In these cases the proposed procedures serve as a tool to derive a detailed description of the transformed load whose complexity can be reduced to an acceptable degree.

Digital Object Identifier: 10.4108/ICST.VALUETOOLS2008.4417
http://dx.doi.org/10.4108/ICST.VALUETOOLS2008.4417

3. ARCHITECTURE OF THE LOAD TRANSFORMER

We believe that tool support is essential for the usability of modeling methods in general. This holds especially for the proposed load transformations of Markovian Arrival Processes where the transformation procedures typically consist of matrix based operations which are not always straightforward to implement. To simplify the application of the proposed methods we designed a tool which allows the composition of transformations and the aggregation of load models by means of a graphical editing process. In this section we present the architecture of this load transformer. For this architecture we identified the following requirements:

R1 Generality The architecture should support multiple input resp. output interfaces and load transformations as presented in this paper. Additionally state space reduction and visualization functionality is desirable.

R2 Expandability Based on the previous requirement it should be possible to extend the functionality without the necessity of modifying the load transformer itself.

R3 Portability It should be possible to run the transformer on all operating systems commonly used.

R4 Support for Concurrent Execution As the transformation of multiple arrival processes which are aggregated at some point of operation allows for the concurrent execution of the transformations the architecture should facilitate parallelism.

R5 Support for Hierarchical Modeling In modeling a transmission process several typical sequences of transformations are frequently met. In order to increase to clarity of the models it should be possible to represent these sequences as one single module.

R6 Usability It should be possible to generate models of transformed loads without detailed knowledge of the underlying algorithms or of a proprietary script language. Once a model is specified the generation of the transformed models should be fully automated.

In the following we describe the mechanisms which were employed in order to fulfil the requirements *R1-R6*. Concerning the first two requirements we opted for a plugin based mechanism, which means that additional modules can be added to the transformer without the source code available. Instead of modifying the source code of the transformer a *C++* interface is provided that has to be implemented. The resulting binary code is integrated by runtime linker mechanisms available in all operating systems commonly used. Based on this decision we implemented the core functionality of the transformer which is illustrated by the class diagram shown in fig. 2. As can be seen the transformation functionality is accessed by the graphical user interface through the transformer module which encapsulates the core functionality. Each module is represented by an instance of the class *TransformAlg* which possesses information about its sucessors, the plugin to be used and the specific parameters. The plugin itself implements the specific transformation functionality and is dynamically linked into the program. In order to get successfully linked and used the plugin module has to provide methods for the transformation and the

