Maximally Flexible Assignment of Orthogonal Variable Spreading Factor Codes for Multi-Rate Traffic

Yang Yang and Tak-Shing Peter Yum

Department of Information Engineering
The Chinese University of Hong Kong, Shatin, Hong Kong

Tel: 852-2609 8346, 2609 8386 Fax: 852-2603 5032
Email: yyang@ie.cuhk.edu.hk, yum@ie.cuhk.edu.hk

Abstract

In UTRA systems, Orthogonal Variable Spreading Factor (OVSF) codes are used to support different transmission rates for different users. In this paper, we first define the *flexibility index* to measure the capability of an assignable code set in supporting multi-rate traffic classes. Based on this index, two single-code assignment schemes, *non-rearrangeable* and *rearrangeable* compact assignments, are proposed. Both schemes can offer maximal flexibility for the resulting code tree after each code assignment. We then present an analytical model and derive the call blocking probability, system throughput and fairness index. Analytical and simulation results show that the proposed schemes are efficient, stable and fair.

Index Terms — Code assignment, Orthogonal Variable Spreading Factor (OVSF) code, Universal Terrestrial Radio Access (UTRA).

1 Introduction

The third-generation mobile communication system has been under active research and development in the past decade. The most important issue to decide on is, of course, the air-interface. After much effort by the various technical groups at ITU, a family of air-interface standards are agreed upon. The Universal Terrestrial Radio Access (UTRA) is mainly a joint European-Japanese contribution. UTRA consists of two parts, the FDD

*This work was supported in part by the Hong Kong Research Grants Council under Grant CUHK 4325/02E.

part (choosing WCDMA as air interface) is for wide-area coverage and paired spectrum allocation; the TDD part (choosing TD-CDMA as air interface) is for local-area coverage and unpaired spectrum allocation.

In UTRA, a traffic channel is identified by an Orthogonal Variable Spreading Factor (OVSF) code and OVSF codes can support multi-rate transmissions for different users [1, 2]. According to $3GPP$ technical specifications [1, 2], multi-code transmission and single-code transmission are both possible to support multi-rate multimedia applications. Two multi-code assignment schemes were proposed in [3] and [4] respectively. In general, however, single-code transmission is preferred due to lower transceiver complexity [5]. We focus on single-code assignment schemes in this paper.

There are two types of code assignment schemes: *non-rearrangeable* and *rearrangeable*. In [6], several code assignment schemes were proposed. The static (non-rearrangeable) approach applies the first-fit scheme (for the bin packing problem) in the algorithm design. The dynamic approach is based on a tree partitioning method, which requires the knowledge of traffic composition (percentage of different data-rate users). Two priority-based rearrangeable code assignment schemes were proposed in [7] and [8], respectively, to accommodate both the real time traffic (circuit-switched, e.g. voice communication and video streaming) and the non-real time traffic (best-effort, e.g. file transfer and email). Obviously, real time traffic has a higher priority to obtain a code. Specifically, the scheme in [8] makes use of the bursty property of real time traffic and can therefore offer higher system utilization. The code assignment scheme suggested in [7] performs code reassignment for real time traffic classes on every call departure instant. At these instants, the "right-most" call in the same layer (of the code tree) is moved to occupy the "just-released" code. As a result, the remaining assignable single-code capacity is maximized. Subsequently, Chen, Wu and Hsiao [9] extended this scheme by partitioning the codes into two groups based on code capacity (the bandwidth that a code can support). When a code is released, the "right-most" or the "left-most" (according to the group the code belongs to) call in the same layer will be rearranged to the "just-released" code. After that, the ongoings calls in the lower layers (if any) are rearranged similarly, layer by layer. It is interesting to note that similar packing methods were used in Dynamic Channel Assignment (DCA) schemes for TDMA/FDMA systems [10, 11]. Furthermore, the Region Division Assignment (RDA) scheme presented in [12] divides the code tree into multiple mutually exclusive regions with each region dedicates to a particular transmission data rate. When a new call cannot be accommodated in the corresponding region, a suitable code in other regions is borrowed and assigned to the new call. This is equivalent to the concept of channel borrowing in literature [13, 14]. In [15], Minn and Siu proposed a rearrangeable assignment scheme whereby the number of OVSF codes that must be rearranged to support a new call is minimized. According to the authors, the main challenge of using this scheme lies in the searching effort of the "minimum-cost" branch [1].

In this paper, the *flexibility index* is defined to measure the capability of an assignable

[1] In [15], the "cost of a branch" is defined as "the minimum number of code rearrangements necessary to reassign all occupied codes in the branch to other branches so that the branch is left empty".

code set in supporting multi-rate traffic. Based on this new concept, two computational efficient single-code assignment schemes, namely *Compact Assignment* (CA) and *Rearrangeable Compact Assignment* (RCA), are proposed and analyzed. Both schemes leave the system as flexible as possible after each code assignment.

The rest of this paper is organized as follows. In Section 2, the OVSF codes are represented by a tree, and some basic concepts are introduced. In Sections 3 and 4, the maximally flexible *non-rearrangeable* and *rearrangeable* assignment schemes are proposed. The Markov chain models for studying the performance of CA and RCA are given in Section 5. Based on these models, the call blocking probability, system throughput and fairness index are derived for RCA under the Poisson arrival of calls and exponential call holding time assumptions. In Section 6, numerical and simulation results are given. In Section 7, performance and implementation complexity are discussed and compared between different schemes.

2 Basic Concepts

2.1 Ancestor Code Set $S_A^{(k,m)}$ and Descendant Code Set $S_D^{(k,m)}$

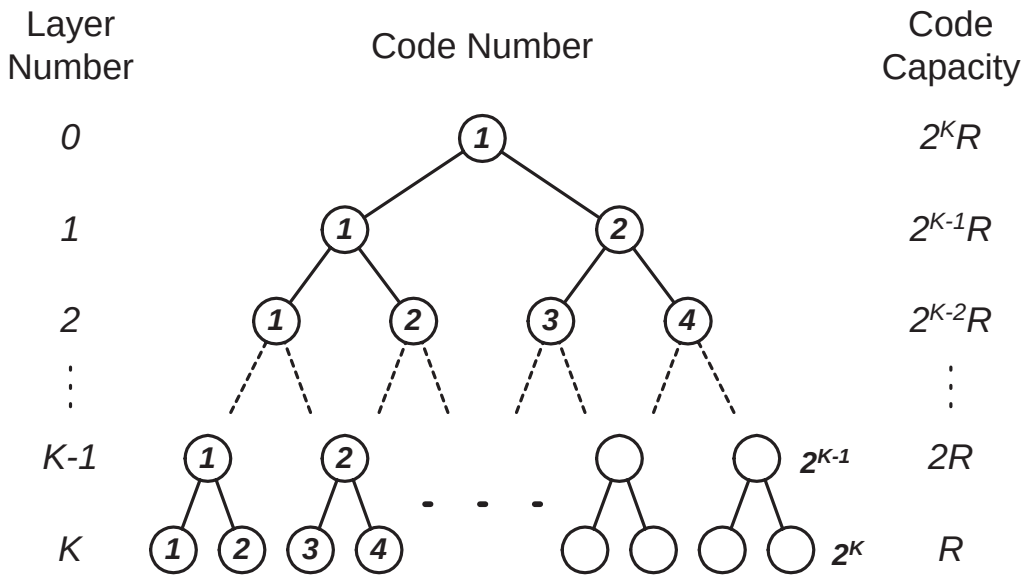

Figure 1: A K-layer Code Tree

The OVSF codes can be represented by a tree [16]. Fig. 1 shows a K-layer code tree[2]. Each layer corresponds to a particular spreading factor, so all codes in the same layer can support the same data rate. The data rate a code can support is called its capacity. Let

[2]In some other notational convention, this is referred to as a $(K+1)$-layer tree.

the capacity of the leaf codes (in layer K) be R. Then, the capacity of the codes in layer $(K-1)$, $(K-2)$, ..., 1 and 0 are $2R$, $4R$, ..., $2^{K-1}R$ and $2^K R$ respectively, as shown in Fig. 1.

Layer k has 2^k codes and they are sequentially labeled from left to right, starting from one. The m^{th} code in layer k is referred to as code (k, m). The total capacity of all the codes in each layer is $2^K R$, irrespective of the layer number.

