Folding Algorithm: A Computational Method for Finite QBD Processes with Level-Dependent Transitions

Jingdong Ye, Member, IEEE and San-qi Li Member, IEEE

Abstract—This paper presents a new computational method for steady state analysis of finite QBD-process with level-dependent transitions. The QBD state space is defined in two-dimension with N phases and K levels. Instead of formulating solutions in matrix-geometric form, the Folding-algorithm provides a technique for direct computation of \( \pi P = \omega \), where \( P \) is the QBD generator which is an \((NK) \times (NK)\) matrix. Taking a finite sequence of fixed-cost binary reduction steps, the K-level matrix \( P \) is eventually reduced to a single-level matrix, from which a boundary vector is obtained. Each step halves the matrix size but keeps the QBD form. The solution \( \pi \) is expressed as a product of the boundary vector and a finite sequence of expansion factors. The time and space complexity for solving \( \pi P = \omega \) is therefore reduced from \( O(N^2K) \) and \( O(N^2\log K) \) to \( O(N^2\log K) \) and \( O(N^2\log K) \), respectively. The Folding-algorithm has a number of highly desirable advantages when it is applied to queuing analysis. First, the algorithm handles the multi-level control problem in finite buffer systems. Second, its total independence of the phase structure allows the algorithm to apply to more elaborate, multiple-state Markovian sources. Its computational efficiency, numerical stability and superior error performance are also distinctive advantages.

I. INTRODUCTION

For an infinite quasi-birth-death (QBD) process without level-dependent transitions, the steady state solution is in the so called matrix-geometric form, where the matrix-geometric factor is obtained from a quadratic matrix equation [1], or by spectral decomposition [2]. An alternative matrix-geometric approach is proposed in [3], which modifies the QBD-process with state space expansion. Its solution then involves a different matrix-geometric factor which is obtainable from a linear matrix equation. Once the QBD-process becomes finite, however, the solution will be in a much more complicated modified matrix-geometric form [4]. Except with special cases [1, 5, 6, 7, 8], the solution of finite QBD-process can hardly be simplified. No technique is available today for finite QBD-process with level-dependent transitions.

Here we consider a generic finite QBD-process with level-dependent transitions. Its state space is defined in two-dimension with \( N \) phases and \( K \) levels. The whole process is described by a state transition matrix \( P \) in block tridiagonal form, where each block is an \( N \times N \) submatrix. Instead of formulating solutions in matrix-geometric form, this paper introduces a new methodology, called Folding-algorithm, for direct computation of \( \pi P = \omega \), where \( \pi \) is the equilibrium solution vector. The Folding-algorithm is developed on the basis of Markov chain reduction. It exploits the QBD structure to diminish large level-induced complexity. The time and space complexity of the Folding-algorithm to solve \( \pi P = \omega \) is equal to \( O(N^2\log K) \) and \( O(N^2\log K) \), respectively. In contrast, a direct application of the block Gaussian elimination (or block LU decomposition) will yield a time and space complexity of \( O(N^3K) \) and \( O(N^3K) \) [9].

The Folding-algorithm finds extensive applications in telecommunication network research. Typically, one can use a finite QBD-process with level-dependent transitions to model a statistical multiplexer with finite buffer and Markov chain modulated input, subject to input rate regulation, buffer overload control and dynamic link capacity allocation. Its state space, \( K \times N \), in reality may well exceed \( 10^6 \) in size. This is because the number of levels \( K \), or the buffer size, can be up to around \( 10^3 \); at about the same order of magnitude is the number of phases \( N \) at each level, which is the size of input Markov chain for aggregate multimedia traffic. For practical purposes, the time complexity by Folding-algorithm is only \( O(N^3) \). This moderate complexity makes implementation on small computers feasible. Since this paper focuses on the algorithm, only a few representative examples are selected to demonstrate the stability, accuracy and efficiency of the algorithm. Extensive numerical studies can be found in [10, 11].

The paper is organized as follows. Section II describes the algorithm. Section III gives the time and space complexity, and error characteristics. Section IV shows the application to statistical multiplexing analysis.

II. THE FOLDING ALGORITHM

In this section, we first examine the structure of finite QBD-process without level-dependent transitions, review some results from Markov chain theory, and then describe the Folding-algorithm in detail. Finally we extend the algorithm with level-dependent transitions.

A. Structure of QBD Models and Solutions

Reflecting the structure of the QBD-processes, as depicted in Fig. 1, is the following transition matrix of block
Fig. 1. Transition diagram of finite QBD-process

di-triagonal form:

\[ P = \begin{bmatrix}
  A_0 & U & & \\
  D & A & U & \\
  & D & A & U \\
  & & D & A_1
\end{bmatrix} \]

(1)

where \( P \) is finite and irreducible; \( A_0, A \) and \( A_1 \) describe the phase transitions; \( U \) and \( D \) specify the level transitions.

Fundamental to the analysis is to determine the equilibrium distribution, \( \pi \), which is the unique solution to the equations:

\[ \pi P = 0, \; \pi e = 1 \]  

(2)

where \( e = [1 \; 1 \; \ldots \; 1]^T \).

Of major interest to us is the block structure of \( P \): the basic triad \((A, D, U)\) makes up the three uniform diagonals, except for \( A_0 \) and \( A_1 \) terminating the main diagonal. Our solution method exploits this type of structure to advantage. \( P \) is of size \((K \times K)\) in blocks of dimension \((N \times N)\). In reality \( N \) and \( K \) can be up to the order of \( 10^9 \). Such an enormous scale precludes obtaining \( \pi \) all at once by ordinary computers. To facilitate computation, we partition \( \pi \) as

\[ \pi = [\pi_0, \pi_1, \ldots, \pi_{K-1}] \]

(3)

where \( \pi_i \), of length \( N \), will be called the solution vector of level \( i \), \( 0 \leq i \leq K - 1 \). The algorithm only describes the inter-block manipulations. All matrices mentioned in this paper are made up by blocks and all references to the matrix size are based on the unit of one block.

B. Markov Chain Reduction

The principle of finite Markov chain (MC) reduction is pertinent to the development of the algorithm. Consider an irreducible Markov chain on a finite state space \( E \). Partition \( E \) so that \( E = S \cup T \) where both \( S \) and \( T \) are not empty. Label the states so that the transition matrix has the form

\[ P = \begin{bmatrix}
  P_s & P_{st} \\
  P_{ts} & P_t
\end{bmatrix} \]

(4)

where \( P_s \) and \( P_t \) express transitions within \( S \) and \( T \), respectively. This simple manipulation of structure has a number of very interesting consequences, as given in the following.

First, \( P_s^{-1} \) exists, as implied by the irreducibility of \( P \). It can be shown that \((-P_s^{-1})\) is the mean sojourn times of the MC on \( S \), conditional on the initial state when the process first enters \( S \). We can construct a new MC from the original process by observing its behavior on the subset \( T \) only. This new MC has its transition matrix \( P'_s \), given by

\[ P'_s = P_t + P_{ts}(-P_s^{-1})P_{st}. \]

(5)

Letting \( \pi = [\pi_s, \pi_t] \) be such that \( \pi P = 0 \), we can verify that

\[ \pi_t P'_s = 0. \]

(6)

Once the vector \( \pi_s \) is obtained from (6), \( \pi_t \) is uniquely determined by

\[ \pi_t = -\pi_t P_{ts} P_s^{-1}. \]

(7)

An equivalent form of (7) is more useful for the later presentation of the algorithm:

\[ [\pi_s, \pi_t] = \pi_t [-P_{ts} P_s^{-1}, I]. \]

(8)

Materials presented above can be found in [12, 13]. It is also interesting to look at the above results from a purely algebraic point of view. The existence of \( P_s^{-1} \), together with (5), readily provides a factorization of the matrix \( P \) in (4):

\[ P = \begin{bmatrix}
  P_s & P_{st} \\
  P_{ts} & P_t
\end{bmatrix} = \begin{bmatrix}
  P_s & 0 \\
  P_{ts} & P_t
\end{bmatrix} \begin{bmatrix}
  I & 0 \\
  0 & P_s^{-1} P_{st}
\end{bmatrix}. \]

(9)

Since the second factor above is nonsingular, \( \pi P = 0 \) is reduced to

\[ [\pi_s, \pi_t] \begin{bmatrix}
  P_s & 0 \\
  P_{ts} & P_t
\end{bmatrix} = 0, \]

(10)

which leads to (6) and (7).

The proceeding results suggest a practical approach to large-scale Markov chain analysis: first reduce a large MC, as many times as necessary, to obtain a new MC whose size allows for direct solution; then reconstruct the full solution in reverse steps. The following development is based on this Markov chain reduction principle.

C. Basic Procedure

The folding algorithm operates in two phases and one intermediate step, in this order: first the forward reduction phase, then the middle step to obtain a boundary vector, and finally the backward expansion phase. It is concerned with block by block manipulations on the matrix \( P \). The underlying matrix-algebraic operations, such as inverse and multiplication of blocks, are taken as atomic.