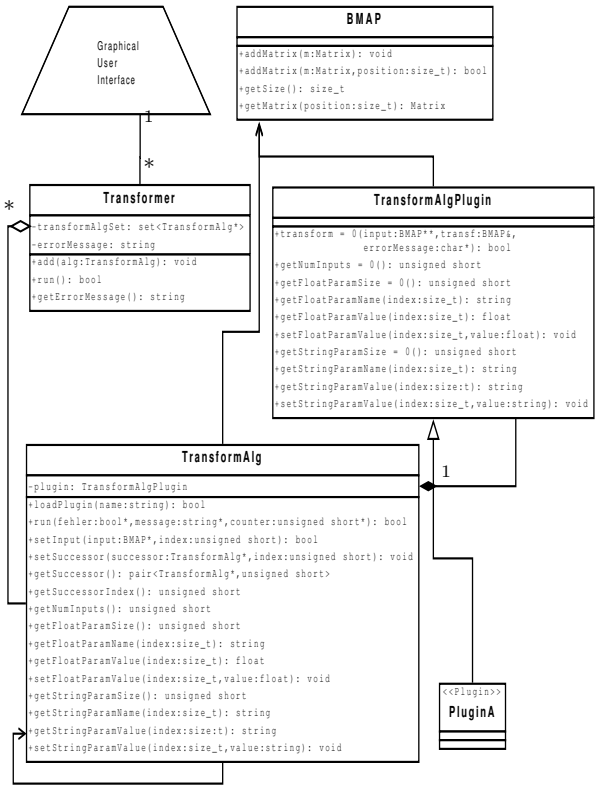


Figure 2: UML-Class Diagram of the transformer

passing of parameters. (These methods are called *pure virtual* in *C++* terminology and are followed by $'' = 0''$ in the class diagram shown in fig. 2). The number of parameters and their type is determined by the plugin itself and depends on the underlying transformation. In case of the token bucket transformation described above we have two parameters: the maximum number of queueable requests n_a and the maximum number of queueable tokens n_t. In case of the fragmentation transformation the two parameters used are the maximum fragment length and the mean interarrival time between fragments. The tool provides adaptive dialogs for the parametrization of each plugin module so that we can adjust the behaviour of the transformation algorithm to that of the network module to be modeled.

The third requirement *Portability* is reached by only using libraries which are available on all major operating systems. For that reason we are using the *wxWidgets*-Library [23] for the graphical user interface and runtime linkage functionality which enables us to compile the resulting *C++*-Code without additional operating system specific adaption of the source code.

In order to allow for Concurrent Execution of the transformations we chose a completely data flow-driven execution by which the degree of parallelism inherent in the model is exploited. This design is very similar to the concept of data flow architecture proposed in the context of programming language and computer design (see e.g. [12]).

The mode of operation of the chosen architecture is illustrated in fig. 3. Here each module waits for its input chan-

Digital Object Identifier: 10.4108/ICST.VALUETOOLS2008.4417
http://dx.doi.org/10.4108/ICST.VALUETOOLS2008.4417

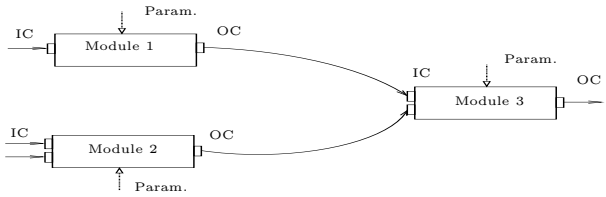


Figure 3: Illustration of the dataflow architecture

nels (IC) to receive input data and starts processing the input data afterwards. After finishing the computation the output data is sent to the succeeding module via the output channel (OC). This way the maximum degree of parallelism possible (at this degree of granularity) is reached without the necessity of additional synchronisation mechanisms.
Although more complex scenarios are possible we restrict ourselves to modules with one output channel and $k \in \{0, 1, 2\}$ input channels. Modules with zero input channels are sources (e.g. reader modules) which can be thought of as starting points of the computation. Moreover, modules with two input channels aggregate two $BMAPs$. By cascading such modules we are able to reach arbitrary degrees of aggregation. The module specific parameters are set a priori, that is during the configuration of the model. The resulting sequence of events during transformation is depicted in fig. 4 where we show the interaction between the transformer module and two TransformAlg modules.
For the fifth requirement (*Hierarchical Modeling*) we implemented a plugin module which is able to run transformations specified by project files previously saved in the transformer. That is we can specify a model in the transformer and use the generated save file directly as single module. For the sake of simplicity we restricted ourselves to models with one input and one output. By using this module we are able to introduce as many layers of hierarchy as desired because as a consequence of the architecture the *MultiPlugin* Module can be used recursively.