For a typical code (k, m) $(k \geq 1)$, its ancestor codes are the codes on the path from (k, m) to the root code $(0, 1)$. Therefore, the set of ancestor codes of (k, m), denoted by $S_A^{(k,m)}$, is given by

$$S_A^{(k,m)} = \left\{ (p, q) \,|\, 0 \leq p \leq k-1,\, q = \left\lceil \frac{m}{2^{k-p}} \right\rceil \right\}, \quad k \geq 1 \tag{1}$$

where $\lceil x \rceil$ is the ceiling function. On the other hand, the descendant codes of (k, m) $(k \leq K-1)$ are the codes in the branch under (k, m). The set of these descendant codes $S_D^{(k,m)}$ is given by

$$S_D^{(k,m)} = \left\{ (p, q) \,|\, k+1 \leq p \leq K,\, (m-1) \cdot 2^{p-k} + 1 \leq q \leq m \cdot 2^{p-k} \right\}, \, k \leq K-1 \tag{2}$$

2.2 Busy Code Set S_B and Assignable Code Set S

When a code is assigned to a call, we say that the code is *busy*. They are denoted by black circles in Fig. 2. Codes that are not assigned to calls are called *idle* codes. Idle codes can be *assignable* or *non-assignable*. An idle code (k, m) is *assignable* if and only if

Condition I: all the ancestor codes of (k, m) *in* $S_A^{(k,m)}$ *are idle, and*

Condition II: all the descendant codes of (k, m) *in* $S_D^{(k,m)}$ *are idle.*

Code (k, m) is *non-assignable* otherwise. These two conditions guarantee that the assignable code under consideration is orthogonal to all the busy codes in the tree [1, 2, 16]. Further, based on the framework given in [16], two propositions can be directly derived as follows.

Proposition 1 *The ancestor and descendant codes of a busy code are non-assignable idle codes.*

Proposition 2 *If code* (k, m)*, where* $k \leq K-1$*, is assignable, so are all its descendant codes.*

Proposition 1 can be illustrated by the busy code $(2, 1)$ in Fig. 2 (a). According to the definitions given in (1) and (2), the ancestor and descendant code sets of $(2, 1)$ can be calculated to be $S_A^{(2,1)} = \{(0, 1), (1, 1)\}$ and $S_D^{(2,1)} = \{(3, 1), (3, 2)\}$. Since $S_A^{(2,1)} \cap S_B = \phi$ (empty set), all the codes in $S_A^{(2,1)}$ are idle. On the other hand, $S_A^{(2,1)} \cap S = \phi$ means all the codes in $S_A^{(2,1)}$ are also non-assignable. Similar argument holds for $S_D^{(2,1)}$, thereby

all the codes in $S_D^{(2,1)}$ are non-assignable idle codes, too. Proposition 2 can be illustrated by code $(2,2)$ in Fig. 2 (a) and its descendant code set $S_D^{(2,2)} = \{(3,3),(3,4)\}$. In other words, $(2,2) \in S$ implies $S_D^{(2,2)} = \{(3,3),(3,4)\} \subset S$.

The occupancy status of a K-layer code tree can be uniquely specified by the set of busy codes S_B. Based on Proposition 1, the set of assignable codes S can be derived from S_B by the following algorithm [3].

S_B to S Transformation Algorithm

INPUT: the busy code set S_B and the tree size K.
OUTPUT: the assignable code set S.

1. Generate $S = \{$*all codes in the K-layer code tree*$\}$.
2. WHILE S_B is not empty, repeat the following:
   2.1 Arbitrarily select a code, say (k,m), from S_B.
   2.2 Generate $S_A^{(k,m)}$ by (1).
   2.3 Generate $S_D^{(k,m)}$ by (2).
   2.4 Update $S = S - S_A^{(k,m)} - S_D^{(k,m)} - \{(k,m)\}$.
   2.5 Update $S_B = S_B - \{(k,m)\}$.
3. Return S.

To illustrate, the occupancy status of the 3-layer code tree shown in Fig. 2 (a) can be specified by the busy code set $S_B = \{(2,1),(3,6),(3,8)\}$. The set of assignable codes is found to be $S = \{(2,2),(3,3),(3,4),(3,5),(3,7)\}$.

2.3 Assignable Capacity r and Flexibility Index f

In our study, J classes of calls are defined where a class-j $(0 \le j \le J-1)$ call has data rate 2^j (in unit of R). A class-j call can be supported by an assignable code in layer $(K-j)$ under the single-code assignment schemes.

The assignable leaf codes in layer K are *inflexible* since they can only support unit data rate R. The assignable codes from layer $(K-1)$ upwards are *flexible* in supporting multiple data rates since their descendant codes are also assignable (Proposition 2). As an example, code $(3,3)$ in Fig. 2 (a) is inflexible, whereas code $(2,2)$ is flexible since it can support a class-1 call or two class-0 calls by its descendant codes $(3,3)$ and $(3,4)$.

The assignable capacity r of a code tree can be calculated by adding the capacity of all inflexible assignable codes in layer K. In unit of R,

$$r = \sum_{m=1}^{2^K} I_A^{(K,m)} \tag{3}$$

[3]The mapping from S_B to S is a surjection but not a bijection.

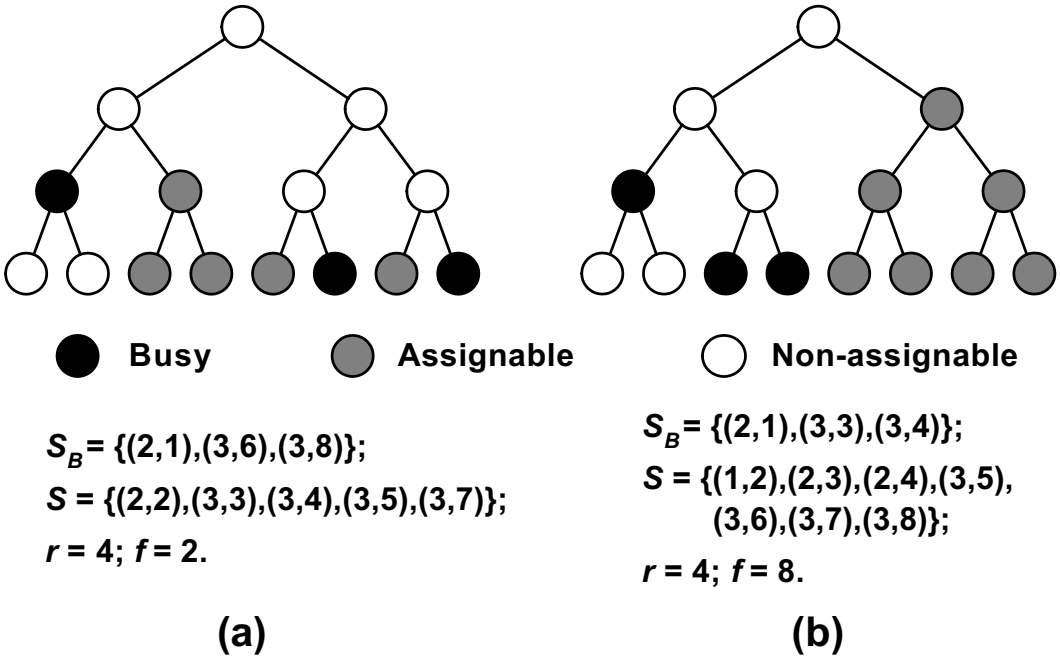

Figure 2: Occupancy Statuses of Two 3-layer Code Trees

where $I_A^{(K,m)}$ is the assignability index function of code (K, m) and is defined as

$$I_A^{(k,m)} = \begin{cases} 1, & (k,m) \textit{ is assignable}; \\ 0, & \textit{otherwise}. \end{cases} \tag{4}$$

For the code tree shown in Fig. 2 (a), $r = 4$.

To measure the capability of a code tree in supporting different data rates, we define *flexibility index* f as the total capacity (in unit of R) of the flexible assignable codes. Specifically,

$$f = \sum_{k=0}^{K-1} \sum_{m=1}^{2^k} 2^{K-k} \cdot I_A^{(k,m)} \tag{5}$$

For example, the flexibility indices for the trees in Fig. 2 (a) and Fig. 2 (b) are computed to be $f = 2$ (the capacity of code $(2, 2)$) and $f = 8$ (the total capacity of codes $(1, 2)$, $(2, 3)$ and $(2, 4)$), respectively.

Proposition 3 *For a K-layer code tree with assignable capacity r ($r \geq 1$), flexibility index f is bounded by*

$$0 \leq f \leq \sum_{i=1}^{\lfloor \log_2 r \rfloor} \left\lfloor \frac{r}{2^i} \right\rfloor \cdot 2^i = f_{max}, \quad r \geq 1 \tag{6}$$

where $\lfloor x \rfloor$ is the floor function.

