Model of the leaky bucket ATM generic flow control mechanism: a case study on solving large cyclic models

J.A.Carrasco, V.Suñé, S.Mahevas and G.Rubino

Abstract: The authors describe and solve a Markov model of the leaky bucket ATM generic flow control mechanism. The model has a space cardinality which grows quickly with its parameters and is challenging to solve. Exploiting the cyclic nature of the model, the authors develop a methodology which allows them to efficiently solve instances of the model with 3 905 134 states and 53 869 532 transitions using 29.8 Mbyte of memory and 222 Mbyte of disc storage. The CPU utilisation is high (between 70% and 90%). The methodology is new and can be easily extended to any kind of finite cyclic Markov models.

1 Introduction

Communication networks based upon the asynchronous transfer mode (ATM) provide high performance user/network interfaces [1]. The high bit rates made available by ATM allow several ATM connections to share a common medium. Examples of such media are a bus, a dual bus, or a ring. Generic flow control (GFC) mechanisms are needed to arbitrate the access to the medium. Several GFC mechanisms/protocols have heen proposed [2–4]. A simple and popular one is the leaky bucket method [5]. In [6] the performances of several GFC protocols are evaluated by simulation. In this paper we evaluate the performances of the leaky bucket protocol by numerically solving a discrete-time Markov chain model. For typical model parameters, the resulting Markov chains are large. In addition the characterisation of the irreducible closed set in which the steady-state regime is established is difficult. That characterisation is necessary to solve the model efficiently. In this paper we first characterise the irreducible closed set of the Markov chain model. Then, we develop an 'on-the-fly' model generation methodology and an associated model solution methodology, which exploit the cyclic structure of the model to reduce memory requirements to a minimum. This allows us to solve instances of the model with 3 905 134 states and 73 869 532 transitions using 29.8 Mbyte of memory and 222 Mbyte of disc storage with a CPU utilisation between 70% and 90% depending on model parameters. Our model generation and model solution methodologies can be used to solve any class of finite cyclic Markov models.

IEE Proceedings online no. 20010285

DOI: 10.1049/ip-com:20010285

Paper first received 4th November 1999 and in revised form 13th December 2000

J.A. Carrasco and V. Suñé are with the Departament d'Enginyeria Electrònica, Universitat Politècnica de Catalunya, Diagonal 647, plta. 9, 08028 Barcelona, Spain

S. Mahevas is with IFREMER, MAERHA Laboratory, Rue de l'ile d'Yeu, BP 21105, 44311 Nantes Cedex 03, France

G. Rubino is with IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France

2 Model description

The purpose of the leaky bucket GFC protocol is to regulate traffic by limiting throughput and cell clumping. The leaky bucket protocol has a parameter K equal to the maximum number of credits given to local cells. At the beginning, the number of credits is equal to K. The number of credits is decremented each time a local cell is put in the medium. The number of credits is incremented up to K at a rate $1/\Delta$ (once every Δ slots). A local cell can only be put in a free time slot (a slot not containing a cell from some other station upstream) if there are credits. If there is a waiting local cell when a free slot is received and there are not credits available, that local cell is either lost, or contained and it will wait until a credit is available, or it is accepted but marked, meaning that it may be handled differently from unmarked cells in the network. We will focus on the first case. An appropriate selection of K then allows a balance between local cell loss probability which decreases with increasing K, and cell clamping, which increases with increasing K. To reduce clamping, one should use the smallest K giving a cell loss probability smaller than or equal to the required value (for instance, 10^{-6}).

Fig. 1 shows a model of the leaky bucket protocol (proposed to the authors by Guillemin and Dupuis [7]). A first queue with infinite capacity and slotted deterministic service time 1 models contention for the medium. The queue has two incoming streams of cells: a global stream G modelling the traffic generated by the upstream stations and a local stream L modelling the traffic generated by the station under consideration. The global stream is 'discrete Poisson' with load parameter ρ (the probability that the number of arriving cells during a time slot x is $P(x) = (\rho^x/x!)e^{-\rho}$). The local stream is assumed to be deterministic with interarrival time equal to D slots. This corresponds to assuming that the station under consideration has negotiated a traffic with peak rate $1/D$ and putting ourselves in the worst case scenario in which the station under consideration generates a cell to be put in the medium every D slots. Cells from the global stream (called global) have priority over cells from the local stream (called local). This models the fact that local cells can be put in the medium only in free time slots. A cell from the global stream

departing from the first queue models the fact that the current slot is filled with a cell coming from the upstream stations. A local cell departing from the upstream models the fact that the current slot is empty and an existing local cell in the buffer of the station under consideration can be put in the medium. However, whether the local cell will be put or not in the medium is determined by the leaky bucket protocol. The leaky bucket protocol is modelled by the second queue. The number of credits available at a given time in the leaky bucket protocol is K minus the number of cells in that queue. The queue has a server with deterministic service time Δ, modelling the fact that the number of credits is incremented up to K once every Δ slots. Local cells departing from the first queue are diverted to the second queue. If the second queue is not full, this means that there are credits available and the local cell will be put in the medium. The resulting decrease by one in the number of credits is modelled by putting the local cell in the second queue. If the second queue is full there are not credits available and the local cell will be thrown away. Thus, cell losses in the local stream diverted to the second queue due to that queue being full are cell losses in the local stream caused by the leaky bucket protocol, and local cells entering the second queue correspond to accepted local cells. The parameter Δ establishes an upper bound $1/\Delta$ for the average local traffic put into the medium and, therefore, $1/\Delta$ should be as close as possible to the negotiated peak rate $1/D$. In addition, to have a small cell loss probability in the local stream, Δ should be strictly less than D. This leads to a typical selection $\Delta = D - 1$.

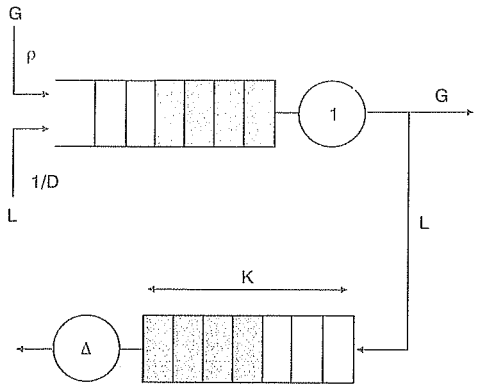

Fig. 1 *Queue model of leaky bucket based GFC protocol*

The behaviour of the model can be described by a discrete time Markov chain (DTMC), whose generic state is the state of the queueing network defined at the beginning of time slots. That state can be described by the vector (N_G, N_L, d, k, δ), where N_G and N_L are the number of cells of, respectively, the global stream and the local stream in the first queue, d is the counter of the local stream source ($1 \leq d \leq D$ and a cell arrives when $d = D$), k is the number of cells in the second queue, and δ is the counter of the service time of the second queue ($\delta = 0$ when $k = 0$, $1 \leq \delta \leq \Delta$ when $k > 0$, and a cell leaves the queue when $\delta = \Delta$).