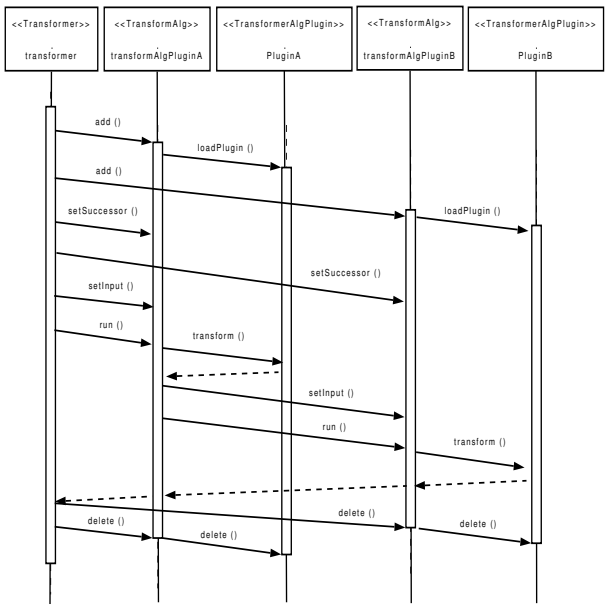


Figure 4: Sequence Diagram

Finally, in order to make the usage of the resulting tool easy we implemented a graphical user interface which allows for model specification as a graph structure (cf. fig. 5). This way the model can be specified without writing program code. Instead the model can be directly "drawn" and after dialog based parametrisation of the model the resulting transformed load can be automatically generated.
We plan to make the described tool publicly avaible soon on our website[1] accompanied by several input resp. output modules, transformations and visualization modules already implemented. The following load transformations can, up to now, be modeled within the transformer tool:

- Fragmentation [6]
- Header Generation [7]
- Token resp. Leaky Bucket Regulation
- Losses on wireless links [7]
- Sliding Window procedures [6]
- Queueing, cf. e.g. [24]
- Aggregation [16]

As mentioned in section 2 the class of $BMAPs$ contains the widely used Poisson Processes and Markov Modulated Poisson Processes for which input modules already exist. That is we can use the transformer directly as a tool for these two process classes. Additionally we implemented a module for the visualization of realizations of the Markovian Arrival Process whose output is illustrated by way of example in figure 5. By means of that module the properties of the transformed arrival process can be investigated by visual inspection.

4. CASE STUDIES

In the following section we illustrate the proposed approach by means of a case study. For that we employ the load transformation procedures in order to model a typical DiffServ scenario. For the generation of the departure process we use the transformer tool described in the previous section. In this scenario a video stream is multiplexed with aggregated flows, which we model by a Markov Modulated Poisson Process. As a model for the video stream we use a simple $BMAP$ with three states. Each state represents one of the frame types (i.e. $I-$,$P-$ and $B-$Frames) typically used in case of *MPEG*-Coding. The probabilities $p_k(i, j)$ (cf. Def. 2) were chosen according to the empirical length distribution derived from a trace of a recorded soccer game. The mean and standard deviation of the state specific batch sizes are shown in table 1. (We do not model varying motion intensities here. This is nonetheless possible at the cost of a higher number of states.)
Before being multiplexed the modeled video stream is subject to fragmentation and header generation. For this we use the two corresponding procedures proposed in [6, 7]. We model fragmentation explicitly for two reasons: First, by using this procedure we are able to incorporate the influence of

[1] http://www.informatik.uni-hamburg.de /TKRN/world/lupus/index.htm

Digital Object Identifier: 10.4108/ICST.VALUETOOLS2008.4417
http://dx.doi.org/10.4108/ICST.VALUETOOLS2008.4417