Proof of Proposition 3: First, since $I_A^{(k,m)}$ is nonnegative, so is f according to (5). Second, the maximum value of f occurs when all assignable codes are located on the same idle branch as much as possible. Adding up the capacity of all the assignable codes from layer $(K-1)$ upwards, we obtain the upper bound f_{max}. ■

When $f = f_{max}$, the code tree is said to be in the *compact* state. In compact state, the capacities of assignable codes are aggregated and the tree is maximally flexible in supporting different data rates. As an example, the code tree in Fig. 2 (a) has $f = 2$ under $r = 4$. It can only support a new call of rate R or $2R$. While in Fig. 2 (b), the code tree has $f = f_{max} = 8$ under the same assignable capacity. Therefore, it is in the compact state and can support a new call of rate R, $2R$ or $4R$.

3 Compact Assignment

The objective of *Compact Assignment* (CA) scheme is to keep the remaining assignable codes in the most compact state after each code assignment without rearranging codes, i.e. to maximize tree's flexibility index. To achieve this purpose, new-code assignments in CA are packed as tightly as possible into the existing busy codes, i.e. the assignable code in the most congested position is found for the new call. As a result, the busy codes are also kept as compact as possible after each code assignment.

3.1 Compact Index $g^{(k,m)}$

To find the assignable code in the most congested position, the *compact index* $g^{(k,m)}$ of an assignable code (k, m) is defined to represent the positional relationship between (k, m) and all the other assignable codes in layer k. Codes (in the same layer) that are connected by an i-layer sub-tree are defined as the i^{th}-layer neighbours. Let $S_i^{(k,m)}$ denote the set of i^{th}-layer neighbours of code (k, m). Then,

$$S_i^{(k,m)} = \left\{(k, m-p+q) \,|\, p = (m-1) \ mod\ 2^i,\ 0 \le q \le 2^i - 1\right\} \tag{7}$$

Take code $(3, 3)$ in Fig. 2 (a) as an example, the sets of 1^{st}- and 2^{nd}-layer neighbours are $S_1^{(3,3)} = \{(3, 3), (3, 4)\}$ and $S_2^{(3,3)} = \{(3, 1), (3, 2), (3, 3), (3, 4)\}$, respectively. The compact index $g^{(k,m)}$ of code (k, m) is the total number of assignable codes in its k different neighbourhoods, or

$$g^{(k,m)} = \sum_{i=1}^{k} |S_i^{(k,m)} \cap S| \tag{8}$$

where $|x|$ denotes the size of set x. For example, in Fig. 2 (a), $g^{(2,2)} = 1 + 1 = 2$ and $g^{(3,3)} = 2 + 2 + 4 = 8$. Note that in (8), the assignable codes located close to (k, m) are counted multiple times in different neighbourhoods. The closer these codes are located to (k, m), the more times they are counted. This is because a closer code is more "compact"

and therefore should carry a higher weight in the computation of compact index. As a result, a small $g^{(k,m)}$ implies that (i) code (k,m) is surrounded by a small number of other assignable codes; and/or (ii) these codes are far away from (k,m).

Proposition 4 *For assignable code (k,m), the range of compact index $g^{(k,m)}$ is given by*

$$k \leq g^{(k,m)} \leq 2^{k+1} - 2. \tag{9}$$

Proof of Proposition 4: First, the minimum value of $g^{(k,m)}$ occurs when code (k,m) is the only assignable code in layer k. In this case, $|S_i^{(k,m)} \cap S| = 1$ for $1 \leq i \leq k$. $g^{(k,m)}$ is then equal to $\sum_{i=1}^{k} 1 = k$. Second, $g^{(k,m)}$ is maximum when the whole tree is empty, i.e. the assignable capacity r is equal to 2^K. In this case, $|S_i^{(k,m)} \cap S| = |S_i^{(k,m)}| = 2^i$ and the maximum value is $\sum_{i=1}^{k} 2^i = 2^{k+1} - 2$. ■

3.2 CA Algorithm

According to the definition of $g^{(k,m)}$, an assignable code with the smallest compact index in layer $(K-j)$ is chosen by CA for carrying a class-j new call. The newly assigned code is marked busy and set S_B is then updated. The detailed algorithm of CA is as follows.

CA Algorithm

INPUT: the busy code set S_B and the layer number $(K-j)$.
OUTPUT: the busy code set S_B, it will be updated after a successful code assignment.

Phase I: Search for assignable code in the most congested position.

1. Generate S by using the "S_B to S Transformation Algorithm".
2. Compute $k^{min} = Min\{\, k \,|\, (k,m) \in S\}$.
3. IF $k^{min} \leq K-j$, THEN do the following:
   3.1 Generate $S_C = \{(K-j,m) \,|\, (K-j,m) \in S\}$.
   3.2 IF S_C is not empty, THEN do the following:
      3.2.1 Compute $g^{(K-j,m)}$ for each code in S_C by (8).
      3.2.2 Compute $g^{min} = Min\{\, g^{(K-j,m)} \,|\, (K-j,m) \in S_C\}$.
      3.2.3 Update $S_C = \{(K-j,m) \,|\, g^{(K-j,m)} = g^{min}\}$.
   ELSE do the following: *($k^{min} > K-j$, all the individual assignable codes in S do not have sufficient capacity for the new call.)*
   3.3 Block the new call.
   3.4 Return S_B. *(S_B is NOT updated.)*

Phase II: Assign a code from S_C to the new call.

4. Arbitrarily select a code, say $(K-j,m')$, from S_C.
5. Assign $(K-j,m')$ to the new call.
6. Return $S_B = S_B \cup \{(K-j,m')\}$.

In the algorithm, k^{min} is the smallest layer number in the assignable code set S and S_C is the set of candidate codes in layer $(K-j)$. According to Proposition 2, $k^{min} \leq K-j$

ensures that S_C is not empty. To illustrate the CA Algorithm, let us assume a class-0 ($j = 0$) call arrives and the existing busy code set is $S_B = \{(2,1),(3,6),(3,8)\}$ as shown in Fig. 2 (a). $S = \{(2,2),(3,3),(3,4),(3,5),(3,7)\}$ will first be generated and k^{min} is then computed to be $2 < K - j = 3$. Next, the candidate code set S_C is generated as $\{(3,3),(3,4),(3,5),(3,7)\}$. Compact indices for the codes in S_C are computed one by one to be $\{g^{(3,3)} = 8, g^{(3,4)} = 8, g^{(3,5)} = 7, g^{(3,7)} = 7\}$. Therefore, we obtain $g^{min} = 7$ and S_C is updated to be $\{(3,5),(3,7)\}$. In *Phase II*, codes $(3,5)$ and $(3,7)$ are selected with the same probability for the new call.

4 Rearrangeable Compact Assignment

Although CA is very simple, it has the drawback that, sometimes, even if there is enough assignable capacity, a new call will still be blocked if individual assignable codes all have smaller capacity than the required data rate. These blockings are *avoidable* since they can be resolved by rearranging the busy codes. Code rearrangement need only be triggered at call arrival instants when avoidable blockings occur. Therefore, we call this *blocking-triggered* code rearrangement. For example, the assignable capacity of the 3-layer code tree shown in Fig. 2 (a) is $r = 4$, but a class-2 new call will still be blocked by CA since each code in S has a smaller capacity than 4. At this instant, code rearrangement is triggered so that the calls on codes $(3,6)$ and $(3,8)$ are reassigned to codes $(3,3)$ and $(3,4)$ as shown in Fig. 2 (b), and the assignable code set becomes $S = \{(1,2),(2,3),(2,4),(3,5),(3,6),(3,7),(3,8)\}$. Code $(1,2)$ being freed up can then be used to accommodate the class-2 new call.

Codes can be rearranged in different ways. One way is to rearrange all the busy codes and pack them as tightly as possible to one side of the tree. In doing so, the assignable codes are aggregated together and the resulting tree is maximally flexible according to Proposition 3. This method is simple, but it incurrs many unnecessary code rearrangements. Alternatively, we can just rearrange all the busy descendant codes of a particular code, say code (k,m), so that code (k,m) can be released for assignment to the new call. In [15], the total number of code rearrangements is used as a performance index. In addition to that, the code rearrangement efficiency (including branch-searching effort) is also considered for design optimization in this paper.