Since the first queue has infinite capacity, the model described previously has infinitely many states. This makes an exact solution of the model extremely difficult, if not impossible (solution techniques for models with infinite state spaces require a certain regularity in the state space and the model lacks that structure). Therefore, we decided to give to the model an approximate solution obtained by solving for increasing values of C, models with the first queue of finite capacity C, monitoring the convergence of the solution, and stopping when some appropriate convergence condition is satisfied. We decided to start with $C = 4$ and increase C by increments of two. We found this an appropriate tradeoff between reducing the number of models which are solved and minimising the size of the largest model solved. Let Π_C be the DTMC modelling the queueing network with a first queue of finite capacity C. To generate Π_C we use a function which for a given state description (N_G, N_L, d, k, δ) of a state s gives the state descriptions of the successors of s and the associated transition probabilities. This function is efficiently implemented using 28 generation rules. Each generation rule has a precondition on the state variables and specifies the successors and transition probabilities which apply when the precondition is satisfied. As an illustration, we give several generation rules in Table 1. Transitions with probabilities smaller than 10^{-50} are discarded.

3 Model generation and solution

The measure we want to compute is the cell loss probability. We took three criteria for convergence with respect to C. Let ε be a small relative tolerance parameter. The first such criterion is to have a relative difference between the cell loss probabilities computed in successive iterations smaller than or equal to ε; the second criterion is to have a relative error for the probability that the server of the first queue is serving a local cell (which is equal to $1/D$) smaller than or equal to ε; the last criterion is to have a relative error for the average number of global cells in the first queue at the beginning of time slots, which is equal to $\rho(2 - \rho)/(2(1 - \rho))$ as it easily follows using well-known results for

Table 1: Some generation rules of Π_C

Rule	Precondition	Transition probability	Successors
1	$N_G = 0 \wedge N_L = 0 \wedge d \neq D$ $\wedge\ k = 0$	$P(x)$ $Q(C)$	$(x, 0, d, +1, 0, 0)$ $(C, 0, d+1, 0, 0)$
7	$N_G = 0 \wedge N_L = 0 \wedge d \neq D$ $\wedge\ k = K \wedge \delta \neq \Delta$	$P(x)$ $Q(C - N_L)$	$(x, N_L - 1, d+1, K, \delta+1)$ $(C - N_L, N_L - 1, d+1, K, \delta+1)$
13	$N_G > 0 \wedge d \neq D \wedge k = 1$ $\wedge\ \delta = \Delta$	$P(x)$ $Q(C - N_G - N_L)$	$(N_G + x - 1, N_L, d+1, 0, 0)$ $(C - N_L - 1, N_L, d+1, 0, 0)$
18	$N_G > 0 \wedge d = D \wedge k > 1$ $\wedge\ \delta = \Delta$	$P(x)$ $Q(C - N_G - N_L)$	$(N_G + x - 1, N_L + 1, 1, k-1, 1)$ $(C - N_L - 1, N_L, 1, k-1, 1)$
20	$N_G = 0 \wedge N_L > 0 \wedge d = D$ $\wedge\ k > 0 \wedge k < K \wedge \delta \neq \Delta$	$P(x)$ $Q(C - N_L)$	$(x, N_L, 1, k+1, \delta+1)$ $(C - N_L, N_L - 1, 1, k+1, \delta+1)$

$P(x) = (e^k/k!)e^{-\rho}$ is the probability that x global cells arrive in a time slot
$Q(y) = \Sigma_{z=y}^{\infty} P(z)$ is the probability that y or more global cells arrive in a time slot
The integer variable x in $P(x)$ extends from 0 to $y-1$ (for instance, for rule 1 we have $y = C$)

the M/G/1 queue [8], smaller than or equal to ε. The last two criteria were added to increase confidence in the procedure, since we are approximating an infinite model with a finite one. In the following we will describe the methodology used to generate and solve the models Π_C.

The space of feasible states of Π_C is $\Omega_C = \{(N_G, N_L, d, k, \delta), N_G \geq 0, N_L \geq 0, N_G + N_L \leq C, 1 \leq d \leq D, 0 \leq k \leq K, \delta = 0$ if $k = 0$, $1 \leq \delta \leq \Delta$ if $k > 0\}$. Detailed analysis of Π_C reveals that some of the states in Ω_C are transient. Intuitively, it is clear that Π_C has a single irreducible closed set (we use the theory of the classification of states of finite DTMCs as presented in [9]), since otherwise the system would not show steady-state behaviour independent of its initial state. However, we should prove that formally and, furthermore, characterise the irreducible closed set. Given the structure of Π_C, an explicit characterisation of its irreducible closed set is extremely difficult. However, the following theorem characterises it implicitly:

Theorem 1: Assume $D > 1$ and $\Delta < D$. Let S_C be the subset of Ω_C including the state $u = (0, 1, 1, 0, 0)$ and the states reachable in Π_C from u. Then, S_C is the only irreducible closed set of Π_C.

Proof: See the Appendix.

In the following we will consider Π_C restricted to S_C. Then, theorem 1 allows the restricted DTMC Π_C to be obtained easily by simply generating the model from state u. The DTMC Π_C is periodic with period D. This follows easily by considering that all successors of states (N_G, N_L, d, k, δ), $1 \leq d < D$ are of the form $(N'_G, N'_L, d + 1, k', \delta')$ and that all successors of states (N_G, N_L, D, k, δ) are of the form $(N'_G, N'_L, 1, k', \delta')$. Let S_C^i denote the subset of S_C including the states with $d = i + 1$. Note that the successors of S_C^i belong to $S_C^{i+1 \bmod D}$. For values of interest for C, K, D and Δ, the number of states of Π_C can be extremely large (of the order of tens of millions). To reduce memory and disc storage requirements to a minimum, we decided to use 'on-the-fly' model generation and model solution techniques [10]. Using 'on-the-fly' techniques, the successors of a given state and the corresponding transition probabilities are obtained dynamically as required by the model generation and model solution algorithms, thus avoiding the storage in memory or disc of the transition probability matrix of the model.