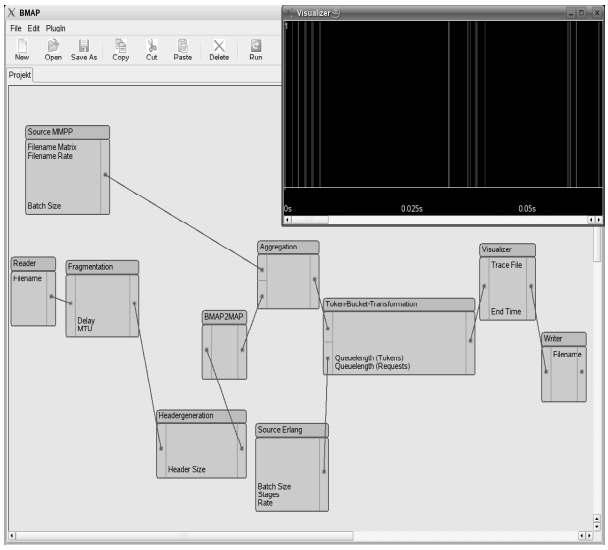


Figure 5: Graphical User Interface of the Transformer

slow transmission facilities which may induce non-negligible delays between single fragments. This phenomenon would have been completely neglected if the model of the unfragmented arrival sequence would have been used. Second, in order to limit the complexity of the model of the departure process from a token bucket module we restrict ourselves to simple Markovian Arrival Processes. For this we have to neglect the batch sizes. As the size of a fragment is typically smaller than the size of the whole packet the error induced by that abstraction is in general smaller if models of fragmented arrival sequences are considered. The model of the fragmented arrival sequence is additionally subject to a transformation which models the generation of fragment headers. This transformation simply modifies the $BMAP$ in such a way that a batch arrival of size $i \in \{1, ..., \infty\}$ in the $BMAP$ representing the untransformed load implies a batch arrival of $i + h$ where h is the specific header length (here assumed to be constant). For the subsequent reduction of the $BMAP$ to a MAP we use the following formulas.

$$D_0^{MAP} = D_0^{BMAP}, \quad D_1^{MAP} = \sum_{i=1}^{M} D_i^{BMAP} \tag{9}$$

Here M represents the maximum fragment length which we set $M = 1000$ in the experiments presented here. The MAP model of the video stream derived by formula (9) is multiplexed with the model of aggregated background traffic for which we use the $MMPP$ illustrated in formula (10). This process describes the aggregated traffic at a packet level so that we derive the MAP of the aggregation of both arrival sequences by means of formula (4).

$$Q_1 = \begin{pmatrix} -5 & 5 \\ 10 & -10 \end{pmatrix} \quad r = \begin{pmatrix} 100 & 500 \end{pmatrix} \tag{10}$$

The resulting model for the aggregated traffic is subsequently policed by a token bucket module. The considered sequence of transformations models a typical DiffServ scenario in which an aggregated flow is policed before entering the subnetwork. The scenario as a whole is illustrated in figure 6. (In terms of analytical transformations the described sequence coincides with the one graphically illustrated in figure 5. The modules were configured according to the parameters described above, such as maximum fragment length.)

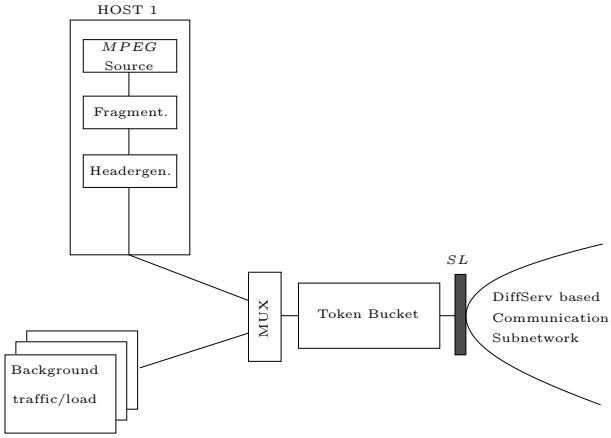


Figure 6: Illustration of the case study

In the following we will investigate the usage of such a module as a shaping device and as a dropping device as well. In the former case arriving packets can be queued up to maximum queue length and are delayed up to the point where the profile a priori agreed on allows the sending of a packet of the specific size. In the latter case packets which do not meet the profile specifications are dropped with a given probability. (In the following experiments we use a dropping probability of one in order to emphasize the effect of the transformation. Lower probabilities are nonetheless possible by increasing the arrival rates accordingly.)

First we present the results of the transformation for the dropping scenario. In order to evaluate the accuracy of the derived traffic model we compare the properties of realizations of the transformed MAP_1 with properties of arrival sequences which were transformed simulatively. For this we simulated the scenario described above in the network simulator $ns2$ and collected measurements at the interface to the network layer of the host following the token bucket module. Furthermore for the analytically transformed load we assume that the interarrival times of tokens are distributed according to a $k-$stage Erlang distribution, which allows us to approximate constant interarrival times arbitrary closely. Unless otherwise stated we use 3 stages in the following.