4.1 Maximally Flexible Code Rearrangement

In this section, we propose a computational efficient blocking-triggered code rearrangement scheme named

Maximally Flexible Code Rearrangement (MFCR). It finds a suitable branch for rearrangement so that the root code of that branch can be released for the new call.

To reduce the branch searching and code rearrangement effort, MFCR chooses the *least loaded* branch for rearrangement. For a typical branch, say the branch under (k,m),

let $r^{(k,m)}$ be the assignable capacity of the branch, which is defined as the total capacity of the assignable leaf codes in this branch. In other words,

$$r^{(k,m)} = \sum_{i=1+(m-1)\cdot 2^{K-k}}^{m\cdot 2^{K-k}} I_A^{(K,i)} \tag{10}$$

Correspondingly, the branch load $l^{(k,m)}$ can be computed by

$$\begin{aligned} l^{(k,m)} &= \left[\begin{array}{l} \textit{Capacity of} \\ \textit{code } (k,m) \end{array} \right] - \left[\begin{array}{l} \textit{Assignable capacity of} \\ \textit{the branch under } (k,m) \end{array} \right] \\ &= 2^{K-k} - r^{(k,m)} \end{aligned} \tag{11}$$

For single-code assignment, the least loaded branch is just the branch with the maximum assignable capacity r^{max}. In other words, (10) and (11) are equivalent for finding the least loaded branch. The detailed operations of identifying the set of least loaded branch(es) is given in *Phase I* of the MFCR Algorithm.

Let $f^{(k,m)}$ denote flexibility index of the branch under (k,m). From (5), we have

$$f^{(k,m)} = \sum_{j=k}^{K-1} \sum_{i=1+(m-1)\cdot 2^{j-k}}^{m\cdot 2^{j-k}} 2^{K-j} \cdot I_A^{(j,i)} \tag{12}$$

Take code $(1,1)$ in Fig. 2 (a) as an example, $f^{(1,1)}$ is equal to 2. When there are several least loaded branches available, MFCR identifies the branch with smallest flexibility index f^{min}, or the *least flexible branch*, for rearrangement. By rearranging the busy codes in the least flexible branch, flexibility index of the whole tree is kept as large as possible after each code assignment. This explains the name *Maximally Flexible Code Rearrangement*. The least flexible branch can be identified by *Phase II* of the MFCR Algorithm.

In *Phase III* of MFCR Algorithm, all the ongoing calls located on the busy codes in the selected branch are reassigned to the assignable codes in neighbouring branches by the RCA algorithm (to be described in the next section). After that, the newly released root code of the selected branch is assigned to the new call.

MFCR Algorithm

INPUT: the busy code set S_B and the layer number $(K-j)$.
OUTPUT: the updated busy code set S_B. *(Code $(K-j,m')$ is released and assigned to the new call.)*

Phase I: Search for the Least Loaded Branch.

1. Generate S by using the "S_B to S Transformation Algorithm".
2. Compute $r^{(K-j,i)}$ for the branch under code $(K-j,i)$ $(1 \le i \le 2^{K-j})$ by (10).
3. Compute $r^{max} = Max\{\, r^{(K-j,i)} \,|\, 1 \le i \le 2^{K-j}\}$.
4. Generate $S_C = \{(K-j,m) \,|\, r^{(K-j,m)} = r^{max}\}$.

Phase II: Search for the Least Flexible Branch.

5. IF $|S_C| \ge 2$, THEN do the following:
   5.1 Compute $f^{(K-j,m)}$ for each code in S_C by (12).
   5.2 Compute $f^{min} = Min\{\, f^{(K-j,m)} \,|\, (K-j,m) \in S_C\}$.
   5.3 Update $S_C = \{(K-j,m) \,|\, r^{(K-j,m)} = r^{max},\ f^{(K-j,m)} = f^{min}\}$.

Phase III: Empty out a branch and assign its root code to the new call.

6. Arbitrarily select a code, say $(K-j,m')$, from S_C.
7. Generate $S_D^{(K-j,m')}$ by (3).
8. Generate $S_B^{(K-j,m')} = S_B \cap S_D^{(K-j,m')}$. *Identify the busy codes in the branch under $(K-j,m')$.)*
9. WHILE $S_B^{(K-j,m')}$ is not empty, repeat the following:
   9.1 Arbitrarily select a code, say (k^*,m^*), from $S_B^{(K-j,m')}$.
   9.2 Rearrange the ongoing call on (k^*,m^*) to a neighbouring branch by using the "RCA Algorithm". S_B is then updated.
   9.3 Update $S_B^{(K-j,m')} = S_B^{(K-j,m')} - \{(k^*,m^*)\}$.
10. Assign $(K-j,m')$ to the new call.
11. Return $S_B = S_B \cup \{(K-j,m')\}$.

4.2 RCA Algorithm

Based on the CA and MFCR algorithms, we design the algorithm for *Rearrangeable Compact Assignment* (RCA) as follows.

RCA Algorithm

INPUT: the busy code set S_B and the layer number $(K-j)$.
OUTPUT: the busy code set S_B, it will be updated after a successful code assignment.

1. Generate S by using the "S_B to S Transformation Algorithm".
2. Compute r by (4).
3. IF $r \geq 2^j$, THEN do the following:
   3.1 Generate $S_C = \{(K-j,m) \,|\, (K-j,m) \in S\}$.
   3.2 IF S_C is not empty, THEN do the following:
      3.2.1 Compute $g^{(K-j,m)}$ for each code in S_C by (8).
      3.2.2 Compute $g^{min} = Min\{\, g^{(K-j,m)} \,|\, (K-j,m) \in S_C\}$.
      3.2.3 Update $S_C = \{(K-j,m) \,|\, g^{(K-j,m)} = g^{min}\}$. *(Search for assignable codes with the smallest compact index.)*
      3.2.4 Arbitrarily select a code, say $(K-j,m')$, from S_C.
      3.2.5 Assign $(K-j,m')$ to the new call.
      3.2.6 Return $S_B = S_B \cup \{(K-j,m')\}$.
   ELSE do the following: *(An empty S_C indicates all the individual assignable codes do not have sufficient capacity for the new call. Therefore, code rearrangement is necessary.)*
      3.2.7 Find, release and assign a suitable code for the new call by using the "MFCR Algorithm". S_B is then updated.
      3.2.8 Return S_B.
   ELSE do the following: *($r < 2^j$, assignable capacity of the code tree is NOT sufficient.)*
   3.4 Block the new call.
   3.5 Return S_B. *(S_B is NOT updated.)*

Again, consider the tree shown in Fig. 2 (a), upon the arrival of a class-2 new call, RCA will first transform the code tree in Fig. 2 (a) to the one shown in Fig. 2 (b) by using MFCR Algorithm. Then, the newly released code $(1,2)$ is assigned to the new call and the busy code set S_B is updated to $\{(1,2),(2,1),(3,3),(3,4)\}$. Note that step 3.2.4 of the RCA Algorithm indicates the major difference between RCA and other algorithms in [7, 9] where the "left-most" or "right-most" code is always chosen when several choices are available. As illustrated by the example shown in Fig. 2 (a), choosing either code $(3,5)$ or $(3,7)$ (the "right-most" one) for a class-0 new call does not make a difference in performance since the adjacent busy codes $(3,6)$ and $(3,8)$ have the same probability to be released. It is the distinction between code sets $\{(3,3),(3,4)\}$ and $\{(3,5),(3,7)\}$ that makes the different. By using code set $\{(3,5),(3,7)\}$ for the class-0 new call, high flexibility index of the resulting code tree is maintained so that a new class-1 call can be supported.

5 Performance Analysis

5.1 Traffic Model

Let there be J ($1 \leq J \leq K+1$) classes of calls where class-j ($0 \leq j \leq J-1$) calls are characterized by

(i) data rate in unit of R equals to 2^j;

(ii) Poisson arrivals with rate λ_j; and

(iii) exponentially distributed call holding time with mean μ_j^{-1}.

Let the class-j offered traffic G_j be defined as $G_j = \lambda_j/\mu_j$ and let $G = \sum_{j=0}^{J-1} G_j$ be the total offered traffic.