In the reduction phase, the original \( K \)-level MC matrix, \( P \), is eventually reduced to a one-level MC matrix by a sequence of reduction steps. Each step produces a child MC matrix with only half the size of its parent according to (5). A key property in reduction is that, with proper block permutation on the parent MC matrix, the child MC matrix will inherit the QBD structure from its parent. The amount of computations in each step is then kept basically identical, regardless of the sizes of intermediate MC matrices. The finally reduced one-level MC matrix, of size one block, is then solved directly by any of the standard methods to obtain the boundary solution vector.

The expansion phase is a reverse process of the reduction phase. It follows exactly the reverse order in which
the sequence of child MC matrices are generated. A key observation is that the solution for each child MC matrix provides an exact half of the solution for its parent. The other half of the solution is then constructed based on (7). Starting from the boundary solution vector, the expansion process will find the solution of each intermediate child MC matrix, in reverse order. In each step, the known portion of the full solution \( \pi \) is therefore doubled in size. The expansion phase is expeditious: no more than three vector-matrix multiplications are required to compute each \( \pi_i \) in (3).

1) Forward Reduction Phase: The objective of this phase is to reduce \( P \) to a MC matrix of size 1. We use a simple example for illustration. Let \( P_0 = P \) as defined in (1), with a convenient size \( K = 2^n \). For this example, let us go through the first step at \( n = 3 \). The subsequent steps are similar.

First, permute both the rows and columns of the blocks in \( P_0 \) according to

\[
\begin{pmatrix}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7
\end{pmatrix},
\]

which means 1 goes to 4's place, 2 goes to 1's place, and so on. As a result, the original order \([0, 1, 2, \ldots, 7]\) becomes \([0, 2, 4, 6, 1, 3, 5, 7]\). The permuted matrix, denoted by \( P_0 \), is further partitioned as follows:

\[
P_0 = \begin{bmatrix}
A_0 & U & D & U & D \\
A & D & U & D & U \\
D & U & A & D & U \\
D & U & D & A & U \\
D & D & D & D & A
\end{bmatrix}
\]

where \( P_s, P_t, P_a, \) and \( P_t \) denote the four parts in the partition.

Next, perform reduction on the \( P_0 \) according to (5). A child MC matrix \( P_1 \) is produced based on

\[
P_1 = P_t - P_{st} P_s^{-1} P_{st}.
\]

One can easily verify

\[
P_1 = \begin{bmatrix}
A^{(1)}_0 & U^{(1)} \\
D^{(1)} & A^{(1)} & U^{(1)} \\
D^{(1)} & A^{(1)} & U^{(1)}
\end{bmatrix},
\]

with

\[
A^{(1)}_0 = A - DA_0^{-1} U - UA_0^{-1} D, \\
A^{(1)} = A - DA_0^{-1} U - UA_0^{-1} D, \\
A^{(1)}_1 = A_1 - DA^{-1} U, \\
U^{(1)} = -UA^{-1} U, \\
D^{(1)} = -DA^{-1} D.
\]

Thus, the child \( P_1 \) inherits the original QBD structure from its parent \( P_0 \), but with only half the size. It is clear that the permutation of \( P_0 \) before reduction is vital to achieve this effect. Note that the permutation is implicit. It is used to derive (14), which are the actually required computations. Each child matrix, thus produced in a reduction step, will become a parent in the next reduction step, until the matrix is finally reduced to one block. For \( K = 2^n \), the parent matrix in each step is of even size and after \( n \) steps the size of the matrix is reduced to 1. The first reduction step, described above, is obviously applied to every other step since all the intermediate parent matrices are of even size. Fig. 2 illustrates the procedure of permutation-reductions throughout the reduction phase for \( n = 3 \).

In general, the permutation on a parent matrix of size \( 2l \) will be

\[
\begin{pmatrix}
0 & 1 & 2 & \cdots & 2i - 1 & 2i & \cdots & 2j & \cdots & 2l - 1 \\
0 & 1 & 2 & \cdots & l - 1 + i & j & \cdots & 2l - 1
\end{pmatrix}
\]

with \( i = 1, 2, \ldots, l \) and \( j = 0, 1, \ldots, l - 1 \). Repeating such a permutation-reduction step \( n \) times, a finite sequence of matrices is generated

\[
P_0 \to P_1 \to \ldots \to P_i \to P_{i+1} \to \ldots \to P_{n-1} \to P_n.
\]

Like \( P_1 \) in (13), each \( P_i \) will be of the QBD structure, consisting of five basic blocks: \( A_0^{(i)}, A^{(i)}, A_1^{(i)}, D^{(i)} \) and \( U^{(i)} \), except \( P_{n-1} \) and \( P_n \). These blocks at \( i = 0 \) are given by the original \( A_0, A, A_1, D \) and \( U \) in (1). The task at the \( i \)-th reduction step is to update the five basic blocks according to the basic update formulae:

\[
\begin{align*}
A_0^{(i+1)} &= A^{(i)} - D^{(i)} A_0^{(i)} U^{(i)} - U^{(i)} A^{(i)} D^{(i)}, \\
A^{(i+1)} &= A^{(i)} - D^{(i)} A^{(i)} U^{(i)} - U^{(i)} A^{(i)} D^{(i)}, \\
A_1^{(i+1)} &= A_1^{(i)} - D^{(i)} A_1^{(i)} U^{(i)}, \\
U^{(i+1)} &= -U^{(i)} A_1^{(i)} U^{(i)}, \\
D^{(i+1)} &= -D^{(i)} A^{(i)} D^{(i)},
\end{align*}
\]

which is directly extended from (14). The amount of computations in each step is fixed, independent of the size of \( P_i \).
A close observation of (17) indicates that the major portion of the workload at the $i$-th step is made up by the following block operations:
- $A_{0d}^{(i)} = D^{(i)} \cdot A_0^{(i-1)}$; $A_{0i}^{(i)} = U^{(i)}$;
- $A_{1d}^{(i)} = U^{(i)} \cdot A_1^{(i-1)}$; $A_{1i}^{(i)} = D^{(i)} \cdot A_1^{(i-1)}$;
- $A_{2d}^{(i)} = U^{(i)} \cdot A_2^{(i-1)}$; $A_{2i}^{(i)} = D^{(i)} \cdot A_2^{(i-1)}$.

All the remaining operations are inexpensive matrix additions.

The last two steps are simpler than a typical step described above. For $P_{n-2}$ of size 4, the recursion for $A'(-1)$ is required:

$$A_1^{(n)} = A_1^{(n-1)} - D^{(n-1)} A_0^{(n-1)^{-1}} U^{(n-1)}.$$  

Hence, we get

$$P_{n-1} = \begin{bmatrix} A_0^{(n-1)} & U^{(n-1)} \\ D^{(n-1)} & A_1^{(n-1)} \end{bmatrix}$$ and $P_n = \begin{bmatrix} A_1^{(n)} \end{bmatrix}$.

$P_n$ is the finally reduced 1-block MC matrix. The solution $\pi^{(n)}$, $\pi^{(n)} P_n = 0$, is called the boundary vector. It can be solved directly by many effective methods available [14]-[16]. The criteria for a particular method should stress accuracy rather than mere speed. This is because the accuracy of the boundary vector critically affects that of the entire solution, while the computational efficiency of this step is of far less concern.

2) Backward Expansion Phase: As shown in Fig. 2, $\pi^{(n)}$ corresponds to the last level of the state space $E$. Starting from this boundary vector, the expansion phase will evaluate each solution vector $\pi^{(i)}$, step by step, in reverse order:

$$\pi^{(n)} \to \pi^{(n-1)} \to \ldots \to \pi^{(i+1)} \to \pi^{(i)} \to \pi^{(i+1)} \to \ldots \to \pi^{(1)} \to \pi^{(0)}.$$  

According to the MC reduction principle in Subsection II.B, we must have

$$\pi^{(n)} \in \pi^{(n-1)} \in \ldots \in \pi^{(i+1)} \in \pi^{(i)} \in \ldots \in \pi^{(1)} \in \pi^{(0)}.$$  

Since $P_i$ has half the size of $P_{i-1}$, $\pi^{(i)}$ provides one half of the solution in $\pi^{(i-1)}$. Recall that $P_{i-1}$ is shuffled to $\tilde{P}_{i-1}$ in the reduction phase. Let $\tilde{\pi}^{(i-1)}$, which is the shuffled $\pi^{(i-1)}$, represent the solution vector of $\tilde{P}_{i-1}$. Denoting $\tilde{\pi}^{(i-1)} = [\pi_{2i}^{(i-1)}, \pi_{1i}^{(i-1)}]$, from (7) and (13) we have

$$\tilde{\pi}^{(i)} = -\pi^{(i)} H^{(i)}$$  

with

$$H^{(i)} = \begin{bmatrix} D^{(i-1)} A_0^{(i-1)^{-1}} U^{(i-1)} A^{(i-1)^{-1}} \\ D^{(i-1)} A^{(i-1)^{-1}} & \ldots & \ldots & \ldots \\ \ldots & U^{(i-1)} A^{(i-1)^{-1}} \\ D^{(i-1)} A^{(i-1)^{-1}} \end{bmatrix}.$$  

The original order of $\pi^{(i-1)}$ is restored by re-shuffling $\tilde{\pi}^{(i-1)}$ in the inverse form of (11). Note that the above expansion step requires only three distinct blocks of intermediate products that are obtained and stored during the reduction phase.