A standard way of reducing memory requirements when dealing with large models is to use keys instead of state descriptions. To support the use of keys, two functions are required: an encoding function computing the key from a state description and a decoding function obtaining the state description from a key. Since $0 \leq N_G \leq C$, $0 \leq N_L \leq C$, $1 \leq d \leq D$, $0 \leq k \leq K$ and $0 \leq \delta \leq \Delta$, a suitable encoding function is:

$$\begin{aligned} b(N_G, N_L, d, k, \delta) = {} & N_G(C+1)(D+1)(K+1)(\Delta+1) \\ & + N_L(D+1)(K+1)(\Delta+1) \\ & + d(K+1)(\Delta+1) + k(\Delta+1) + \delta \end{aligned}$$

The decoding function can be implemented by:

$$\begin{aligned} \delta &= b \bmod(\Delta+1), & b_1 &= \lfloor b/(\Delta+1) \rfloor, \\ k &= b_1 \bmod(K+1), & b_2 &= \lfloor b_1/(K+1) \rfloor, \\ d &= b_2 \bmod(D+1), & b_3 &= \lfloor b_2/(D+1) \rfloor, \\ N_L &= b_3 \bmod(C+1), & N_G &= \lfloor b_3/(C+1) \rfloor. \end{aligned}$$

We have represented keys using 32-bit-long unsigned integer variables. This sets a maximum value for a key of $2^{32} - 1 = 4\,294\,967\,295$, which has been enough in all the instances of Π_C we have tried.

Model generation requires a procedure of test of existence and insertion of a state with a given key into a subset of already generated states. If the key exists, the procedure should return the index of the state. If the key does not exist, the key should be inserted with a state index one unit greater than the maximum index in the subset. Holding the keys and the associated state indices in a 2–3 tree [11] indexed by keys, the operation can be done in $O(\log n)$ time, where n is the number of states held in the subset.

We are now ready to present the model generation algorithm. The purpose of the algorithm is to identify S_C. According to theorem 1, S_C can be obtained by generating all its states starting from the state $u = (0, 1, 1, 0, 0)$. The memory requirements of 'on-the-fly' techniques are typically vectors or data structures (e.g. search trees) with size equal to the cardinality of the generated state space. However, the cyclic nature of the model allows us to translate those requirements to disc and reduce the memory requirements to the components associated with two subsets S_C^i. This is achieved by generating the state space following a breadth-first approach. We start by state u which belongs to S_C^0 and generate all its successors, which will belong to S_C^1. Then, we expand the successors of u, obtaining states in S_C^2, and so on. It is clear that the generation process can be organised in steps. At each step we expand the generated and unexpanded states of a subset S_C^i and obtain states of $S_C^{i+1 \bmod D}$ which, if new, are added. The generation process can be finished when a complete cyclic sequence of subsets S_C^i has been visited without expanding any state. If data structures are partitioned according to the subsets S_C^i, only the data structures associated with the subset S_C^i from which states are expanded and the subset $S_C^{i+1 \bmod D}$ have to be kept in memory, holding in disc the data structures associated with the other subsets. We give in Fig. 2 a description of the generation algorithm. We use sequential files *file_i*, $0 \leq i \leq D - 1$, where *file_i* holds the data structures associated with a subset S_C^i. These data structures include the number of generated states, the last expanded state, an array of keys giving for each state index the corresponding key, and a 2–3 tree holding the keys of the generated states and, for each key, the corresponding state index. The function look_and_insert(*key_tree*, *b*, *&n*) looks for the key *b* in the search tree *key_tree*, returning YES and, in *n*, the index of the state, if the key exists, and returning NO and inserting the key *b* into the tree with index one unit greater than the maximum index in the tree, if the key does not exist. The variable *previous_done* is set to YES when no state has been expanded in a sequence of cyclically consecutive subsets S_C^i ending in the previously visited subset. When *previous_done* is equal to YES, the variable *first_done* is equal to the index i of the subset S_C^i starting the cyclic sequence of 'done' subsets.

We next describe the solution procedure. Since Π_C is periodic, it does not have a steady-state probability distribution. However (see, for instance [9]) it has an invariant measure $\boldsymbol{\nu} = (\nu_i)_{i \in S_C}$, where ν_i can be interpreted as the long term average frequency of visits to state i.

The invariant measure $\boldsymbol{\nu}$ is the only normalised vector ($\boldsymbol{\nu}^T \mathbf{1} = 1$, where $\mathbf{1}$ is a vector with all its components equal to 1) satisfying the linear system:

$$\boldsymbol{\nu}^T Q = \boldsymbol{\nu}^T \tag{1}$$

where Q is the transition probability matrix of Π_C. Defining $A = Q - I$, eqn. 1 can be written as

$$\boldsymbol{\nu}^T A = \mathbf{0}^T \tag{2}$$

where $\mathbf{0}$ is a vector of the appropriate dimension with all its elements equal to 0. The cell loss probability can be

```
n_states = 1; last_exp = 0;
key[1] = encode(D,C,K,Δ,(0,1,1,0,0));
make empty tree key_tree;
answer = look_and_insert(key_tree, b, &n);
write n_states, last_exp, key[] and key_tree in file file_0;
for (i = 1; i ≤ D − 1; i++){
   n_states = 0; last_exp = 0;
   make empty array key[];
   make empty tree key_tree;
   write n_states, last_exp, key[] and key_tree in file file_i;
}
end = NO;
previous_done = NO;
f = 0;
while (!end){
   read n_states_f and last_exp_f from file file_f;
   if (last_exp_f == n_states_f){
      if (!previous_done){
         previous_done = YES;
         first_done = f;
      }
      else if (first_done == f + 1 mod D) end = YES;
   }
   else previous_done = NO;
   if (!end){
      read key_f[] and key_tree_f from file file_f;
      t = f + 1 mod D;
      read n_states_t, last_exp_t, key_t[] and key_tree_t from file file_t;
      for (i = last_exp_f + 1; i ≤ n_states_f; i++){
         decode(D,C,K,Δ,key_f[i], &(N_G, N_L, d, k, δ));
         obtain the set S of descriptions of the successors of (N_G, N_L, d, k, δ);
         for (each (N_G, N_L, d, k, δ) ∈ S){
            b = encode(D,C,K,Δ,(N_G, N_L, d, k, δ));
            if (!look_and_insert(key_tree_t, b, &n)){
               n_states_t++;    /* new state */
               key_t[n_states_t] = b;
            }
         }
      }
      last_exp_f = n_states_f;
      write n_states_f, last_exp_f, key_f[] and key_tree_f in file file_f;
      write n_states_t, last_exp_t, key_t[] and key_tree_t in file file_t;
      f = t;
   }
}
```