In the upper part of figure 7 we plotted the empirical distribution of the interarrival times of the analytically resp. simulatively transformed load. For comparison purposes we additionally plotted the corresponding distribution for

Table 1: Mean and standard deviaton of packet distribution generated in state i of the $BMAP$ modeling the video stream

State	$E[L^P]$	$Std[L^P]$
I	8396.7	2211.7
P	6360.6	2299.4
B	4863.2	1880.6

Digital Object Identifier: 10.4108/ICST.VALUETOOLS2008.4417
http://dx.doi.org/10.4108/ICST.VALUETOOLS2008.4417

the case of unpoliced load for which we deactived the token bucket transformation. As can be seen the analytically transformed load captures the characteristics of its simulatively transformed counterpart with good accuracy. Except for small time ranges the curves nearly coincide. Nonetheless as can be seen in the lower part of figure 7 we are clearly able to match the main characteristics in this region, too. Compared to the unpoliced load we see that the probability of small interarrival times is reduced as expected.
Besides interarrival times we have to investigate the rate with which departures from the Token Bucket module occur. For that purpose we compared the rate which is induced by k successive departures in the mean defined by:

$$R(k) = \frac{1}{N-k} \sum_{i=k+1}^{N} \frac{\sum_{j=i-k}^{i-1} l_j}{t_i - t_{i-k}}$$

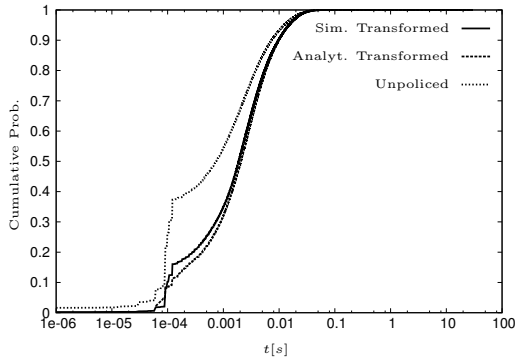
Here t_i denotes the i^{th} arrival time and l_i is the length of the i^{th} packet. The resulting rate for the experiment under consideration is depicted in the middle part of figure 7. The curves representing the mean rate of the simulatively resp. analytically transformed sequences of requests are nearly identical which indicates that the transformations are able to capture the mean rate with very good accuracy. In DiffServ scenarios not only the mean rate is of interest but also the maximum rate. It is evident that we cannot capture the mean rate with the markovian methods used here deterministically. Nonetheless we see that the methods used here are able to capture the charateristics of the maximum rate with good accuracy even for small values of k. With increasing k the curve corresponding to the analytically transformed load converges to the curve corresponding to the simulatively transformed load.

We evaluated the transformation procedures for varying

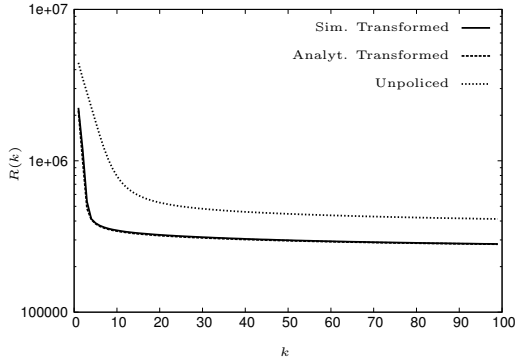
Table 2: Ratio of induced bytes of the analytical model compared to the simulative model of the transformed MAP_1 in percent for different k

$k=1$			
Rate R_t |Bucket Size	1	3	5
1.2	96.5%	103.9%	104.7%
1.6	95.3%	104.3%	103.7%
2.0	96.1%	101.7%	102.5%
$k=3$			
Rate|Bucket Size	1	3	5
1.2	97.5%	100.9%	103.6%
1.6	98.1%	101.2%	103.9%
2.0	97.8%	99.5%	100.6%
$k=5$			
Rate|Bucket Size	1	3	5
1.2	98.5%	102.5%	102.9%
1.6	97.5%	99%	102.3%
2.0	98.2%	101.5%	100.5%