5.2 Markov Chain for Compact Assignment

The code assignment and release process can be modeled by a Markov chain. Consider the simple *Random Assignment* (RA) scheme whereby all suitable codes are identified and one is picked at random. Its Markov chain model is shown in Fig. 3 for the case $K = 2$ and $J = 3$. This chain has 26 states and they are denoted by α_i ($0 \leq i \leq 25$) as shown. Let $S(\alpha_i)$ be the set of assignable codes in state α_i. Code assignments and releases are represented by transition between states. As an example, the transition rates to and from α_6 are shown in Fig. 4. For class-0 new calls, the transition rates from α_6 to α_{12}, α_{13} and α_{16} are each $\frac{\lambda_0}{3}$ since all the layer-2 codes in $S(\alpha_6)$, namely $(2,1)$, $(2,2)$ and $(2,4)$, have the same chance of being assigned.

The Markov chain for CA scheme is the same as that for RA scheme in Fig. 3 without the dashed lines and with different transition rates. As compact index is used to choose codes, some transitions are now removed. For example, the new transition rates for α_6 is given in Fig. 5, where the rates from α_6 to α_{12}, α_{13} and α_{16} are now 0, 0 and λ_0, respectively. For a class-0 new call, only code $(2,4)$ is used since it has smaller compact index than codes $(2,1)$ and $(2,2)$.

5.2.1 Blocking Probability

Let π_i be the limiting probability for state α_i. These probabilities can be computed by solving the Markov chain in the usual manner. The blocking probability $P_B(j)$ of class-j calls is then given by

$$P_B(j) = \sum_{\alpha_i \in \Omega_j} \pi_i \tag{13}$$

where $\Omega_j = \{\alpha_i \,|\, S(\alpha_i)$ *does not contain any layer-*$(K-j)$ *codes*$\}$.

Size of Code Tree	**Size of the Markov Chain**	
	$\lvert\Phi_1(J,K)\rvert$	$\lvert\Phi_2(J,K)\rvert$
$K=0$	2	2
$K=1$	5	4
$K=2$	26	10
$K=3$	677	36
$K=4$	4.6×10^5	202
$K=5$	2.1×10^{11}	1828
$K=6$	4.4×10^{22}	2.7×10^4
$K=7$	1.9×10^{45}	6.9×10^5
$K=8$	3.8×10^{90}	3.0×10^7

Table 1: Size of the Markov Chain for Different Code Assignment Schemes and Tree Sizes, J=K+1.

5.2.2 Size of State Space

Let $\Phi_1(J,K)$ denote the set of all possible occupancy states of a K-layer code tree with J classes of calls. When $J=K+1$, all codes in the tree have the chance to be occupied. In this case, the K-layer code tree with J classes of calls can be decomposed into two $(K-1)$-layer sub-trees each with $(J-1)$ classes of calls. Therefore the size of $\Phi_1(J,K)$, denoted as $|\Phi_1(J,K)|$, can be calculated iteratively by

$$\begin{aligned}|\Phi_1(J,K)| &= |\Phi_1(J,J-1)| \\ &= |\Phi_1(J-1,J-2)|^2+1, \quad J=K+1,\ K\geq 1\end{aligned} \tag{14}$$

starting from $|\Phi_1(1,0)|=2$. Table 1 shows the values of $|\Phi_1(J,K)|$ for $0\leq K\leq 8$ and $J=K+1$. It can be derived from (14) that $2^{2^K}\leq|\Phi_1(J,K)|<2^{2^{K+1}}$.

When $1\leq J\leq K$, the codes from layer $(K-J)$ upwards will not be occupied. In this case, the size of $\Phi_1(J,K)$ can be derived by

$$|\Phi_1(J,K)| = [|\Phi_1(J,J-1)|]^{2^{K-J+1}}, \quad 1\leq J\leq K,\ K\geq 1 \tag{15}$$

where $|\Phi_1(J,J-1)|$ can be compute iteratively from (14).

5.3 Markov Chain for Rearrangeable Compact Assignment

Recall that for the blocking-triggered rearrangeable code assignment scheme, rearrangement is performed only for avoidable blockings. As unavoidable blockings only occurs when there is insufficient assignable capacity, the blocking probability for rearrangeable code assignment schemes is just the probability of these unavoidable blocking instants. In other words, whether the codes are being packed tightly or loosely when there are spare assignable capacity arround does not affect the blocking probability. With that, we introduce in the following an equivalent model for blocking probability calculation called *event-triggered* model. Under the event-triggered model, codes are rearranged as tightly as possible after every arrival or departure event.

Let n_j denote the number of ongoing class-j calls in the system. Under the event-triggered model, vector $\vec{n} = (n_0, n_1, \cdots, n_{J-1})$ can uniquely characterize the code occupancy status of the code tree and so can be taken as a state vector. The Markov chain for the event-triggered model under the case $K = 2$ and $J = 3$ is shown in Fig. 6. Note that the Markov chain in Fig. 6 can be derived from that in Fig. 3 by aggregating states with the same number of busy codes in each layer. As an example, the six states α_{11} to α_{16} in Fig. 3 all have two busy codes each accommodating a class-0 call and so can be collapsed into state $(2, 0, 0)$ in Fig. 6.

5.3.1 Blocking Probability

Let $\Phi_2(J, K)$ denote the state space under the event-triggered model and let $\pi_{\vec{n}}$ be the limiting probability for state $\vec{n}$. Since the event-triggered model is actually a multi-server Markovian queueing system with no waiting room and with linear constraints on the state space. Its solution was derived in [17] to be of the product form. Specifically, for state $\vec{n}$ in $\Phi_2(J, K)$,

$$\pi_{\vec{n}} = \pi_0 \cdot \prod_{j=0}^{J-1} \frac{1}{n_j!} \, (G_j)^{n_j} \tag{16}$$

where π_0 is the limiting probability of the empty state ($\vec{n} = (0, 0, \ldots, 0)$) and is given by

$$\pi_0 = \left[\sum_{\vec{n} \in \Phi_2(J,K)} \prod_{j=0}^{J-1} \frac{1}{n_j!} \, (G_j)^{n_j} \right]^{-1} \tag{17}$$

For a particular state $\vec{n}$, a class-j new call will be blocked if and only if the assignable capacity r of state $\vec{n}$ is less than 2^j. In other words,

$$P_B(j) = \sum_{\vec{n} \in \xi_j} \pi_{\vec{n}} \tag{18}$$

where $\xi_j = \left\{ \vec{n} \,|\, 2^K - \left[\sum_{i=0}^{J-1} n_i \, 2^i \right] < 2^j \right\}$.

5.3.2 Size of State Space

The state space $\Phi_2(J,K)$ contains all possible combinations of n_j's under the capacity constraint $\sum_{j=0}^{J-1} n_j\, 2^j \le 2^K$. In other words,

$$\Phi_2(J,K) = \{\vec{n}\,|\, \sum_{j=0}^{J-1} n_j \cdot 2^j \le 2^K\} \tag{19}$$

When $J = K+1$, (19) can be simplified to

$$\begin{aligned}\Phi_2(J,K) &= \bigcup_{i=0}^{2^K} \{\vec{n}\,|\, \sum_{j=0}^{J-1} n_j \cdot 2^j = i\} \\ &= \bigcup_{i=0}^{2^K} \theta_i, \quad J = K+1,\, K \ge 1\end{aligned} \tag{20}$$

where θ_i is the set of all binary partitions of the integer i (binary partition means the partitioning of an integer into powers of 2). Sloane's database of integer sequences [18] denotes $|\theta_i|$, the size of set θ_i, as sequence "$A018819$". It can be computed iteratively by

$$|\theta_i| = \begin{cases} |\theta_{i-1}|, & i\ odd; \\ |\theta_{i-1}| + |\theta_{\frac{i}{2}}|, & i\ even. \end{cases} \tag{21}$$

starting from $|\theta_0| = 1$. Since the set union operation in (20) is for different values of i, θ_i and θ_j are disjoint if $i \ne j$. Therefore, from (20), we obtain

$$|\Phi_2(J,K)| = \sum_{i=0}^{2^K} |\theta_i|, \quad J = K+1,\, K \ge 1 \tag{22}$$

Following the derivation in Appendix A, (22) can be further simplified to be

$$|\Phi_2(J,K)| = |\theta_{2^J}|, \quad J = K+1,\, K \ge 1 \tag{23}$$

The values of $|\Phi_2(J,K)|$ $(J = K+1)$ for $0 \le K \le 8$ are shown in Table 1. Comparing to non-rearrangeable code assignment schemes, the Markov chain size for the event-triggered model is much smaller.