Fig. 2 *Model generation algorithm*

obtained as $lp = lr/(1/D)$, where lr is the cell loss rate. The quantity lr can be computed by adding the ν_is of the states in which a cell is lost. These states are those with $N_G = 0$, $N_L > 0$ and $k = K$. The other quantities to be computed are P, the probability that the server of the first queue is serving a local cell, and B, the average number of global cells in the first queue at the beginning of a time slot. They can also be easily obtained from $\boldsymbol{\nu}$ and the corresponding state descriptions: P can be obtained by adding the ν_is of the states with $N_G = 0$ and $N_L > 0$; B can be computed by adding the products of the ν_is by N_G.

Sorting the states according to the succession $S_C{}^0$, $S_C{}^1$, ..., $S_C{}^{D-1}$, we obtain for A the block structure:

$$A = \begin{pmatrix} -I_0 & Q_{0,1} & 0 & \dots & 0 \\ 0 & -I_1 & Q_{1,2} & \dots & 0 \\ 0 & 0 & -I_2 & \dots & 0 \\ & & \dots & & \\ Q_{D-1,0} & 0 & 0 & \dots & -I_{D-1} \end{pmatrix}$$

where I_i denote identity matrices of appropriate dimensions and $Q_{i,j}$ is the restriction of Q to pairs in $S_C{}^i \times S_C{}^j$. Such an ordering guarantees the convergence of the Gauss–Seidel method ([12], Section 7.4.2). SOR is an iterative method which is usually used to speed up Gauss–Seidel iteration. For matrices with the block structure of A, under the condition that all eigenvalues of $\boldsymbol{J}^D$, where $\boldsymbol{J}$ is the Jacobi iteration matrix, are real and non-negative, a theory is available ([12], Section 7.7.1) providing the optimum relaxation parameter value. Computation of the optimum value for the relaxation parameter requires the knowledge of the convergence factor for Jacobi $\rho(\boldsymbol{J})$, which, using the known relationship ([12], Section 7.7.1) between the eigenvalues of the Jacobi and Gauss–Seidel iteration matrices, can be computed from the convergence factor of the Gauss–Seidel, $\rho(G)$, which can be estimated using:

$$\rho(G) = \lim_{k\to\infty} \frac{\left\|\nu^{(k+1)} - \nu^{(k)}\right\|_\infty}{\left\|\nu^{(k)} - \nu^{(k-1)}\right\|_\infty}$$

We performed numerical experiments and found that: (i) stabilisation of the estimate for $\rho(G)$ takes, for large models, a very substantial portion of the iterations needed for the convergence under the Gauss–Seidel method (about 20%), (ii) the computed ωs are far from the optimum ones (due to the fact that J^D has negative or complex eigenvalues). Thus, the conditions behind the SOR optimisation theory do not seem to hold for the matrices of our models and, given the large number of iterations required for the stabilisation of the estimates of the convergence factors, we decided that trying to develop a sophisticated adaptive SOR optimisation procedure such as that proposed in [13] was not worthwhile. Thus, we decided to use the Gauss–Seidel method.

Denoting the elements of Q by q_{ij}, the Gauss–Seidel iterative step applied to the solution of eqn. 2 can be described as:

$$\nu_i^{(k+1)} = \sum_{j<i} q_{ji}\nu_j^{(k+1)} + \sum_{j>i} q_{ji}\nu_j^{(k)}, \quad i = 1, 2, \ldots, |S_C| \quad (3)$$

The trivial implementation of eqn. 3 requires access to the elements of Q in a column-by-column manner. This is inconvenient, since it requires functions running backwards in the model and these functions are difficult to obtain. To solve the problem, an algorithm has been proposed in [10] that, at the cost of requiring two vectors instead of one, implements the Gauss–Seidel iteration with access to the elements of the matrix Q in a row by row manner. However, the cyclic nature of matrix Q allows a natural implementation of the Gauss–Seidel iteration in which the elements of the matrix are also accessed in a row by row manner. To see that, it is enough to note that all predecessors of $S_C{}^i$ are in $S_C{}^{i-1 \bmod D}$ and that this subset has only successors in $S_C{}^i$. Then, to perform, for instance, the iterative step on the states of $S_C{}^0$, it is enough to generate successors of states $j \in S_C{}^{D-1}$ and, for each successor $i \in S_C{}^0$, to cumulate in ν_i the contribution $q_{ji}\nu_j$. This observation leads to the algorithm given in Fig. 3. First, the files *file_i* left by the algorithm given in Fig. 2 are read, the iteration vectors initialised with all components equal to 1 and the files written with the information required by the solu-

```
old_lr = 0; old_P = 0; old_B = 0;
for (i = 0; i ≤ D − 1; i++){
   read n_states_f, last_exp_f, key_f[] and key_tree_f from file file_i;
   for (j = 1; j ≤ n_states; j++){
      ν_f[j] = 1;
      decode(D,C,K,Δ,key_f[j], &(N_G, N_L, d, k, δ));
      update_contributions((N_G, N_L, d, k, δ), ν_f[j], &old_lr, &old_P, &old_B);
   }
   write n_states_f, key_f[], key_tree_f and ν_f[] in file file_i;
}
read n_states_t, key_t[], key_tree_t and ν_t[] from file file_D − 1;
end = NO;
while (!end){
   lr = 0; P = 0; B = 0; sum_ν = 0;
   for (t = 0; t ≤ D − 1; t++){
      n_states_f = n_states_t; key_f[] = key_t[]; key_tree_f = key_tree_t; ν_f[] = ν_t[];
      read n_states_t, key_t[], key_tree_t and ν_t[] from file file_t;
      for (i = 1; i ≤ n_states_t; i++) ν_t[i] = 0;
      for (i = 1; i ≤ n_states_f; i++){
         decode(D,C,K,Δ,key_f[i], &(N_G, N_L, d, k, δ));
         obtain the set S of descriptions of the successors of (N_G, N_L, d, k, δ)
         and the corresponding transition probabilities;
         for (each (N_G, N_L, d, k, δ) ∈ S){
            let q be the transition probability associated with (N_G, N_L, d, k, δ);
            b = encode(D,C,K,Δ,(N_G, N_L, d, k, δ));
            answer = look_and_insert(key_tree_t, b, &j);
            ν_t[j] += q × ν_f[i];
         }
      }
      for (i = 1; i ≤ n_states_t; i++){
         sum_ν += ν_t[i];
         decode(D,C,K,Δ,key_t[i], &(N_G, N_L, d, k, δ));
         update_contributions((N_G, N_L, d, k, δ), ν_t[i], &lr, &P, &B);
      }
      write n_states_t, key_t[], key_tree_t and ν_t[] in file file_t;
   }
   if (|lr − old_lr|/lr ≤ ε' && |P − old_P|/P ≤ ε' && |B − old_B|/B ≤ ε')
      end = YES;
   else end = NO;
   old_lr = lr; old_P = P; old_B = B;
}
lp = lr × D/sum_ν; P = P/sum_ν; B = B/sum_ν;
```