The expansion phase operates recursively in the order of $i = n, n-1, \ldots, 1$, each step doubling the size of $\pi^{(i)}$. The finally expanded solution vector $\pi^{(0)}$ will be the full solution vector $\pi$ for $P$. We will provide more details later on the implementation of the expansion phase.

As illustrated in Fig. 2 and described above, each reduction step "folds" away the even-numbered levels so that the "work front" is halved in size. Hence the name the Folding Algorithm. The assumption $K = 2^n$ is only for a clear presentation of the main ideas. As is to be shown later, it will be easily removed so that $K$ can be any valid integer. Moreover, the uniform diagonal assumption for $P$ will be relaxed with level-dependent transitions.

D. Factor Form Representation of $\pi$

The expansion phase can be represented as a sequence of matrix multiplications. Recall the Markov reduction principle in Subsection II.B, and the notations thereof. Note that the $\pi$-vectors used here generally have not been normalized. Without loss of generality, the following also assumes $K = 2^n$ for some integer $n$.

Using the same subscripts to represent a similar partition of an intermediate $P^{(i)}$, it is straightforward from (8) in Subsection II.B that

$$\pi^{(i)} = \pi^{(i+1)} \cdot [-P_{ts}^{(i)} P_{s}^{(i)}] \cdot S^{(i)} ; \ i = n-1, \ldots, 0,$$  

where

$$P_{ts}^{(i)} P_{s}^{(i)} = \begin{bmatrix} D^{(i)} A_0^{(i-1)^{-1}} U^{(i)} A^{(i-1)^{-1}} \\ D^{(i)} A^{(i-1)^{-1}} & \ldots & \ldots & \ldots \\ \ldots & U^{(i)} A^{(i-1)^{-1}} \\ D^{(i)} A^{(i-1)^{-1}} \end{bmatrix},$$  

which is obtained from (18). $S^{(i)}$ is the matrix representation of the permutation defined in (15), such that $\tilde{P}_{i}^{(i)} = S^{(i)} P^{(i)} S^{(i)T}$. Note that $S^{(i)}$ is a unitary matrix, where $S^{(i)}$ has precisely a single identity block in each row and each column and the rest are 0's. It is used here implicitly to restore the proper order in $\pi^{(i)}$.

Define

$$E^{(i)} = [-P_{ts}^{(i)} P_{s}^{(i)}] \cdot S^{(i)}.$$  

From (21) one can write (19) as

$$\pi^{(i)} = \pi^{(i+1)} \cdot E^{(i)} ; \ i = n-1, \ldots, 0,$$  

which immediately provides a compact expression for the global solution vector $\pi$, as the following theorem asserts. Note that $\pi^{(0)} = \pi$, $\pi^{(n)} = \pi_{K-1}$, as in (3).

**Theorem 1** Let $P$ be a QBD-type stochastic matrix, as defined in (1), with the number of levels $K = 2^n$. The
solution to the equation
\[ \pi \cdot P = [\pi_0, \pi_1, \ldots, \pi_{K-1}] \cdot P = 0 \]
is given by
\[ \pi = \pi_{K-1} \cdot E^{(n-1)} \cdots E^{(1)} \cdot E^{(0)}, \]
where \( \pi_{K-1} \cdot P_n = 0, \) and \( E^{(i)}, i = n-1, \ldots, 0, \) are constructed according to (21), (20), and the update formulae (17).

This representation of \( \pi, \) together with the update formulae in (17) for constructing the \( E^{(i)} \)'s, summarizes the entire algorithm. An \( E^{(i)} \) in (22) is of size \( (2^{n-i-1} \times 2^{i}), \) doubling the size of the \( \pi^{(i+1)} \)-vector at the \( i \)-th step. The theorem can be easily generalized to allow \( K \) to be of arbitrary size, without destroying the factor form of (23). Details will be presented later.

The form of (23) suggests a space-efficient implementation. To obtain the entire solution \( \pi \) explicitly, all the \( E^{(i)} \)'s would be required in the order \( i = n-1, \ldots, 0. \) Since they become available in the order \( i = 0, 1, \ldots, n-1, \) all of them need to be stored in memory. Note that direct multiplications of the non-square \( E \) matrices are extremely costly and must be avoided. Most often, however, an explicit solution of \( \pi \) is unnecessary. In some queueing applications, for instance, the main interest is in the loss probability. To obtain the loss measure only requires the blocking boundary vector \( \pi_{K-1} = \pi^{(n)} \) and the normalizing constant \( B, \) which can be recursively computed through
\[ B = \pi \cdot e = \pi^{(n)} \cdot E^{(n-1)} \cdots E^{(1)} \cdot E^{(0)} \cdot e \]

Proceeding from the right, the above computation of \( B \) can be carried out along the reduction steps. At each step only a matrix-vector multiplication is required. When the reduction phase terminates and the boundary vector is obtained, \( B \) will also be available by a simple inner product operation. In other words, one can compute \( B \) without explicit knowledge of \( \pi. \) In the course of computing \( B, \) \( E^{(i)} \) is used as soon as it becomes available, and can be discarded after the inner product is performed. As a result, we are able to save a substantial amount of memory otherwise needed to store the blocks in \( E^{(i)}, i = 1, 2, \ldots, n-1, \) for the computation of the total vector \( \pi. \)

Other measures of performance can be found in the same way. The computed quantities will be divided by \( B \) to get the proper scale. Since this is a trivial operation, it is equivalent to assuming in the following that \( \pi \) has been normalized. In queueing applications, each level corresponds to a queue length while the phases on each level reflect both input and service dynamics. The \( n \)-th moment of the queue length is thus obtained, with the vector \( e \) replaced by \( f, \) as follows:
\[ E[q^n] = \pi \cdot f = \pi \cdot [0^m e, 1^m e, \ldots, (K-1)^m e]^T. \]

When more than one quantity of interest is desired, the \( f \) vector above can be extended to a matrix form. For example, the marginal density of queue length in steady state is given by
\[ [\pi(0), \pi(1), \ldots, \pi(K-1)] = \pi \cdot F = \pi \cdot \begin{bmatrix} e & e & \cdots & e \end{bmatrix}, \]

where \( \pi(j) = \pi_j \cdot e = Pr(q = j), \) and \( F \) contains \( K \) column vectors.

In general, a steady-state performance measure is defined to be \( E[f(x(t), q(t))] \) of some function \( f \) on the system state space. In vector form it can be expressed as an inner product:
\[ E[f(x(t), q(t))] = \pi \cdot f, \]
where \( f \) is the vector form of \( f, \) with respect to the indexing of the entire state space of the model. Therefore the computation of \( \pi \cdot f \) can be carried out along the reduction steps and no \( E^{(i)} \)'s need to be stored in memory.

E. Extension: Arbitrary Buffer Size
In our modeling, the size of \( P \) is \( K \) in blocks, representing a system with buffer size \( (K-1). \) So far we have assumed \( K \) to be some integral power of 2. This condition can be easily removed by slight modifications to the basic procedure.

An arbitrary positive integer \( K \) can always be uniquely factored as:
\[ K = 2^n \cdot (2m + 1), \quad m, n = 0, 1, 2, \ldots \]

If \( n > 0, \) the first \( n \) steps of reduction are carried out exactly as described before, since the intermediate reduced matrices, \( P_0 \) through \( P_{n-1}, \) are all of even sizes with the original QBD structure. At the beginning of the \( n \)-th step, \( P_n \) is of odd size \( 2m+1, \) but still has the original structure. Without loss of generality, therefore, assume \( P_0 \) to be of size \( K = 2m + 1, m = 1, 2, \ldots \)

Identify the states by levels \( 0, 1, 2, 3, \ldots, 2m - 1, 2m. \) Split the space \( E \) as \( E = S \cup T \) where \( T \) contains all the odd levels plus level \( 2m \) and \( S \) contains the rest. Such a split results in \( P_0 \) being of size \( m, \) and \( P_1 \) of \( m + 1. \) As one will see, this asymmetry causes the resulting \( P_t^* \) to form an additional triad \( (A_b, D_b, U_b) \) in its lower boundary. As an initialization, we relabel \( P_0 \) to reflect this structural change:
It is important to know that the change of structure in this step is only symbolic: we only relabel the four blocks in the lower right hand corner of the matrix; their contents have not yet been changed. Only after the first reduction step will there be a true structural change. We call this new structure the augmented form of the original structure; from here on it is this form which will propagate throughout the rest of the reduction phase.