When $1 \le J \le K$, $|\Phi_2(J,K)|$ can be derived from the definition of $\Phi_2(J,K)$ given in (19). To count all the possible combinations of n_j's $(0 \le j \le J-1)$ under the capacity constraint, let us start from n_{J-1}, the number of calls with highest bandwidth requirement in the system. Since there are altogether 2^{K-J+1} layer-$(K-J+1)$ codes for accommodating class-$(J-1)$ calls, the range of n_{J-1} is simply $[0, 2^{K-J+1}]$. As to class-$(J-2)$ calls, the maximum value of n_{J-2} under the capacity constraint is obtained by subtracting $2n_{J-1}$ (the number of non-assignable layer-$(K-J+2)$ codes) from 2^{K-J+2}

(total number of layer-$(K-J+2)$ codes). The range of n_{J-2} is then $[0, 2^{K-J+2}-2n_{J-1}]$. Similar reasoning holds for n_{J-3}, n_{J-4} and so on until n_0. Therefore, $|\Phi_2(J,K)|$ is given by

$$|\Phi_2(J,K)| = \sum_{n_{J-1}=0}^{2^{K-J+1}} \sum_{n_{J-2}=0}^{2^{K-J+2}-2n_{J-1}} \sum_{n_{J-3}=0}^{2^{K-J+3}-4n_{J-1}-2n_{J-2}} \cdots \sum_{n_0=0}^{2^K-2^{J-1}n_{J-1}-2^{J-2}n_{J-2}-\cdots-2n_1} 1, \quad 1 \le J \le K+1,\ K \ge 1 \tag{24}$$

5.4 Overall Blocking Probability P_B

With the blocking probability $P_B(j)$ of class-j calls given in (13) and (18), the overall blocking probability P_B is simply the weighted sum of $P_B(j)$'s, or

$$P_B = \sum_{j=0}^{J-1} \frac{G_j}{G} \cdot P_B(j) \tag{25}$$

5.5 System Throughput T

The offered load L to the system is defined as offered traffic weighted by the bandwidth requirements (in unit of R). Specifically,

$$L = \sum_{j=0}^{J-1} L_j = \sum_{j=0}^{J-1} G_j \cdot 2^j \tag{26}$$

where $L_j = G_j \cdot 2^j$ is the offered load (in unit of R) of class-j calls. The throughput of class-j calls, denoted as T_j, is then given by

$$T_j = [1 - P_B(j)] \cdot L_j \tag{27}$$

The system throughput T is just the sum of T_j's.

5.6 Fairness Index F

Fairness measures have been studied extensively in the literature. For a J-class system, a convenient fairness index F derived from the definition of variance is given in [19] as

$$F = \frac{\left\{\sum_{j=0}^{J-1} [1 - P_B(j)]\right\}^2}{J \sum_{j=0}^{J-1} [1 - P_B(j)]^2} \tag{28}$$

As seen, F is nonnegative and its maximum value of one is achieved when all $P_B(j)$'s are the same, indicating all classes of calls have the same probability of getting served.

6 Numerical and Simulation Results

In UTRA-FDD [20], the spreading factor of the uplink Dedicated Physical Data Channels (DPDCH) may range from 4 to 256. The downlink DPDCH and the downlink Dedicated Physical Control Channels (DPCCH) are merged into the so called downlink Dedicated Physical Channels (DPCH) by time multiplexing with spreading factor ranging from 4 to 512. For UTRA-TDD, the range of spreading factor that may be used for uplink physical channels is from 1 to 16 [21].

Without loss of generality, our simulation model consists of a 6-layer code tree ($K = 6$) with a total system capacity $2^K = 64$ (in unit of R) and four classes of calls ($J = 4$). Further, we assume G_j's take on two different ratios:

Case I: $G_0 : G_1 : G_2 : G_3 = 1 : 1 : 1 : 1$. This is the case where the number of calls per second is the same for all four classes. The offered loads for the four classes, however, are in the ratio $L_0 : L_1 : L_2 : L_3 = 1 : 2 : 4 : 8$.

Case II: $G_0 : G_1 : G_2 : G_3 = 8 : 4 : 2 : 1$. This is the case where the bandwidths required by the four classes of calls are the same. In other words, $L_0 : L_1 : L_2 : L_3 = 1 : 1 : 1 : 1$.

Fig. 7 shows the overall blocking probability P_B versus total offered traffic G under different code assignment schemes. We see that, in both cases, RCA gives the lowest blocking probability among the three schemes considered. Further, the analytical results in solid lines matches well with the simulation results in dashed lines. Also, the performance improvements of CA and RCA over that of RA are very significant over the entire range of offered traffic. Specifically, at $G = 16$ in *Case II*, $P_B^{RCA} = 0.005$, $P_B^{CA} = 0.008$ and $P_B^{RA} = 0.045$.

It is seen that at the same G value, *Case II* has much smaller blocking than *Case I*. This is expected as most of the calls are the low-bandwidth type for *Case II* and hence the total load is much smaller than that for *Case I*.

Fig. 8 shows the system throughput T as a function of offered load L for *Case II*. The 95% confidence intervals are all made comparable to the marker size shown. It is seen that under CA and RCA, the system throughputs are monotonically increasing with respect to the offered load and are uniformly higher than that of RA. Specifically, at offered load $L = 64$ (i.e. at system capacity), the throughput values for RA, CA and RCA schemes are 41, 48 and 52, respectively. Similar results hold for *Case I* and are therefore omitted.

Fig. 9 compares the throughput under the two cases of traffic mix for RCA. The analytical results show that *Case II* is more efficient in utilizing the system resources as there are more low-bandwidth calls.

Fig. 10 shows the fairness index F as a function of G for *Case II*. It demonstrates that CA and RCA are *fairer* to different classes of calls than RA over the entire loading range.

7 Discussions and Comparisons

7.1 Performance Metrics

In Section 5.3, the event-triggered model is used for analyzing the blocking performance of RCA. Actually, it is equivalent to the model for studying "complete sharing policy" in shared resource environment [17][4]. RCA and other proposed rearrangeable code assignment schemes [6, 7, 8, 9, 15] all use the "complete sharing policy" in code assignments. Therefore, all these proposed schemes have the same blocking performance as RCA, which is given by (18) and (25).

System throughput T could be maximized by applying Hardy's theorem [22]. To do so, the system needs to adjust (say by blocking calls) the J individual blocking probabilities always in the reversed order with respect to the corresponding offered loads. In other words, for maximum system throughput, the largest load class should have the lowest blocking probability; the second largest load class should have the second lowest blocking probability and so on. This is a complex operation and has the fairness concern of discriminating the low offered load class(es).

Very often, maintaining the fairness of service quality among different classes is more important than purely maximizing the system throughput. We have shown that CA and RCA are very fair under the First-Come-First-Serve (FCFS) discipline. If differentiated blocking performance needs to be maintained, other strategies, such as Sharing with Minimum Allocation (SMA) and Sharing with Maximum Queue Length (SMXQ) [23], can be used. This, however, is beyond the scope of this paper.

7.2 Signaling Overhead

Each code rearrangement leads to a set of signalings between the base station and the mobile station. In [7, 9], codes are rearranged after each call departure and the scope of rearrangement is the whole tree. As a result, more code rearrangements are needed than RCA scheme and the "minimum-cost" branch scheme [15], where code rearrangements are performed only when avoidable blockings occur and the scope of rearrangement is limited to the particular branch concerned. Specifically, in [15], the "minimum-cost" branch is chosen (if possible) for code rearrangement, thereby the number of code rearrangements (or signaling overhead) for accommodating a new call is minimized. On the other hand, the RCA scheme simply identifies the *least loaded* branch (by the MFCR Algorithm) for further code rearrangements reduction. As the number of code rearrangements needed for emptying a branch and the code rearrangement efficiency are both proportional to the branch load, the choice of least loaded branch in RCA also means lighter signaling overhead.

[4] In [17], the "complete sharing policy" is defined as "a customer requiring b units of resource is blocked if and only if fewer than b units of the (total) resource is available".

7.3 Computational Complexity

As mentioned in Section 1, computational complexity lies mainly in identifying the proper branch for emptying. In [15], three "minimum-cost" branch searching algorithms are presented. Among them, *exhaustive search* has very high computational complexity; *code pattern search* does not always yield a "minimum-cost" branch, i.e. it is not accurate enough; and *topology search* needs the assistance of a cost comparison table, whose size grows dramatically with the increase of tree size and traffic classes [6]. While in MFCR, to find out the least loaded branch (with its root code in layer-k), the loads of 2^k branches (at most) need to be compared. For each branch, the assignable capacity (defined in (10)) is the sums of 2^{K-k} code assignability index function values ($I_A^{(A,i)} = 1$ or 0 as defined in (4)). Altogether, the least loaded branch(es) can be identified by at most $(2^{K-k}-1)\cdot 2^k \cdot (2^{k-1}) = 2^{K+k} - 4^k - 2^K + 2^k < 2^{K+k}$ operations.