Fig. 3 *Model solution algorithm using Gauss–Seidel method with access to elements of Q in row-by-row manner*

tion algorithm: number of states, array of keys, key tree and iteration vector. The function update_contributions() updates the unnormalised contributions to lr, P and B of a state with given description. The iteration step involves looping through the subsets S_C^t from $t = 0$ to $D - 1$. In each iteration of the loop, *n_states_f*, *key_f*[], *key_tree_f* and *v_f*[] hold the data structures of the predecessor subset $S_C^{t-1 \bmod D}$ and *n_states_t*, *key_t*[], *key_tree_t* and *v_t*[] hold the data structures of the current subset S_C^t. The iteration vector is kept unnormalised during the Gauss–Seidel iterations. Convergence is considered achieved when the relative tolerance in all the unnormalised quantities lr, P and B is smaller than or equal to a given small quantity ε', significantly smaller than ε. Once convergence is achieved, the cell loss probability lp and the normalised P and B are computed using the sum of the components of the iteration vector which is cumulated in *sum_v*. Note that, at a given time, only the data structures associated with subsets S_C^t and $S_C^{t-1 \bmod D}$ are held in memory.

4 Results

Table 2 illustrates the more relevant parameters related to the computational effort of the solution of the model. For several values of ρ, D, and K ($\Delta = D - 1$ in all cases) and $\varepsilon = 10^{-5}$, $\varepsilon' = 10^{-10}$, we give the value of C for which the solution converged. For that value of C, we also give the number of states, number of transitions, memory and disc storage requirements in Mbytes, and number of required Gauss–Seidel iterations. The amount of disc storage is approximately the amount of memory which would be required were our techniques to reduce it based on the cyclic nature of the model not implemented. The required value of C increases with K and is very sensitive to the utilisation factor of the first queue ($\rho + 1/D$). The required number of Gauss–Seidel steps is also sensitive to the utilisation factor of the first queue and increases slightly with K. We can note that memory requirements are very small relative to the number of states and transitions of the models. Our code reads arrays from disc and writes arrays into disc using, respectively, the ANSI standard C functions `fread()` and `fwrite()`. CPU utilisation was high (between 70% and 90%, depending basically on the value of C). That high CPU utilisation is due to the fact that the average number of transitions per state is high (the contribution to the wall clock time due to disc accesses is approximately proportional to the number of states while the CPU processing time is approximately proportional to the number of transitions). Thus, the price paid by our techniques to reduce the memory requirements exploiting the cyclic nature of the model is small. The flop rate our code achieved in a 167MHz UltraSPARC1 workstation was about 174Kflops, which is reasonable considering the use of the 2–3 trees to find the indices of the states from their keys.

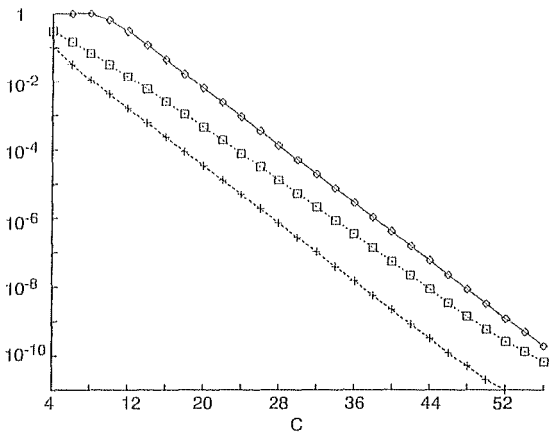

Fig. 4 *Relative tolerance in loss rate (criterion 1) and relative errors for probability that server of first queue is serving a local cell (criterion 2) and expected number of global cells in first queue (criterion 3) for increasing C, for case $\rho = 0.7$, $D = 10$, $\Delta = 9$, $K = 4$*
—◇— criterion 1
– –+– – criterion 2
···□··· criterion 3

Fig. 4 gives the relative tolerance for the loss rate and the relative errors for the probability that the server of the first queue is serving a local cell and the expected number of global cells in the first queue for increasing C, for the case $\rho = 0.7$, $D = 10$, $\Delta = 9$, $K = 4$ and $\varepsilon' = 10^{-14}$. We can note first that both the relative tolerance and the relative errors decrease at a high rate when C increases. Then, we should expect the relative error in the loss rate to be smaller than the relative tolerance. Also, of the two other criteria, the strongest is the third one. For this particular set of parameters, the required C would be determined by the first criterion.