Next, rearrange the odd-sized $P_0$ according to the permutation:

\[
\begin{pmatrix}
0 & 1 & 2 & \cdots & 2i-1 & 2j & \cdots & 2m-1 & 2m \\
0 & m & 1 & \cdots & m-1+i & j & \cdots & 2m-1 & 2m
\end{pmatrix}
\]

(26)

Note that the last item is unaffected by the permutation, while the permutation of the rest is identical to that as if $K = 2m$ described in (15). It suggests that the rearrangement of $P_0$ can be achieved this way: first mask its last row and last column, then shuffle, just as before, the remaining $(2m \times 2m)$ principal submatrix. The following shows the result of such a permutation-partition:

\[
\tilde{P}_0 = \begin{bmatrix}
A_0 & U & D & U & 0 & 0 \\
D & U & A & D & U & 0 \\
D & U & 0 & A & 0 & D \\
0 & \cdots & \cdots & 0 & A_1 & U_b \\
& & & & A_1 & U_b \\
& & & & D_b & A_1
\end{bmatrix}
\]

which has the same structure of

\[
\tilde{P}_0 = \begin{bmatrix}
P_i & P_{st} \\
P_{st} & \tilde{P}_t
\end{bmatrix}
\]

The 0's in the last column of $P_{st}$ and the last row of $P_{st}$ imply that the last row and last column of $P_{st}(-P_{st}^{-1})P_{st}$ are both null. According to (5), the same triad $(A_b, D_b, U_b)$ will then remain unchanged in $P_1$ in this step. Thus the reduction can be viewed as being performed on the $(2m \times 2m)$ principal submatrix, which is of the shuffled original form, so the actual operations are identical to those for the case when $K = 2m$. In short, the entire reduction step is accomplished by performing the basic reduction only on the $(2m \times 2m)$ principal submatrix of $P_0$. Since the masked blocks are not updated, $P_1$ inherits the augmented form from the odd-sized $P_0$. In general, $P_i$ inherits the augmented form from an odd-sized $P_{i-1}$:

\[
P_i = \begin{bmatrix}
A_b^{(i)} & U^{(i)} & D^{(i)} & U^{(i)} \\
D^{(i)} & A^{(i)} & U^{(i)} \\
D^{(i)} & A_b^{(i)} & U_b^{(i)} \\
D^{(i)} & A^{(i)} & U^{(i)} \\
D^{(i)} & A_b^{(i)} & U_b^{(i)}
\end{bmatrix}
\]

(27)

with

\[
A_b^{(i)} = A_b^{(i-1)}, \quad D_b^{(i)} = D_b^{(i-1)}, \quad U_b^{(i)} = U_b^{(i-1)}.
\]

The other five basic blocks in (27) are updated by the basic update formulae of (17). If $P_i$ is still odd-sized, repeat the above reduction step until the reduced matrix becomes even-sized. In each such step, the blocks of the augmented $P_i$ are updated based on (17) and (28).

Without loss of generality, now assume $P_i$ is of the augmented form with an even size. This step begins by shuffling and partitioning $P_i$ exactly as we did for matrices of original structure and even size. The difference is in that the block $A_b^{(i)}$ now appears in the $P_s$ position of $\tilde{P}_i$, as partitioned below. Hence this reduction step requires additional boundary block updates. The following describes the partition of the shuffled form and the result of the reduction:

\[
\tilde{P}_i = \begin{bmatrix}
\begin{bmatrix}
A_b^{(i)} \\
A_b^{(i)} \\
D_b^{(i)} \\
D_b^{(i)}
\end{bmatrix} & \begin{bmatrix}
U^{(i)} \\
U^{(i)} \\
U_b^{(i)} \\
U_b^{(i)}
\end{bmatrix} \\
\begin{bmatrix}
D^{(i)} \\
D^{(i)} \\
A^{(i)} \\
A^{(i)}
\end{bmatrix} & \begin{bmatrix}
A^{(i)} \\
A^{(i)} \\
D^{(i)} \\
D^{(i)}
\end{bmatrix}
\end{bmatrix}
\]

which can also be denoted by

\[
\tilde{P}_i = \begin{bmatrix}
P_i \\
P_{st} \\
\tilde{P}_t
\end{bmatrix}
\]

Taking the one step reduction of $P_i$ then leads to

\[
P_{i+1} = \begin{bmatrix}
A_b^{(i+1)} & U^{(i+1)} & D^{(i+1)} & U^{(i+1)} \\
D^{(i+1)} & A^{(i+1)} & U^{(i+1)} \\
\cdots & \cdots & \cdots & \cdots \\
D_b^{(i+1)} & A^{(i+1)} & U_b^{(i+1)} \\
D^{(i+1)} & A^{(i+1)} & U_b^{(i+1)}
\end{bmatrix}
\]

Thus, for an even-sized $P_i$, the four boundary blocks at the far-end are updated by

\[
A_b^{(i+1)} = A_b^{(i)} - D_b^{(i)}A_b^{(i-1)}U_b^{(i)} - U_b^{(i)}A_b^{(i)-1}D^{(i)},
\]

\[
A_b^{(i+1)} = A_b^{(i)} - D_b^{(i)}A_b^{(i-1)}U_b^{(i)},
\]

\[
D_b^{(i+1)} = -D_b^{(i)}A_b^{(i)-1}D^{(i)},
\]

\[
U_b^{(i+1)} = -U_b^{(i)}A_b^{(i)-1}U_b^{(i)}.
\]

The remaining four basic blocks are obtained by the basic update formulae of (17). Accordingly, the following additional block operations are required in order to carry out all the updates:

- $A^{(i)-1}$ in LU factored form;
- $A_1^{(i)} = D_b^{(i)}A_1^{(i)-1}$ and $A_{1u}^{(i)} = U_b^{(i)}A_1^{(i)-1}$;
- $A_{1d}^{(i)}, A_1^{(i)}U_b^{(i)}, A_{1d}^{(i)}D_b^{(i)}, A_1^{(i)}D^{(i)}, A_{1u}^{(i)}U_{b}^{(i)}$. 

The backward expansion process differs very little in the general case. According to (7), all we need to know in carrying out this phase is to remember the left half of each intermediate matrix: the $P_i$'s and the $P_{is}$'s. In our case they are highly structured so that we need only to keep the information of their constructions and to store their basic building blocks.

The modified algorithm is now fully capable of handling $P$ of arbitrary size $K$. This comes at a slight price: it has to process the "corroded" lower boundary of the intermediate $P_i$'s as a result of the inevitable skewness in the partition-reduction process.

One may view the entire reduction phase as a single macro step of partition-reduction on $P$:

$$
P = \begin{bmatrix}
A_0 & U \\
D & A & U \\
& D & A & U \\
& & D & A_1
\end{bmatrix} \rightarrow [A_1'] .
$$

(30)

A brief review of the update operations will indicate

$$A_1' = A_1 + G ,$$

where $G$ is an accumulation of all the updates throughout the reduction phase. Based on (5), $G$ must be independent of $A_1$. Obviously, the same reduction can be applied to a principal submatrix of $P$ as well, where all the block operations producing $G$ will be performed just the same way, regardless of $A_1$. Likewise, if the solution vector corresponding to $A_1$'s level is known, the full solution can be recovered by a single expansion phase without involving $A_1$. This view of the reduction and expansion is the main vehicle for the next extension.

F. Extension: Level-Dependent Transitions

In queueing analysis, we are especially interested in system behavior under different overload control policies. Such controls can be applied to both arrival process and service discipline. The parameters of such control policies are adjusted with the queue contents. It is reflected in the model's matrix through the modifications of the triad $(A,D,U)$. As a result, the transition matrix of the system model will be in piece-wise uniform block tri-diagonal form. An example of such a matrix with one level-control is given here:

$$
\hat{P} = \begin{bmatrix}
A_0 & U \\
D & A & U \\
& D & A & U \\
& & D & A_1
\end{bmatrix}
$$

which can also be expressed by

$$\hat{P} = \begin{bmatrix}
P_s & \dots & \dots & \dots \\
& P_{is} & \dots & \dots \\
& & \dots & \dots \\
& & & P_1
\end{bmatrix} ,$$

where $c$ is the control level and the triad $(\hat{A}, \hat{D}, \hat{U})$ describes the controlled transitions.