Besides the lower time complexity, the storage requirements of CA and RCA are both minimum: only the $(2^{K+1}-1)$ code assignability index function values need to be maintained since no table look-up is required.

8 Conclusions

The flexibility index of a code tree is a measure of how well the code tree can support multi-rate traffic. In general, code choice should obey the principle that the code should be as tightly fit into the existing busy code body as possible so as to leave the maximum flexibility for the remaining codes to accommodate future multi-rate calls. Depending on specific implementations, a code can or cannot be rearranged after assignment. For each case, we have proposed a computational efficient code assignment scheme and analyzed its performance. Analytical results, verified by simulation results, shown that the proposed schemes are efficient, stable and fair.

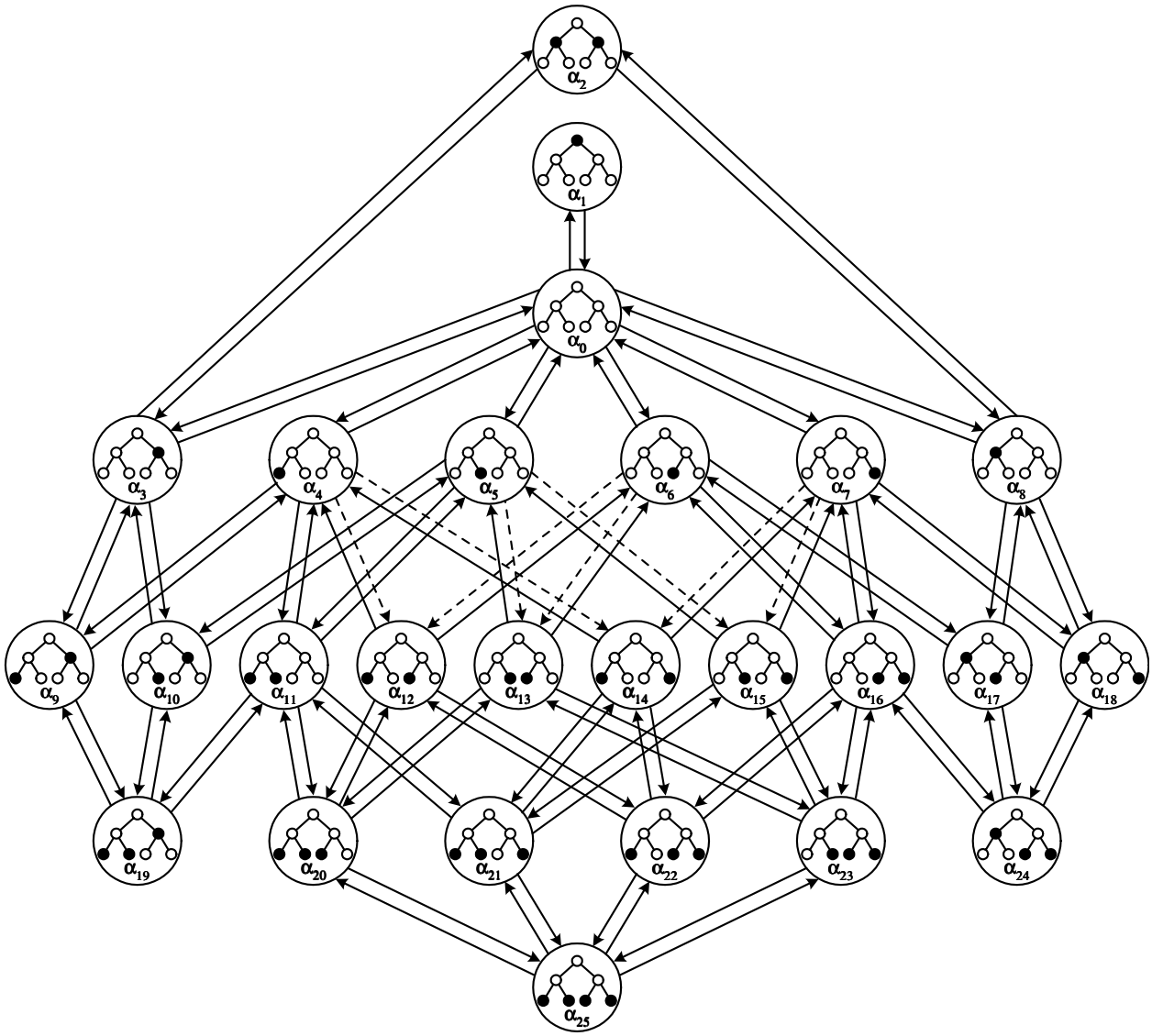

Figure 3: Markov Chains for RA and CA Schemes, $K = 2$ and $J = 3$.