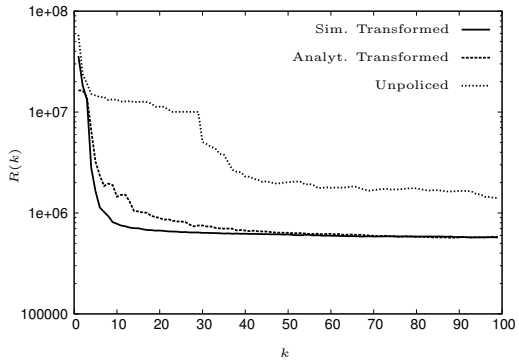
parametrisations of the token bucket module and of the $MMPP$ and for other $MPEG$ traces and achieved comparable degrees of accuracy. This fact is demonstrated by table 2 where we present the ratio of bytes induced by the simulatively transformed load compared to its analytically transformed counterpart for various token bucket parametrisations. Additionally, we illustrate the effect of the number of Erlang stages (k) used. As can be seen the deviations



(a) Interarrival Times ($R_t = 1.6R; n_t = 3$)



(b) Mean Rate ($R_t = 1.6R; n_t = 3$)



(c) Maximum Rate ($R_t = 1.6R; n_t = 3$)

Figure 7: Comparison of the mean and maximum Arrival Rate of the transformed MAP_1 for varying Token Bucket parameters

Digital Object Identifier: 10.4108/ICST.VALUETOOLS2008.4417
http://dx.doi.org/10.4108/ICST.VALUETOOLS2008.4417

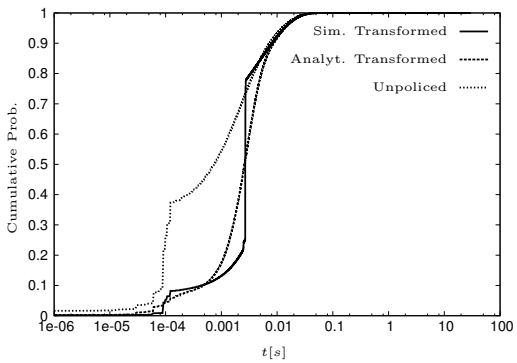
from 100% do not exceed 4% and are typically much lower. It is noteworthy that we achieve this without actually considering the exact packet length in the analytical model but only model load at a packet level.

As mentioned before we can use the token bucket transformation shown in Prop. 1 to model the departure process from a token bucket module used as a shaping device. For that purpose we have to choose $n_a > 0$. In the following, we present a modification of the preceeding experiment where we set $n_a = 3$. (The maximum number of tokens is kept fix at $n_t = 3$ and the rate is reduced to $R_t = R$.) Analogously to figure 7 we show the results for the resulting MAP_2 in figure 8. Concerning the distribution of interarrival times which is shown in the upper part of the figure we see that the curve corresponding to the simulatively transformed load exhibits a strong, abrupt increase at $t = 2.7ms$ which corresponds to the deterministic rate at which fragments of maximum length depart from the system if no tokens are initially present. We cannot capture this characteristic exactly but as can be seen the properties with respect to the interarrival times can be matched with good accuracy. For the mean rate which is shown in the middle part of figure 8 we note that the proposed transformation procedure is able to capture the properties of the modeled system with very good accuracy. Similarly to the preceeding experiment the maximum rate of the analytically transformed load deviates locally from the maximum rate of the simulatively transformed load but the behaviour with respect to the maximum rate is clearly captured.

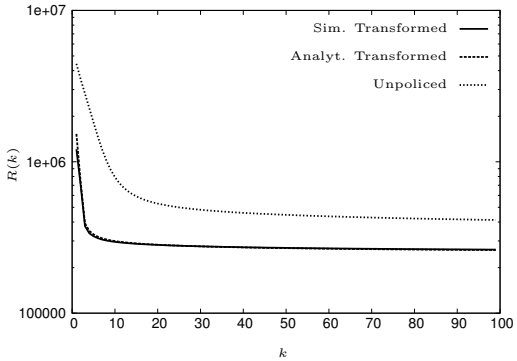
As in the dropping scenario we also evaluated the accuracy of the transformed load for several parametrisations of the token bucket module. Here too the quality is continuously rather close to that of presented results. This is illustrated by the ratio of induced bytes for the second scenario in table 3.