Figs. 5–7 display the loss probability as a function of K for, respectively, $\rho = 0.3$, $\rho = 0.5$ and $\rho = 0.7$, for $D = 5$, $D = 10$ and $D = 15$, and $\Delta = D - 1$ in all cases. All results were obtained using $\varepsilon = 10^{-5}$ and $\varepsilon' = 10^{-10}$. The K required to achieve a given loss probability increases with the load (i.e. with ρ, and with $1/D$). Except for the cases in which the load of the system $\rho + 1/D$ is very high, the minimum K required to achieve a loss probability smaller than or equal to a typical value is moderate. Thus, for instance, Table 3 gives the minimum required K to achieve a loss

Table 2: C required for convergence in results and number of states, number of transitions, memory and disc storage requirements in Mbytes, and number of required iterations for that value of C for several sets of model parameter values

ρ	D	K	C	States	Transitions	Memory	Disc storage	Iterations
0.5	10	2	18	29513	223968	0.341	1.64	25
0.5	10	8	32	365224	4557871	4.24	21.1	42
0.7	5	2	46	42167	708045	1.01	2.50	513
0.7	5	8	56	249129	4896546	5.66	14.0	631
0.7	5	14	66	598935	13159009	14.0	34.7	731
0.7	15	2	28	166578	1819108	1.30	9.62	49
0.7	15	8	42	1479612	23397586	11.3	84.3	70
0.7	15	14	52	3905134	73869532	29.8	222	86

$\Delta = D - 1$ in all cases

probability smaller than or equal to 10^{-6} for all pairs of ρ, D values considered in Figs. 5–7. Since the first queue has infinite capacity, the traffic entering the second queue has average rate $1/D$. The average service rate of that queue is $1/\Delta = 1/(D-1)$. Thus, the utilisation of the second queue is $(D-1)/D$, which increases with D. However, the results given in Figs. 5–7 show that the loss probability decreases with D, i.e. with the utilisation. The counter-intuitive behaviour can be explained by the fact that the burstiness of the traffic diverted to the second queue decreases with D, compensating the larger utilisation for larger D.

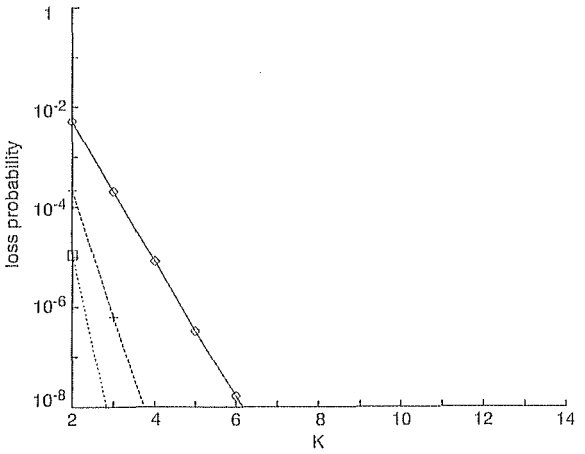

Fig. 5 *Loss probability as a function of K for $\rho = 0.3$, $D = 5$, 10, 15 and $\Delta = D - 1$*
—◇— $D = 5$
– –+– – $D = 10$
···□··· $D = 15$

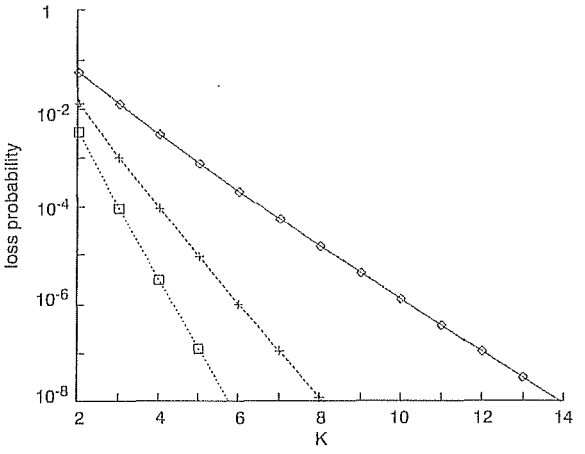

Fig. 6 *Loss probability as a function of K for $\rho = 0.5$, $D = 5$, 10, 15 and $\Delta = D - 1$*
—◇— $D = 5$
– –+– – $D = 10$
···□··· $D = 15$

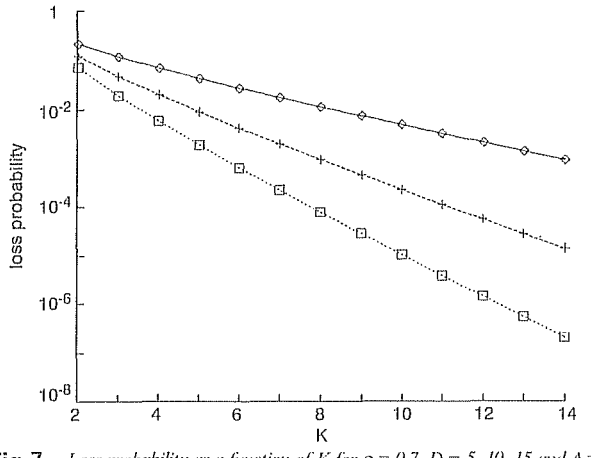

Fig. 7 *Loss probability as a function of K for $\rho = 0.7$, $D = 5$, 10, 15 and $\Delta = D - 1$*
—◇— $D = 5$
– –+– – $D = 10$
···□··· $D = 15$

Table 3: Minimum required K to achieve a loss probability smaller than or equal to 10^{-6} for several values of ρ and D

ρ	D	K
0.3	15	3
0.3	10	3
0.3	5	5
0.5	15	5
0.5	10	6
0.5	5	11
0.7	15	13
0.7	10	>14
0.7	5	>14

5 Conclusions

We have developed a methodology for the solution of finite large cyclic DTMC models and have applied it to the performance analysis of an ATM leaky bucket GFC protocol. The methodology includes 'on-the-fly' model generation techniques and exploits the cyclic nature of the model to reduce to a minimum the memory requirements. In addition, we have shown that the class of ATM leaky bucket models considered in the paper have a single irreducible closed set and have characterised this set implicitly by showing that there exists a path from any feasible state of the model to a given particular state u included in the irreducible closed set. The characterisation allows the irreducible closed set to be generated very easily (it is enough to generate states from u). The approach we have followed to show that the models have a single irreducible closed set and characterise it could be used for other classes of models. Using the methodology developed in the paper, we have been able to solve very large models (3 905 134 states and 73 869 532 transitions) using moderate amounts of memory (29.8 Mbyte) and disc storage (222 Mbyte). The results obtained can be used to select an appropriate value for the parameter K as a function of the negotiated portion ($1/D$) of the bandwidth of the medium and the monitored upstream load (ρ) depending on the required loss probability (typically between 10^{-3} and 10^{-6}). CPU time requirements are, however, large. In the future we are planning to develop and test approximations requiring the solution of smaller models so that more efficient computations can be performed. In that sense, the methodology proposed in the paper will allow the goodness of such approximations to be tested.