Consider reducing $\hat{P}$ as suggested by the above partition. Because both $P_{is}$ and $P_s$ contain only one non-zero block respectively, the updating term $P_{is}P_{is}^{-1}P_{st}$ in (5) will only have one non-zero block, whose position exactly coincides with that of the first $A$ in $P_t$. As a result, $P_t^*$ is identical to $P_t$, except on the first $A$'s position of $P_t$ which is to be updated by the reduction. That is,

$$\hat{P} \rightarrow P_t^* = \begin{bmatrix}
\hat{A}_0 & \hat{U} \\
\hat{D} & \hat{A} & \hat{U} \\
& \hat{D} & \hat{A} & \hat{U} \\
& & \hat{D} & \hat{A}_1
\end{bmatrix}$$

According to (5), we get $\hat{A}_0$, the only updated element in the reduction, equal to

$$\hat{A}_0 = \hat{A} - [0, \ldots, 0, \hat{D}]P_{is}^{-1}[0, \ldots, 0, U]^T .$$

As indicated by (30), this is equivalent to applying one phase of reduction to the following matrix:

$$P_s^* \overset{\text{def}}{=} \begin{bmatrix}
A_0 & U \\
D & A & U \\
& D & A & U \\
& & D & A
\end{bmatrix} \rightarrow [\hat{A}_0] ,$$

Likewise, once $\hat{A}_0$ is obtained, one can further reduce $P_t^*$ to the final block, denoted by $[\hat{B}_0]$, in the next reduction phase:

$$P_t^* = \begin{bmatrix}
\hat{A}_0 & \hat{U} \\
\hat{D} & \hat{A} & \hat{U} \\
& \hat{D} & \hat{A} & \hat{U} \\
& & \hat{D} & \hat{A}_1
\end{bmatrix} \rightarrow [\hat{B}_0] .$$

Hence, the whole reduction process is accomplished in two phases. The boundary vector is the solution of $\pi^{(n)}[\hat{B}_0] = 0$.

Similarly, two expansion phases are required to construct the full solution vector from the boundary vector $\pi_{K-1} = \pi^{(n)}$. In the first expansion phase, based on Theorem 1 one can expand the boundary vector $\pi_{K-1}$ to the first solution vector $[\pi_c, \pi_{c+1}, \ldots, \pi_{K-1}]$, given by

$$[\pi_c, \pi_{c+1}, \ldots, \pi_{K-1}]P_t^* = 0 .$$

Since the first block of $P_t^*$ and the last block of $P_t^*$, $\hat{A}_0$ and $\hat{A}$, essentially have the same position in $\hat{P}$, the first
element in the solution vector of $P^*_i$, i.e., $\pi_c$, will then be used as a boundary vector in the second expansion phase to construct the second solution vector $[\pi_0, \pi_1, \ldots, \pi_d]$, given by

$$[\pi_0, \pi_1, \ldots, \pi_d]P^*_i = 0.$$  

The full solution vector $\pi$ is therefore obtained.

In the same manner, we can deal with matrices with multiple segments of uniformity. If a model has $k$ segments of local uniformity as shown above, the entire solution process at the phase level resembles that of the basic uniform case at the step level, consisting of the reduction cycle and the expansion cycle:

- **R-cycle**: \( R_1, R_2, \ldots, R_k \)
- **E-cycle**: \( E_k, E_{k-1}, \ldots, E_1 \)

With this generalization, we see that a queue with finite buffer but without control is a special case of the general control problem. It also allows us to study some complex and large-scale problems arising from queueing applications. For instance, $k$ may represent the number of levels of overload controls implemented in a queueing system. Such a scheme can fine-tune the system to provide guaranteed quality services to different users.

### III. Complexity and Error

We here examine the algorithm in three aspects: time and space complexity and error performance.

#### A. Time and Space Complexity

It is obvious that in this folding algorithm the reduction phase presents the major demands on computation time and space. Assume \( K = 2^n \), for convenience. Since each reduction step reduces the state space of the intermediate MC by half, there are \( \log_2 K \) steps in this phase.

The time complexity is measured in FLOP unit: one FLOP equals one ADD plus one MULTIPLY in floating point. According to (17), each reduction step (except the first and last ones) requires two matrix inversions and eight multiplications, plus some negligible matrix additions. Since the inversions are not explicitly needed, the products like $A^{-1}U$ can be more economically obtained by solving the equation $AX = U$. Thus $A^{(i)}_0$ and $A^{(i)}_1$ in (17) are LU-decomposed, each requiring $N^3/3$ FLOPs. Each block multiplication requires $N^3$ FLOPs. The overall time complexity in the reduction phase is therefore given by

- **Basic R-phase**
  - Time complexity \( \approx 8\frac{2}{3} \cdot N^3 \cdot \log_2 K \) FLOPs.

The reduction phase requires two types of memory space: **on-line space** and **off-line space**. The on-line space, usually the random-access memory, is used to hold the five basic building blocks and three intermediate blocks. The off-line space, usually the disk storage, is used to save $E^{(i)}$ as in (21), or the four basic building blocks, $\{A^{(i)}_0, A^{(i)}_1, D^{(i)}, U^{(i)}\}$, obtained in each reduction step. According to (20), only three blocks need to be stored at each step to reconstruct the intermediate solution vectors $\pi^{(i)}$, $i = 1, 2, \ldots, n - 2$.

Note that $P_0$ and $P_{n-1}$ need not be stored: the latter is used immediately, and the former can be easily regenerated from given system parameters. Using FPN (one floating point number) as the basic unit for space, we have

- **Basic R-phase**
  - Space Complexity \( \approx 3(\log_2 K - 2) N^2 \) off-line FPNs.

The time complexity for the expansion phase is given by

- **E-phase**
  - Time Complexity \( \approx 2KN^2 \) FLOPs;

since computing each $\pi_i$, according to (18), involves at most two vector-matrix multiplications. The on-line space requirement is

- **E-phase**
  - Space Complexity \( \approx 3N^2 + KN \) on-line FPNs,

in which $3N^2$ is for the three blocks of intermediate products used in (20) at each expansion step and $KN$ is for the solution vector $\pi^{(i)}$. Clearly, the reduction is expected to dominate the overall time and space requirements, except when $N$ is small and $K$ large.

For arbitrary values of $K$, some reduction steps may require up to 75% more time and space to update the additional four boundary blocks in (29). For example, as found in a case study, the total elapsed time is increased by 56% as $K$ changes from $2^{10}$ to $2^{10} - 1$ (the worst case). In summary, the entire process for solution of $\pi P = 0$ is typically dominated by the reduction phase both in time and space. The overall solution requires

- **Total Time** \( \approx 8(\frac{2}{3}) \cdot N^3 \cdot \log_2 K + 2KN^2 \) FLOPs,

- **Total Space** \( \approx \begin{cases} \max((8 + \delta) \cdot 3N^2, 3N^2 + KN) & \text{on-line FPNs,} \\ \alpha \cdot 3(\log_2 K - 2)N^2 & \text{off-line FPNs,} \end{cases} \)

with $1 \leq \alpha < 2$; $\delta = 0$ if $\alpha = 1$ ($K = 2^n$), and 1 otherwise ($K \neq 2^n$). As indicated in Subsection II.D, the off-line space is needed only when the explicit solution of $\pi$ is required. Note that the off-line space requirement gives another measure of time complexity in terms of memory and disk access time, which often occupies a significant portion of the total computation time.

Hence, the time and space complexity for the Folding-algorithm is given by $O(N^3 \log_2 K)$ and $O(N^2 \log_2 K)$, respectively. In contrast, a direct application of the block Gaussian elimination (or, block LU decomposition) will yield a time and space complexity of $O(N^3 K)$ and $O(N^2 K)$ [9, 17]. Perhaps more important than time and space complexity, the Folding-algorithm is much less prone to round-off error than the block Gaussian elimination method. One reason is that the Folding-algorithm only has $\log_2 K$ iterations while the block Gaussian elimination requires $K$ iterations. Each iteration introduces round-off error. The numerical stability of both methods is strongly affected by the round-off error accumulation in a sequence of iterations.
That is why, as we have found, the block Gaussian elimination method can only be applied to a small buffer system. Of course, in the extreme case where the transitions change with each level (which is unlikely in a real system), the two methods becomes algorithmically equivalent.

B. Error Performance

The accuracy of the final solution in the Folding-algorithm relies on two factors: the errors in the underlying block operations, and the stability of the reduction process. First look at the block operations. Clearly all matrix multiplications involve positive numbers only. In each reduction step, \(-A^{-1}\) is strictly positive since it represents the sojourn times. The blocks \(D^{(i)}\) and \(U^{(i)}\) are non-negative, because \(P_t\) in each step is stochastic. In each expansion step the solution vector \(\pi(0)\) is of course positive. This presents the most favorable condition as it minimizes the relative roundoff errors.

The LU decomposition of \(A^{(i)}\) critically affects the overall accuracy. Since all its principal minors are substochastic and thus positive-definite, the stability for LU decomposition of \(A^{(i)}\) is ensured. If we take advantage of the diagonal redundancy in a stochastic matrix, all subtractions can be replaced by additions [14]. This will make the algorithm virtually error-free. Even without such error control means, for all examples reported in this paper and also in [10, 11], the residual error in the final solution \(\|P_t\|_{\infty}\), is found to be negligible (in range of \(10^{-12} \sim 10^{-16}\)).