Table 3: Ratio of induced bytes of the analytical model compared to the simulative model of the transformed MAP_2 in percent for different n_a

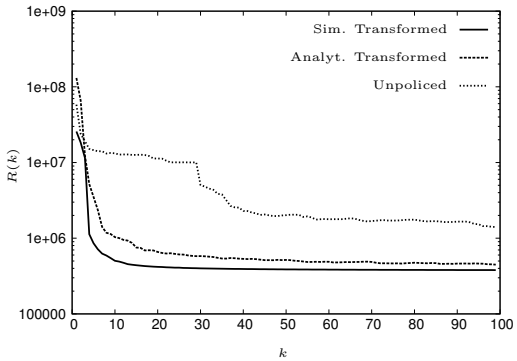
$n_a = 1$			
Rate R_t |Bucket Size	1	3	5
1.0	100.7%	101.3%	101.6%
1.2	100%	101.4%	102.2%
1.6	98.7%	101.3%	101.1%
$n_a = 3$			
Rate|Bucket Size	1	3	5
1.0	101.3%	102%	101.8%
1.2	101.4%	102.5%	102.1%
1.6	101.5%	101.5%	101.7%
$n_a = 5$			
Rate|Bucket Size	1	3	5
1.0	102.1%	102.1%	102.0%
1.2	102.6%	102.3%	101.2%
1.6	101.7%	102%	100.1%



(a) Interarrival Times ($R_t = R; n_t = 3$)



(b) Mean Rate ($R_t = R; n_t = 3$)



(c) Maximum Rate ($R_t = R; n_t = 3$)

Figure 8: Comparison of the mean and maximum Arrival Rate of the transformed MAP_2 for varying Token Bucket parameters

5. CONCLUSIONS

In this paper we presented the concept of load transformation for modeling loads at different interfaces in computer networks. In particular we discussed load transformations of *Batch Markovian Arrival Processes* (BMAPs) which we

Digital Object Identifier: 10.4108/ICST.VALUETOOLS2008.4417
http://dx.doi.org/10.4108/ICST.VALUETOOLS2008.4417

use as a characterization of primary as well as of secondary load. This approach applies to all types of primary load which can be validly described by BMAPs. Our modeling approach was illustrated by means of a transformation procedure describing the influence that a token bucket regulator has on arrival sequences described by BMAPs.
Furthermore the requirements for a tool which allows the generation of process descriptions, which are transformed possibly multiple times, in terms of BMAPs were identified. Based on these requirements we developed an architecture for an according load transformer which constituted the base for the realized tool. This tool which was presented as one of the main contributions of this paper allows for the specification of primary loads and complex transformation sequences in a convenient and intuitive manner. Finally we demonstrated the potentials and benefits of the presented tool by means of a case study. Moreover this case study documents exemplarily that complex sequences of transformations can be modeled with good accuracy with our modeling approach. We plan to combine the presented tool with an existing load generator (*UniLoG*, cf. [3]) in order to evaluate the modeling approach in real computer networks by measurements. Furthermore we want to use the transformer in advanced courses on Traffic Engineering and Performance Evaluation in order to teach students our modeling approach and transformation processes in computer networks in general. We also continue to develop new transformation procedures especially in the context of connection-oriented protocols.

Acknowledgements

This research has been supported by the *Deutsche Forschungsgemeinschaft* (DFG) as part of the "LUPUS" project.

6. REFERENCES

[1] G. Bai. *Load measurements and modeling for distributed multimedia applications in high-speed networks*. PhD thesis, University of Hamburg, 1999.

[2] A. Chydzinski and R. Winiarczyk. Blocking Probability in a BMAP Queue. In *ISCC '06: Proc. 11th IEEE Symp. on Computers and Communications*, pages 547–553, 2006.

[3] J. Cong and B.E. Wolfinger. A Unified Load Generator Based on Formal Load Specification and Load Transformation. In *ValueTools 2006, Proc. of the 1st Int. Conf. on Performance Evaluation Methodologies and Tools*, 2006.

[4] P.J. Courtois. *Decomposability: Queueing and Computer System Applications*. Academic Press, 1977.