6 Acknowledgments

The work of the two first authors has been supported by the Comisión Interministerial de Ciencia y Tecnología (CICYT) under the research grant TIC95-0707-C02-02. The authors are grateful to the anonymous reviewers, whose comments have allowed the improvement of a former version of the paper.

7 References

1 RITTLER, M.: 'Multirate models for dimensioning and performance evaluation of ATM networks'. COST-242 report, June 1994
2 ADAMS, J.: 'A multiservice GFC protocol', *Int. J. Digit. Analog Commun. Syst.*, 1991, 4, pp. 123–130
3 BUDRIKIS, Z.I., MERCANKOU, G., BLASIKIEWIECZ, M., ZUKERMAN, M., YAO, L., and POTTER, P.: 'A generic flow control for B-ISDN'. Proceedings of IEEE Infocom'92, 1992, pp. 895–904
4 FALCONER, R., and ADAMS, J.: 'Orwell: a protocol for an integrated services local network', *British Telecommun. Tech. J.*, 1985, 3, (4), pp. 27–35
5 NIESTEGGE, G.: 'The leaky bucket policing method in ATM networks', *Int. J. Digital Analog Commun. Syst.*, 1990, 3, pp. 187–197
6 GUILLEMIN, F., and MONIN, W.: 'Analysis of cell clumping caused by ATM network GFC protocols', *Comput. Commun.*, 1994, 17, (9), pp. 637–646
7 GUILLEMIN, F., and DUPUIS, A.: private communication, 1996
8 KLEINROCK, L.: 'Queueing systems. Volume 1: Theory' (John Wiley, 1975)
9 ÇINLAR, E.: 'Introduction to stochastic processes' (Prentice-Hall, 1975)
10 DEAVOURS, D.D., and SANDERS, W.H.: 'On-the-fly solution techniques for stochastic petri nets and extensions'. Proceedings of the 7th IEEE international workshop on *Petri nets and performance models* (PNPM97), 1997, pp. 132–141
11 AHO, A.V., HOPCROFT, J.E., and ULLMAN, J.D.: 'Data structures and algorithms' (Addison-Wesley, 1985)
12 STEWART, W.J.: 'Introduction to numerical solution of Markov chains' (Princeton University Press, 1994)
13 CIARDO, G., BLAKEMORE, A., CHIMENTO, P.F., MUPPALA, J.K., and TRIVEDI, K.S.: 'Automated generation and analysis of Markov reward models using stochastic reward nets' *in* MEYER, C., and PLEMMONS, R. (Eds.): 'Linear algebra, Markov chains and queuing models' (IMA, 1992)

8 Appendix: Proof of theorem 1

We will use the theory of classification of the states of finite DTMCs as presented in [9]. It is enough to prove that there exists a path in Π_C from any state $s \in \Omega_C - \{u\}$ to u. To see that, note first that u cannot be transient, since, if it were, some state in $\Omega_C - \{u\}$ would be recurrent and the existence of a path from such state to u would contradict that u is transient (there cannot be paths from recurrent states to transient states). Not being transient, u has to belong to some irreducible closed set. Let S_C be the irreducible closed set of Π_C including u. States which are not reachable from u cannot belong to S_C. Also, every state s reachable from u belongs to S_C, since there exist paths between s and u in both directions. Thus, S_C includes precisely u and the states reachable from u. To show that S_C is the only irreducible closet set of Π_C, assume the existence of another irreducible closed set S'_C and pick any state $s \in S'_C$. The existence of a path from s to $u \notin S'_C$ is in contradiction with S'_C being a closed set.

We split the proof of the existence of a path from any state $s \in \Omega_C - \{u\}$ to u in two parts. In the first part, we prove the existence of a path from any state (N_G, N_L, d, k, δ) to a state (0, 0, d', k', δ'). In the second part, we prove the existence of a path from any state (0, 0, d, k, δ) to u. We use the notation

$$s \xrightarrow{x} s'$$

to indicate that state s' is the successor of s reached when x arrivals from the global stream occur.

First part: Consider an arbitrary state (N_G, N_L, d, k, δ) and the path made up of transitions associated with 0 arrivals of the global stream. Such a path reaches a state (0, 0, d', k', δ'). To prove that, consider first the case $N_G > 0$. Each transition will reduce N_G by 1 and after N_G transitions the path will reach a state (0, N'_L, d', k', δ'). Then, it is enough to prove the result for states (0, N_L, d, k, δ) with $N_L > 0$. Consider a path of transitions associated with 0 arrivals from the global stream. Each transition will decrement N_L if $d < D$ and will leave unchanged (or decrement if $N_L = C$) N_L if $d = D$. Since $D > 1$, a d cycle will decrement N_L and a state (0, 0, d', k', δ') will be eventually reached.

Second part: We will analyse nine cases for the values of d, k, δ, D, Δ and K. For each case we will find a path to either u or a state of the form (0, 0, d', k', δ') in such a way that the existence of a path to u will be guaranteed.

Case 1: $k = 0$

Note that $k = 0$ implies $\delta = 0$. Therefore, the state will be of the form (0, 0, d, 0, 0). A path to u is:

$$(0,0,d,0,0) \xrightarrow{0} (0,0,d+1,0,0) \xrightarrow{0} \ldots \xrightarrow{0} (0,0,D,0,0) \xrightarrow{0} (0,1,1,0,0)$$

Case 2: $\Delta - \delta > D - d + 1$, $0 < k < K$, $K > 1$

Note that $D - d + \delta + 2 \leq \Delta$ and that, with $d \leq D$ and (since $k > 0$) $\delta \geq 1$, $\Delta - D + d - \delta + 1 \leq \Delta - \delta + 1 \leq \Delta < D$. These two inequalities and $k < K$, $D \geq 2$ guarantee the existence of the path:

$$\begin{aligned}
&(0,0,d,k,\delta) \xrightarrow{0} \ldots \xrightarrow{0} (0,0,D,k,D-d+\delta) \\
&\xrightarrow{0} (0,1,1,k,D-d+\delta+1) \\
&\xrightarrow{0} (0,0,2,k+1,D-d+\delta+2) \\
&\xrightarrow{0} \ldots \xrightarrow{0} (0,0,\Delta-D+d-\delta,k+1,\Delta) \\
&\xrightarrow{0} (0,0,\Delta-D+d-\delta+1,k,1)
\end{aligned}$$

Let $d' = \Delta - D + d - \delta + 1$ and $\delta' = 1$. We have $\Delta - \delta' - (D - d') = d - \delta - 2(D - \Delta) < d - \delta - (D - \Delta) = \Delta - \delta - (D - d)$. Then, by concatenating an enough number of such paths we will reach a state corresponding to case 5, 8 or 9.