The overall reduction process is numerically stable, as the elements of \(P_t\) in each step are bounded by the main diagonal of the original \(P\). Since each diagonal element in a transition matrix dominates all the off-diagonals on its row, it suffices to show that, by each reduction, the diagonal of the reduced matrix does not increase in magnitude. This can be observed from the definition of \(P_t^{(i)}\) in (5). Note that both \(P_t\) and \(P_t^{(i)}\) must have negative diagonals. Since the diagonal of \(P_t^{(i)}(-P_t^{(-1)})P_t^{(i)}\) is non-negative, the diagonal of \(P_t^{(i)}\) will not increase in magnitude. One may interpret this from the Markov chain reduction principle. \(P_t^{(i)}\) is an MC reduced from \(P\) with partial observation on subset \(T\). Hence, the transitions from a state \(t\) in \(T\) to \(S\), and then back to the same \(t\), will no longer be visible in \(P_t^{(i)}\). The rate of leaving state \(t\) in \(P_t^{(i)}\) must therefore be less than that in \(P_t\). As a result, the \(t\)th diagonal element of \(P_t^{(i)}\) is reduced in magnitude.

The reduction process can also be viewed as a special form of block LU decomposition, especially for implementation of the algorithm. As such, it is easily shown that the following diagonal dominance condition [9]:

\[ ||A||^{-1}(||D|| + ||U||) < 1 \]

is met by the structural property of the QBD model, ensuring the overall stability of the algorithm.

C. Comparison with Existing Techniques

Various techniques exist for the solution of infinite QBD-process without level-dependent transitions. In particular, Neuts [1] and others have developed the matrix-geometric solution method for a class of Markov models that includes the QBD-process. Hajek [4] extended the Neuts' result to finite QBD-process as defined in (1). This method mainly involves the determination of a matrix-geometric factor \(R\) by iteratively solving the quadratic equation:

\[ R = -R^2DA^{-1} - U A^{-1}. \]  

(31)

The above iteration converges very slowly [18, 19] for typical parameter values. In [18] an acceleration scheme was proposed for a special case: \(R^2 = -RA - U\). Despite its marked improvement over the Neuts' original method, it still requires a considerable amount of time.

Consider a typical QBD-process that models the queue of voice and video integrations on a transmission link [10, 11, 20, 21]. Fig. 3 shows both the number of iterations and computation time required to find \(R\) using the above two iteration schemes, as a function of system utilization \(\rho\). For simplicity, the matrix \(R\) in (31) is chosen to be of size \(100 \times 100\). The number of iterations, denoted by \(I(\rho)\), is defined with convergence tolerance \(10^{-7}\). As one can see, \(I(\rho)\) is prohibitively large using the iteration scheme in (31), hence the matrix-geometric method can easily become infeasible even for moderate phase size. For the accelerated iteration scheme, which happens to hold here, \(I(\rho)\) is still large and rising rapidly with the system utilization \(\rho\). It is interesting to observe that in this example \(I(\rho)\) closely resembles the function \(100\rho/(1 - \rho)\). Of course, the additional amount of computation time to find the entire solution vector \(\pi\) from \(R\) is \(O(KN^2)\) in FLOPS.

When the matrix-geometric solution is applied to a finite model in general, the simple geometric form is lost. Instead the solution is in the dual geometric form [4]. The computation time required is more than \(2I(\rho)\). Both \(R\) and its dual \(\bar{R}\) are required. To find the solution \(\pi\), a substantial amount of computations is further required. Details can be found in [4].

Folding-algorithm, on the other hand, is a direct solution method. Its complexity depends only on \(K\) and \(N\).
As shown in Fig. 3, the upper bound computation time for the entire solution of π is less than 200 seconds, for \( K \leq 1024 \). Taking advantage of the logarithmic complexity due to \( K \), one can use Folding-algorithm to approximate the solution of an infinite buffer, by letting \( K \) be sufficiently large. Fig. 3 also plots the computation time for Folding-algorithm to obtain an approximate solution for the infinite buffer case, where \( K \) is so chosen that \( \Pr(q = K - 1) \sim 10^{-6} \). The figure clearly shows that Folding-algorithm is significantly faster in the practical range of parameter values, especially under heavy load conditions.

Besides mere computational speed, Folding-algorithm is a more powerful computational method than the existing techniques. Its real strengths are in the capabilities of handling the finite process with level-dependent transitions. To the best of our knowledge, no technique is available to-date for solution of finite QBD-process with level-dependent transitions.

Folding-algorithm is suitable for queueing analysis of finite buffer system with overload control, because of its versatility and efficiency. It is geared to the more realistic finite models, whose effects on delay and loss performance are difficult to gauge on the infinite buffer models. It can also be used to find approximate solutions to infinite QBD-process if necessary, without much increase in complexity. In any case, \( \log_2 K \) is unlikely to exceed 20, if \( K \) is the buffer size. The algorithm, therefore, can be considered of order \( O(N^3) \). This complexity is significantly less than that of most existing methods which rely on both infinite buffer and no overload control assumptions.

IV. Application to Statistical Multiplexing Queueing Analysis

Consider a statistical multiplexer, concentrating traffic from multimedia sources. The system has a buffer of finite size \( K - 1 \). The service rate is \( C \). The overall arrival rate of the traffic at time \( t \), denoted by \( a(t) \), is contributed by \( M \) independent traffic streams: \( a(t) = \sum_{i=1}^{M} a_i(t) \) where \( a_i(t) \) represents the arrival rate of the \( i \)-th traffic stream at time \( t \). Let \( a_i(t) \) be modulated by an independent Markov chain \( x_i(t) \) with \( x_i(t) \in \{1, 2, ..., N_i\} \). That is, \( a_i(t) \) is a function of the state of the Markov modulator \( x_i(t) : a_i(t) = \lambda_{x_i(t)} \), \( a_i(t) \) assumes \( N_i \) values (not necessarily distinct), according to the state of \( x_i(t) \). This traffic model has been widely used to characterize individual traffic streams, such as voice and video [22]-[24]. The overall arrival rate process \( a(t) \) is modulated by an \( M \)-dimensional modulator: \( x(t) = [x_1(t), x_2(t), ..., x_M(t)] \in X \). The Markov-modulated \( a(t) \) is expressed by

\[
a(t) = \sum_{i=1}^{M} a_i(t) = \sum_{i=1}^{M} \lambda_{x_i(t)}.
\]

Denote \( Q_i \) as the transition rate matrix of \( x_i(t) \). The transition rate matrix of \( x(t) \) is given by

\[
Q = Q_1 \oplus Q_2 \oplus ... \oplus Q_M
\]

where the notation \( \oplus \) represents the Kronecker sum [25]. \( Q \) is an \( N \times N \) matrix with \( N = \prod_{i=1}^{M} N_i \), which tends to get very large in multimedia traffic applications. The queueing process is represented by \( q(t) \in \{0, 1, ..., K - 1\} \). Depending on how \( a(t) \) drives \( q(t) \), two QBD models can be constructed. The first one is to extend the finite \( M/M/1 \) model by allowing the arrival rate \( A \) to be replaced by \( a(t) \), and \( C \) is the exponential service rate. The overall arrival process then becomes a superposed Markov-modulated Poisson process (MMPP). It is referred to as Markov-modulated exponential queueing model (MM/EX). The second queueing model is a Markovian version of the continuous fluid-flow model [26]-[29]. Ignoring the local dynamics of individual arrivals and departures, the queue changes at the exponential rate \( a(t) - C \), unless the queue becomes full or empty. Unlike the continuous fluid-flow model, however, \( q(t) \) assumes a discrete state space. It is therefore referred to as Markov-modulated fluid-flow queueing model (MM/FF). Fig. 4 shows the state transition diagram of this model.

Both Markov-modulated Poisson arrivals and Markov-modulated fluid-flow arrivals are commonly used to represent multimedia traffic in queueing analyses. The difference between the two arrival processes has negligible impact on queueing performance, when the time varying scale of \( a(t) \) is sufficiently greater than the average service time which is true in typical voice and video applications [11, 24].

Both MM/EX and MM/FF models are described by a finite QBD-process, where each level represents a buffer occupancy \( q(t) \) and each phase on each level corresponds to a state of \( x(t) \). At each transition, the queue \( q(t) \) is permitted to change at most by one. Without overload control, \( x(t) \) is independent of \( q(t) \), hence the phase transitions on each level are identical except at the two boundary levels 0 and \( K - 1 \). The blocks \( A \), \( A_0 \) and \( A_1 \) describe the phase transitions, whose off-diagonal elements are identical to those of \( Q \); the blocks \( U \) and \( D \) specify the level transitions, representing one-step changes in buffer occupancy. Note that the blocks can be level-dependent under overload control. In our modeling, both \( U \) and \( D \) are diagonal matrices. For the MM/EX model, \( U \) represents the arrival

Fig. 4. Transition diagram of MM/FF model
rate matrix equal to \( \text{diag}[\lambda(x(t))] \), which is an \((N \times N)\) diagonal matrix with its \(x(t)\)-th coordinate element on the diagonal equal to \( \lambda(x(t)) \); \( D \) is the service rate matrix given by \( CI \) where \( I \) is an identity matrix. For the MM/FF model, \( U \) represents the queue increasing rate matrix equal to \( \text{diag}[(\lambda(x(t)) - C)^+] \) and \( D \) is the queue decreasing rate matrix given by \( \text{diag}[(C - \lambda(x(t)))^+] \), where the symbol \((\cdot)^+\) denotes the larger of 0 or its argument.