[5] H.-W. Ferng and J.-F. Chang. Departure Processes of BMAP/G/1 Queues. *Queueing Syst. Theory Appl.*, 39(2-3):109–135, 2001.

[6] S. Heckmüller and B.E. Wolfinger. Load Transformations for Markovian Arrival Processes. In *ASMTA 2007*, pages 35–43, June 2007.

[7] S. Heckmüller and B.E. Wolfinger. Modellierung verlustinduzierender Lasttransformationen für markovsche Ankunftsprozesse. In *MMBnet 2007 Workshop*, pages 36–49, 2007. (in German).

[8] S. Heckmüller and B.E. Wolfinger. Analytical Modeling of Token Bucket Based Load Transformations. In *SPECTS 2008*, June 2008.

[9] H. Hermanns, J.-P. Katoen, J. Meyer-Kayser, and M. Siegle. A tool for model-checking Markov Chains. *Int. J. on Softw. for Technology Transfer*, 4(2):153–172, 2003.

[10] T. Hofkens, K. Spaey, and C. Blondia. Transient Analysis of the D-BMAP/G/1 Queue with an Application to the Dimensioning of a Playout Buffer for VBR Video. In *NETWORKING*, pages 1338–1343, 2004.

[11] J. Hofmann. The BMAP/G/1 Queue with Level-Dependent Arrivals - An Overview. *Telecommunication Systems*, 16(3-4):347–359, 2001.

[12] W. M. Johnston, J. R. P. Hanna, and R. J. Millar. Advances in dataflow programming languages. *ACM Comput. Surv.*, 36(1):1–34, 2004.

[13] A. Klemm, C. Lindemann, and M. Lohmann. Modeling IP traffic using the batch Markovian arrival process. *Perform. Eval.*, 54(2):149–173, 2003.

[14] H. W. Lee and J. W. Baek. Threshold Workload Control in the BMAP/G/1 Queue. In *Int. Conf. on the Quantitative Evaluation of Systems (QEST)*, pages 353–364. IEEE, 2006.

[15] D. M. Lucantoni. New results for the single server queue with a batch Markovian arrival process. *Stoch. Mod.*, 7:1–46, 1991.

[16] D. M. Lucantoni. The BMAP/G/1 queue: a tutorial. In *Models and Techniques for Performance Evaluation of Computer and Communication Systems*, pages 330–358. Springer-Verlag, 1993.

[17] M. F. Neuts. A versatile Markovian point process. *J. Appl. Prob.*, 16:764–779, 1979.

[18] A. V. Ratzer, L. Wells, and H. M. Lassen et al. CPN Tools for Editing, Simulating, and Analysing Coloured Petri Nets. In *Proc. 24th Int. Conf. on Applications and Theory of Petri Nets 2003, ICATPN 2003*, pages 450–462, 2003.

[19] A. S. Tanenbaum. *Computer Networks*. Prentice-Hall, 4 edition, 2003.

[20] Malathi Veeraraghavan, Nabeel Cocker, and Tim Moors. Support of Voice Services in IEEE 802.11 Wireless LANs. In *INFOCOM*, pages 488–497, 2001.

[21] B. E. Wolfinger. Characterization of Mixed Traffic Load in Service-Integrated Networks. *Systems Science Journal*, 25(2):65–86, 1999.

[22] B.E. Wolfinger, M. Zaddach, K.D. Heidtmann, and G. Bai. Analytical modeling of primary and secondary load as induced by video applications using UDP/IP. *Computer Communications*, 25(11-12):1094–1102, 2002.

[23] wxWidgets - Cross Platform GUI Library. http://www.wxwidgets.org. Last Access: March 2008.

[24] Q. Zhang, A. Heindl, and E. Smirni. Models of the departure process of a BMAP/MAP/1 queue. *SIGMETRICS Perform. Eval. Rev.*, 33(2):18–20, 2005.

Digital Object Identifier: 10.4108/ICST.VALUETOOLS2008.4417
http://dx.doi.org/10.4108/ICST.VALUETOOLS2008.4417