Case 3: $\Delta - \delta > D - d + 1$, $k = K > 1$

Note that $D - d + \delta + 2 \leq \Delta$ and, as shown in the discussion of case 2, $\Delta - D + d - \delta + 1 < D$. These two inequalities with $D \geq 2$ guarantee the existence of the path:

$$\begin{aligned}
&(0,0,d,K,\delta) \xrightarrow{0} \ldots \xrightarrow{0} (0,0,D,K,D-d+\delta) \\
&\xrightarrow{0} (0,1,1,K,D-d+\delta+1) \\
&\xrightarrow{0} (0,0,2,K,D-d+\delta+2) \\
&\xrightarrow{0} \ldots \xrightarrow{0} (0,0,\Delta-D+d-\delta,K,\Delta) \\
&\xrightarrow{0} (0,0,\Delta-D+d-\delta+1,K-1,1)
\end{aligned}$$

and we have reached a state corresponding to case 2, 5, 8 or 9.

Case 4: $\Delta - \delta > D - d + 1$, $k = K = 1$

Note that $D - d + \delta + 2 \leq \Delta$ and, as shown in the discussion of case 2, $\Delta - D + d - \delta + 1 < D$. Since $D \geq 2$, the following path exists:

$$\begin{aligned}
&(0,0,d,1,\delta) \xrightarrow{0} \ldots \xrightarrow{0} (0,0,D,1,D-d+\delta) \\
&\xrightarrow{0} (0,1,1,1,D-d+\delta+1) \\
&\xrightarrow{0} (0,0,2,1,D-d+\delta+2) \\
&\xrightarrow{0} \ldots \xrightarrow{0} (0,0,\Delta-D+d-\delta,1,\Delta) \\
&\xrightarrow{0} (0,0,\Delta-D+d-\delta+1,0,0)
\end{aligned}$$

and we have reached a state corresponding to case 1.

Case 5: $\Delta - \delta = D - d + 1$, $0 < k < K$, $K > 1$

Since $D \geq 2$ we have the path:

$$\begin{aligned}
&(0,0,d,k,\delta) \xrightarrow{0} \ldots \xrightarrow{0} (0,0,D,k,\Delta-1) \\
&\xrightarrow{0} (0,1,1,k,\Delta) \xrightarrow{0} (0,0,2,k,1) = s
\end{aligned}$$

Calling the 'd' and 'δ' parameters of s, respectively, d' and δ', we have $\Delta - \delta' = \Delta - 1 \leq D - 2 = D - d'$. Notice also that $k > 0$. Then, we have reached a state corresponding to case 8 or case 9.

Case 6: $\Delta - \delta = D - d + 1$, $k = K > 1$

Since $D \geq 2$ we have the path:

$$(0,0,d,K,\delta) \xrightarrow{0} \ldots \xrightarrow{0} (0,0,D,K,\Delta-1)$$
$$\xrightarrow{0} (0,1,1,K,\Delta) \xrightarrow{0} (0,0,2,K-1,1) = s$$

Calling the 'd' and 'δ' parameters of s, respectively, d' and δ', we have $\Delta - \delta' = \Delta - 1 \leq D - 2 = D - d'$. Notice also that $K - 1 > 0$. Then, we have reached a state corresponding to case 8 or case 9.

Case 7: $\Delta - \delta = D - d + 1$, $k = K = 1$

Since $D \geq 2$ we have the path:

$$(0,0,d,1,\delta) \xrightarrow{0} \ldots \xrightarrow{0} (0,0,D,1,\Delta-1)$$
$$\xrightarrow{0} (0,1,1,1,\Delta) \xrightarrow{0} (0,0,2,0,0)$$

and we have reached a state corresponding to case 1.

Case 8: $\Delta - \delta = D - d$, $0 < k \leq K$

For $k = 1$ we have the path to u:

$$(0,0,d,1,\delta) \xrightarrow{0} \ldots \xrightarrow{0} (0,0,D,1,\Delta) \xrightarrow{0} (0,1,1,0,0)$$

For $k > 1$ and $\Delta = 1$, since $D \geq 2$, we have the path:

$$(0,0,d,k,\delta) \xrightarrow{0} \ldots \xrightarrow{0} (0,0,D,k,\Delta)$$
$$\xrightarrow{0} (0,1,1,k-1,1) \xrightarrow{0} (0,0,2,k-1,1)$$

For $k > 1$ and $\Delta > 1$, since $D \geq 2$, we have the path:

$$(0,0,d,k,\delta) \xrightarrow{0} \ldots \xrightarrow{0} (0,0,D,k,\Delta)$$
$$\xrightarrow{0} (0,1,1,k-1,1) \xrightarrow{0} (0,0,2,k,2)$$
$$\xrightarrow{0} \ldots \xrightarrow{0} (0,0,\Delta,k,\Delta)$$
$$\xrightarrow{0} (0,0,\Delta+1,k-1,1)$$

In both last cases we reach a state with 'k' one less.

Case 9: $\Delta - \delta < D - d$, $0 < k \leq K$

Note that $\Delta - \delta + d + 1 \leq D$. For $k = 1$ we have the path:

$$(0,0,d,1,\delta) \xrightarrow{0} \ldots \xrightarrow{0} (0,0,\Delta-\delta+d,1,\Delta)$$
$$\xrightarrow{0} (0,0,\Delta-\delta+d+1,0,0)$$

and we reach a state corresponding to case 1. For $k > 1$ the previous path leads to (0, 0, $\Delta - \delta + d + 1$, $k - 1$, 1), a state with 'k' one less.

To summarise, cases 4 and 7 are reduced to case 1; cases 2, 3, 5 and 6 are reduced to case 8 or case 9. In case 1 there exists a path to u. In cases 8 and 9 either there exists a path to u or a path to a state with 'k' one less, guaranteeing that the case 1 ($k = 0$) will be eventually reached. Thus from any state (0, 0, d, k, δ) there exists a path to u and the proof of the theorem is finished.