The solution of \( \pi P = 0 \) provides the microscopic state equilibrium distribution \( \pi(j;k) \in \pi \), where \( j \) is the queue length with \( 0 \leq j < K \) and \( k \in X \) is the state of the traffic modulator. The marginal queue length distribution is given by \( \pi(j) = \sum_{k \in X} \pi(j;k) \). The mean and variance of the queue are given by \( E(q) = \sum_j j \pi(j) \) and \( \sigma_q^2 = \sum_j (j - E(q))^2 \pi(j) \).

Define \( P_E \) as the probability of an empty system, \( R_B \) the total packet loss rate, and \( \rho \) the normalized link offered load. Also, let \( \rho_i \) and \( R_{Bi} \) be the corresponding measures for the \( i \)-th type traffic. While \( x(t) = k = [k_1, \cdots, k_M] \), the overall arrival rate is equal to \( \lambda_k = \sum_i \lambda_{ki} \), where \( \lambda_{ki} \) represents the arrival rate of the \( i \)-th type traffic. For the MM/EX model, we have

\[
P_E = \pi(0),
\]

\[
R_{Bi} = \rho_i^{-1} \sum_{k \in X} \lambda_{ki} - \rho_k \cdot \pi(K - 1; k).
\]  

where \( \rho_k = \lambda_k/C \) is the normalized link input rate, conditional on \( x(t) = k \). We assume packets are blocked regardless of traffic types.

For the MM/FF model to be an irreducible chain, there must exist a nontrivial partition on \( x(t) \) state space: \( X = X_u \cup X_o \), where \( X_u = \{k|\lambda_k \geq C\} \) and \( X_o = \{k|\lambda_k < C\} \). For the queue only increases when \( x(t) \in X_o \), and only decreases when \( x(t) \in X_u \). Based on the fluid-flow assumption, while \( q = 0 \) the server will not be idle unless \( x(t) = k \in X_o \). Similarly, while \( q = K - 1 \) no blocking will occur unless \( k \in X_u \). The following is readily obtained:

\[
P_E = \sum_{k \in X_u} (1 - \rho_k) \pi(0; k),
\]

\[
R_{Bi} = \rho_i^{-1} \sum_{k \in X_u} \lambda_{ki} - \rho_k \cdot (1 - \rho_k) \cdot \pi(K - 1; k).
\]

To derive \( P_E \), note that \((1 - \rho_k)\) is the idle probability conditioned on \( k \in X_u \). To find \( R_{Bi} \), note that \( \rho_k \) gives the normalized link input rate conditioned on \( k \in X_o \).

For both models, the total overflow loss rate is given by \( R_B = \rho^{-1} \sum_i \rho_i R_{Bi} \). Based on the total probability law, the consistency of a model can be checked by the identity

\[
P_E = 1 - \rho(1 - R_B),
\]

where \( P_E \) is obtained from \( \pi(0; k) \) for \( k \in X_u \), and \( R_B \) is from \( \pi(K - 1; k) \) for \( k \in X_o \). Hence, \( (38) \) relates two quantities that are computationally far apart. The effect of the total roundoff errors from these two distant sets can then be checked in \( (38) \). All our numerical results agree with \( (38) \) to machine precision (i.e., in the order of \( 10^{-16} \)).

In numerical study, we consider the queueing performance of voice and video integration. As in [19, 20, 24], each stream is assumed to be modeled by the superposition of \( N_i \) i.i.d. 2-state Markovian source elements for \( i = 1, 2 \). Each element, alternating between ON and OFF periods, is characterized by

\[
\begin{align*}
\text{OFF: } & \begin{bmatrix} -\alpha_i & \alpha_i \\ \beta_i & -\beta_i \end{bmatrix} \\
\text{ON: } & \begin{cases} 
\alpha_i & \text{if } n = m - 1 \geq 0 \\
\alpha_i - (N_i - m) \beta_i & \text{if } m = n \\
(N_i - m) \beta_i & \text{if } m = n + 1 \leq N_i \\
0 & \text{else}
\end{cases}
\end{align*}
\]

where \( 0 \leq m, n \leq N_i \). The modulator of the aggregate input traffic is \( Q = Q_1 \oplus Q_2 \).

For a single source element, define \( \lambda_i \) as the access bit rate while in ON state, \( T_{on} \), the average ON time, and \( T_{off} \), the average OFF time. While \( x(t) = [x_1(t), x_2(t)] = [k_1, k_2] \), we get \( \lambda_k = k_1 \lambda_1 + k_2 \lambda_2 \). Let \( K \) be the buffer size in packets, and \( l \) be the packet size in bits. The packet transmission interval is given by \( \Delta = l/C \). Accordingly, both \( \alpha_i \) and \( \beta_i \) are normalized in \( \Delta \) units: \( \alpha_i = \Delta/T_{off} \) and \( \beta_i = \Delta/T_{on} \). The matrix \( P \) can therefore be constructed.

In the example, voice sources [21] are described by \( N_1 = 30, \lambda_1 = 64 \text{ Kbps}, T_{on} = 400 \text{ ms}, T_{off} = 600 \text{ ms} \); video sources by \( N_2 = 20, \lambda_2 = 1.14 \text{ Mbps}, T_{on} = 386 \text{ ms}, T_{off} = 765 \text{ ms} \) where 10 source elements characterize one video [20]. Choose the buffer size \( K = 511 \) with packet length \( l = 512 \) bits. Performance based on both MM/EX and MM/FF models is compared for their applicability. Plotted in Fig. 5 are the queue length distributions, indexed by \( \rho \). The solution of \( \{ E(q), \sigma_q, R_b \} \) in function of \( \rho \) is also plotted in Fig. 6. As above mentioned, the difference between the two models is negligible for voice and video queueing analysis.

One may compare the Folding-algorithm to the fluid-flow approach in [26]-[29]. The fluid-flow approach, like the generating function approach in [23, 24], requires the arrival Markov chain \( Q \) to be decomposable into a set of independent MC elements, where the eigenvalues of each element must be given in explicit analytical form. Typically, the size of each MC element has to be no more than four states, which substantially limits the range of arrival MC's for source modeling [31]-[35]. By contraries, the Folding-algorithm only requires the QBD structure, and therefore can handle any type of Markovian source models. Note that we use the two-state Markovian source models here simply because they are available. It is shown in [10] that the results based on both MM/FF and fluid-flow approach are almost identical, given that the problem is also solvable by the fluid-flow approach.

Relying on the assumption of \( N \) i.i.d. 2-state MC sources and an infinite buffer, the fluid-flow approach gives a closed
form solution [26]. In this case, Table 1 compares the Folding-algorithm with the fluid-flow approach in terms of CPU time for $N = 50, 100, 200, 250$ at $\rho = 0.8, 0.9$. Note that we here have chosen the buffer size as large as $K = 1023$ and 2047 for the Folding-algorithm. The smaller the $K$, the less the CPU time for the Folding-algorithm. As one can see, the fluid-flow approach, even in this 'most favorite' case, does not exhibit any advantage in computation time as compared to the Folding-algorithm, especially when $N$ is large. This is because the Folding algorithm can benefit more from the sparse matrix techniques, while the fluid-flow approach requires a substantial amount of combinatorial calculations to find the eigenvectors. In comparison, the Folding-algorithm is a much more general method. Its real strengths are in the capabilities of handling the finite and controlled cases, especially when the modulator matrix $Q$ assumes an arbitrary structure. Its time complexity is independent on the structure of $Q$. Further, both $D$ and $U$ blocks don't have to be diagonal matrix for the Folding-algorithm. Note that a much general class of queueing systems can be modeled by QBD process.

Let us now consider an example of ATM link with integration of preemptive emulated circuit-switched (ECS) services and packet-switched (PS) services, which cannot be directly solved by the fluid-flow approach. In ATM network, the ECS services are guaranteed to be lossless and non-delayed, but without a dedicated circuit for each end-to-end connection. Here we investigate the effects of ECS services on the performance of PS services.

Consider an ATM-type multiplexer carrying both real-time PS and ECS services simultaneously. Assume that each ECS channel has an access rate of $\lambda_c$. The multiplexer allows at most $N_c$ number of ECS connections, where $N_c \leq C$. The admitted ECS traffic is given preemptive priority in accessing bandwidth. Their packets can be transmitted immediately upon arrival. The circuit-switched qualities are thus ensured for the ECS services. The PS services are allowed to use all the remaining capacity.

The analysis of performance for each type of service can be conducted separately. The performance of the ECS services is measured by the blocking probability $P_b$ for an incoming ECS call request. Assume each ECS call arrives and departs according to Poisson rates $\alpha$ and $\beta$, respectively. $P_b$ is then determined from $\alpha, \beta$ and $N_c$ via Erlang's formula.

The delay and loss behavior of the PS services are described by a finite queue. It accepts only the PS traffic and has a server with time-varying service rate. Because of the preemptive priority given to the ECS packets, the effective service rate for the PS traffic is now state-dependent. Let $x(t)$ be the number of ECS channels occupied at time $t$, and $C_p(t)$ be the effective service rate for the PS services. From the above server allocation scheme, we have $C_p(t) = C - x(t)$ clearly, $x(t)$ is a birth-death Markov process with state space $\{0, 1, \ldots, N\}$. By analogy, $C_p(t)$ is called the Markov-modulated service rate, or MMSR. Therefore, both arrival and service rates of the queue for PS traffic are Markov-modulated. The overall phase process is $\pi(t) = [x_1(t), x_2(t)]$, whose Markov generator $Q$ is given by $Q = Q_1 \oplus Q_2$. $Q_1$ is the generator for $x_1(t)$, describing the dynamics of the aggregate PS traffic; $Q_2$ is the

<table>
<thead>
<tr>
<th>Number of Voice Sources</th>
<th>$\rho = 0.8$</th>
<th>$\rho = 0.9$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FOLD</td>
<td>FLUID</td>
</tr>
<tr>
<td>50</td>
<td>9.7</td>
<td>4.6</td>
</tr>
<tr>
<td>100</td>
<td>38</td>
<td>32</td>
</tr>
<tr>
<td>200</td>
<td>220</td>
<td>243</td>
</tr>
<tr>
<td>250</td>
<td>445</td>
<td>470</td>
</tr>
</tbody>
</table>

FOLD: folding algorithm with MM/FF model, $\rho = 0.8$, Buf=1023; $\rho = 0.9$, Buf=2047.

FLUID: fluid-flow approach in closed-form.

CPU times are measured on SUN SPARC station 1

Table 1. Comparison of computation time with the closed-form solution of fluid flow model in [25].
generator for $x(t)$, the number of active ECS connections. Given $Q$, $K$, and the assumption of MM/EX or MM/FF, the matrix $P$ can be constructed for numerical analysis, as shown before. From the above modeling of ECS traffic, we get for $Q_e$

$$[Q_e]_{m,n} = \begin{cases} \frac{m\beta}{\alpha} & \text{if } n = m - 1 \geq 0 \\ -\frac{\alpha - m\beta}{\alpha} & \text{if } m = n \\ \frac{\alpha}{\alpha} & \text{if } n = m + 1 \leq N_e \\ 0 & \text{else} \end{cases}$$ (41)

It is obvious that $Q_e$ cannot be further decomposed, to which the fluid-flow approach fails to apply.

In numerical study, consider a statistical multiplexor concentrating voice packets from both PS and ECS services. The source access rate for both types is set at $\lambda = 64$ Kbps. The buffer size is $K = 190$. For the ECS traffic, we have $N_e = 20$, $\Delta / \beta = 100$ sec, for average call holding time and $\Delta / \alpha = 0.12$ sec for average call interarrival time. These three parameters, which are used to construct $Q_e$ in (41), gives the ECS call blocking probability $P_b = 0.01$. For the PS traffic, $N_1 = 30$, $T_{on} = 0.4$ sec, and $T_{off} = 0.6$ sec, which are used to construct $Q_1$ in (40). The link utilization factor $\rho$ is chosen to be 0.7, 0.8, 0.9, and 0.95, respectively. The MM/FF model is used here.

Because of the fluctuated service rate for PS traffic, its delay and loss performance can only be improved through adaptive overload controls. The control can be based on both $z(t)$ and $q(t)$. Its objective is to prevent the queue from further increase once the buffer size exceeds a given threshold $L < K$. For example, the improvement can be achieved by reducing the excessive PS arrival rate $[a_1(t) - C_p(t)]^+$ by $\theta$ portion whenever $q(t) \geq L$ and $a_1(t) > C_p(t)$. We have $[a_1(t) - C_p(t)]^+ = [(k_1 + k_e)\lambda - C]^+$ while $[z_1(t), x_1(t)] = [k_1, k_e]$.

For multi-level control, whenever $L_{i+1} > q(t) \geq L_i$ suppose that one can discard the excessive PS arrivals up to $\theta_i$ fraction of the total PS arrivals, given $\theta_i \in (0, 1)$ for $i = 1, 2, \ldots, J$. The $i$-th level control, designed with $(L_i, \theta_i)$, is then measured by the average PS discarding rate:

$$R_{C_i} = \min \left\{ \theta_i, \frac{\max [(k_1 + k_e)\lambda - C, 0]}{k_1 \lambda} \right\}$$ (42)

while $L_{i+1} > q(t) \geq L_i$ with $L_{i+1} = K$. The overall PS discarding rate is equal to $R_{C} = \sum_i R_{C_i}$. In our case, the 2-level control parameters are given by $(L_1, \theta_1) = (63, 0.25)$ and $(L_2, \theta_2) = (126, 0.75)$.

We consider three versions of the above system: non-controlled (NC); one-level-controlled (C1), using $(L_1, \theta_1)$; and two-level-controlled (C2), using both $(L_1, \theta_1)$ and $(L_2, \theta_2)$. All the queue length distribution functions are plotted in Fig. 7, and the rest of the performance data are listed in Table 2. The control objective is two-fold: to reduce blocking probability $R_b$, and to lower the overdue loss rate by shortening the average queue length $E[q]$.

The Folding-algorithm has been extensively used in [11] to explore many issues in multimedia traffic communication systems. They include packet loss, overload control, dynamic resource allocation and approximation techniques. A variety of examples of large scale systems are given in [11] to demonstrate the algorithm’s viability for performance analysis in this area.

V. CONCLUSION

This paper introduces a new computational method for steady state analysis of finite QBD process with level dependent transitions. Instead of formulating solutions in matrix-geometric form, the Folding-algorithm provides a technique for direct computation of $\pi P = 0$. Taking a finite sequence of fixed-cost binary reduction steps, the original $P$ is reduced to a single-level matrix, from which a boundary vector is obtained. Each step halves the matrix size but keeps the QBD form. The solution $\pi$ is expressed as a product of the boundary vector and a finite sequence of expansion factors. The time and space complexity to solve $\pi P = 0$ is therefore reduced from $O(N^3K)$ and $O(N^3K)$
The Folding-algorithm has a number of highly desirable advantages when it is applied to solving queueing problems. In particular, the algorithm handles the multi-level control problem in finite buffer system. Its total independence of the phase structure allows the algorithm to apply to more elaborate, multiple-state Markovian sources. Its computational efficiency, numerical stability and superior error performance are also distinctive advantages. Implemented on a workstation, the algorithm has handled problems with \( N \) over 1,400, \( K \) up to 16,384, and combined \( NK \) often exceeds \( 10^6 \), without any noticeable roundoff errors. Because of these qualities, the Folding-algorithm is indeed an efficient solution tool and is generally applicable for statistic multiplexing queueing analysis. The Folding-algorithm has recently been extended to the transient analysis of finite QBD processes [36].

REFERENCES

[17] H. Yamada and F. Machihara, "Performance analysis of a statistical multiplexer with control on input and/or service pro-

Jingdong Ye (S'85-M'91) received the B.S. degree from South China Institute of Technology, Guangzhou, China, in 1982, and M.S. and Ph.D. degrees from Columbia University in 1984 and 1991 respectively. All the degrees are in electrical engineering.

Since 1991, he has been a member of the engineering staff at Hughes LAN Systems, Mountain View, CA, a subsidiary of Hughes Aircraft Company. He is a key contributor in the company's efforts to develop the first generation ATM products that will facilitate the deployment of ATM technologies to enterprise networks.

Dr. Ye's interests include network performance analysis, numerical solution methods of Markov models, multimedia services in enterprise networks. He is particularly interested in the signaling, traffic management, and routing aspects of local area ATM networks.

San-qi Li received the B. S. degree from the Beijing University of Posts and Telecommunications, China, in 1975, and the M. A. Sc. and Ph. D. degrees from the University of Waterloo, Canada, in 1982 and 1985, respectively, all in electrical engineering. From 1976 to 1980, he worked in the Posts and Telecommunications Academy of China as a Research supervisor. From 1985 to 1989, he was an Associate Research Scientist and Principal Investigator of Center for Telecommunications Research at Columbia University where he was actively involved in conducting research, supervising doctoral students, and collaborating with faculty members. In September 1989, he joined the faculty of the Department of Electrical and Computer Engineering, University of Texas at Austin, where he is presently Assistant Professor. He published more than 70 technical papers in computer communication network research field. His current research interests are multimedia traffic analysis and control of high speed networks.

Dr. Li has served as a member of the Technical Program Committee for the IEEE INFOCOM Conference since 1988.