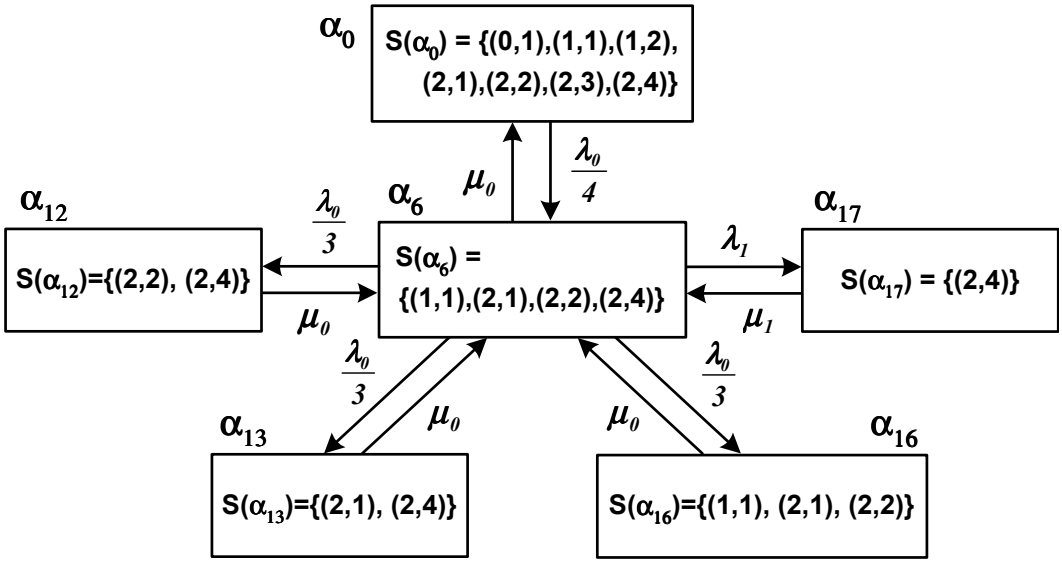

Figure 4: Transition Rates of State α_6 in RA Scheme

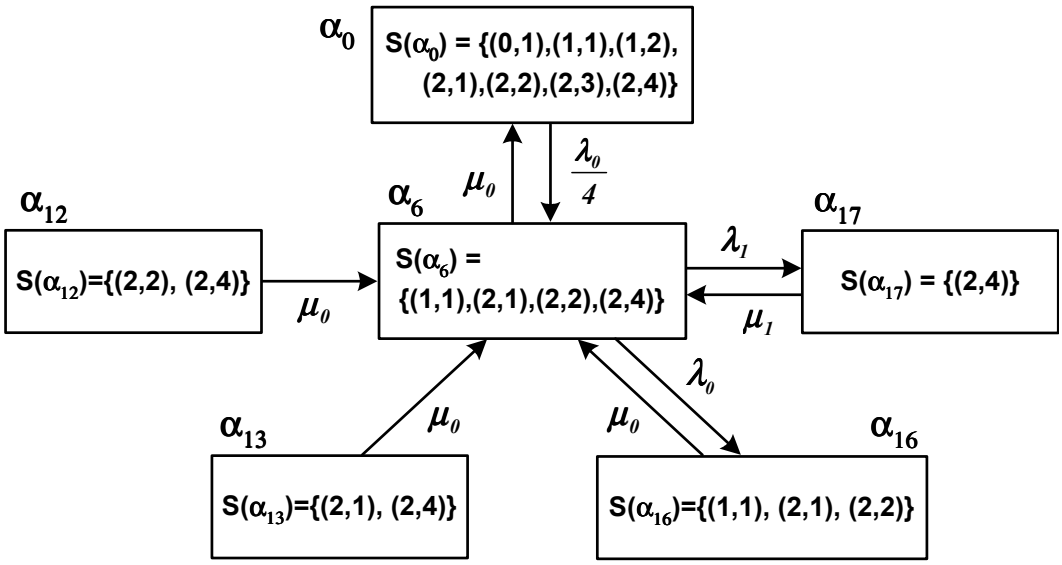

Figure 5: Transition Rates of State α_6 in CA Scheme

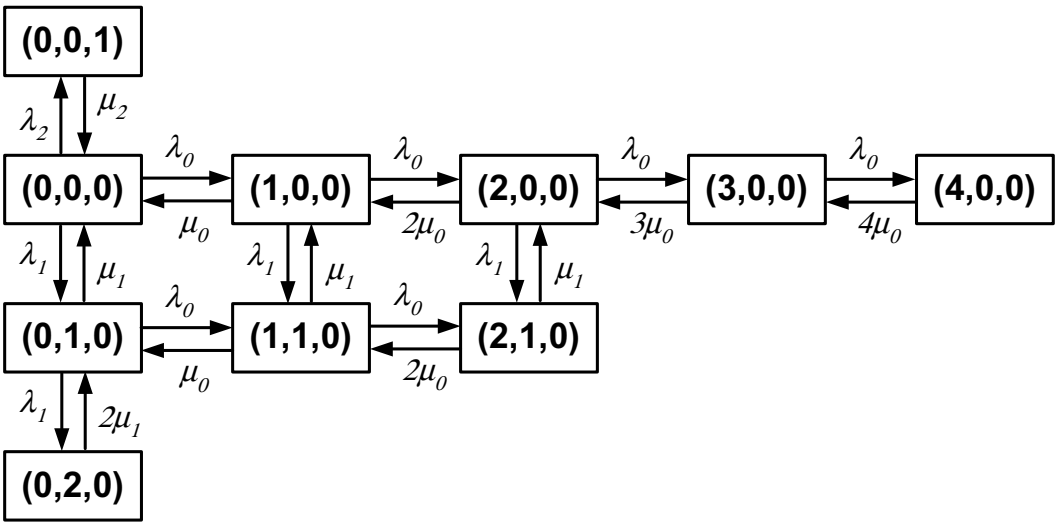

Figure 6: Markov Chain for the Event-Triggered Model of RCA Scheme, $K = 2$ and $J = 3$.

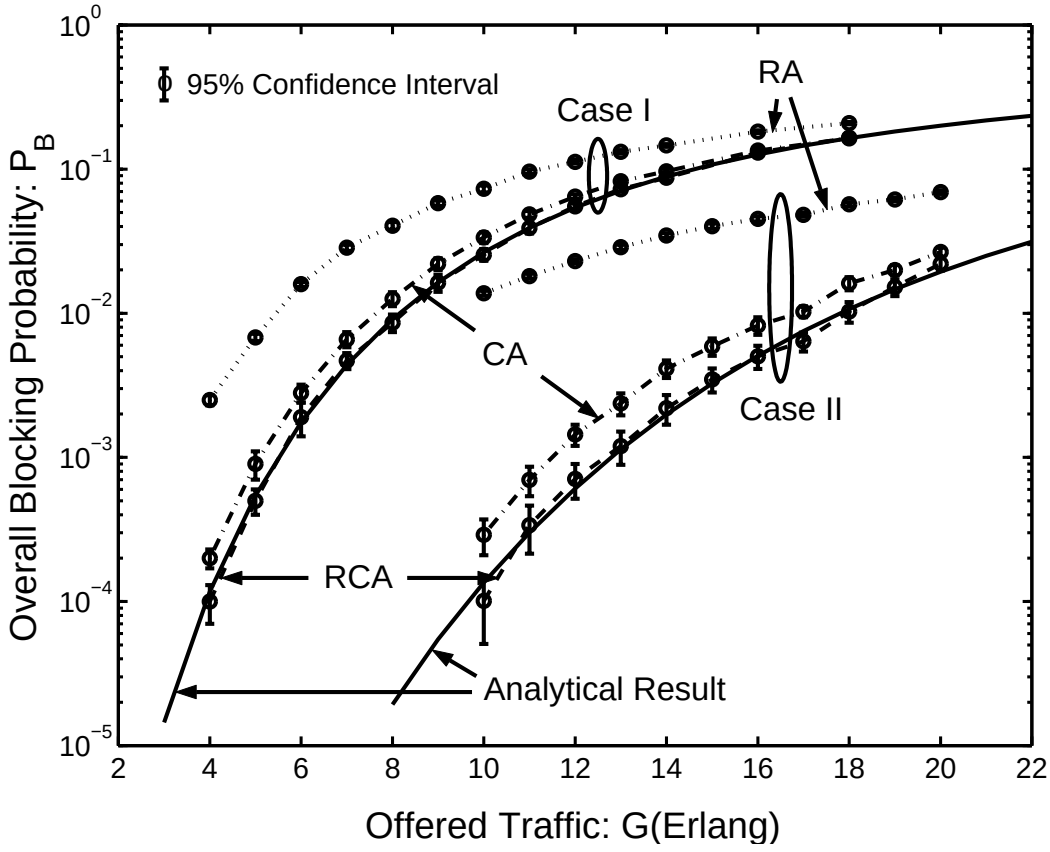

Figure 7: Overall Blocking Probability

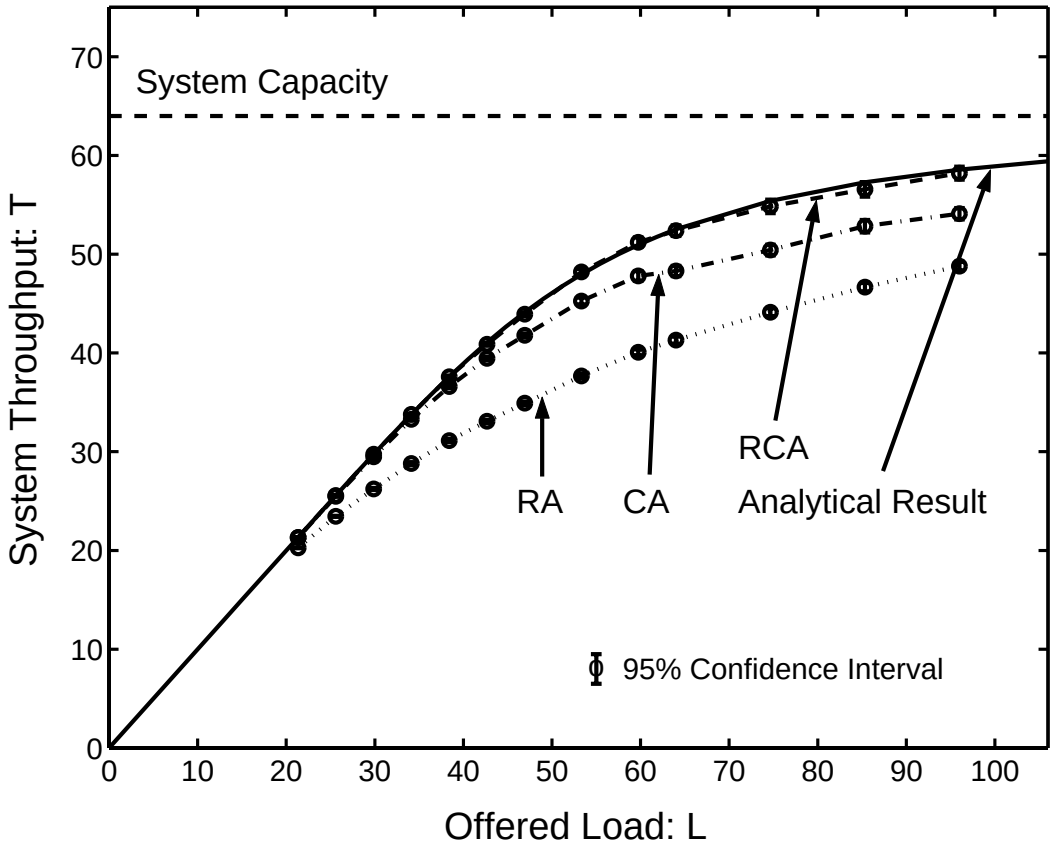

Figure 8: System Throughput, *Case II*.

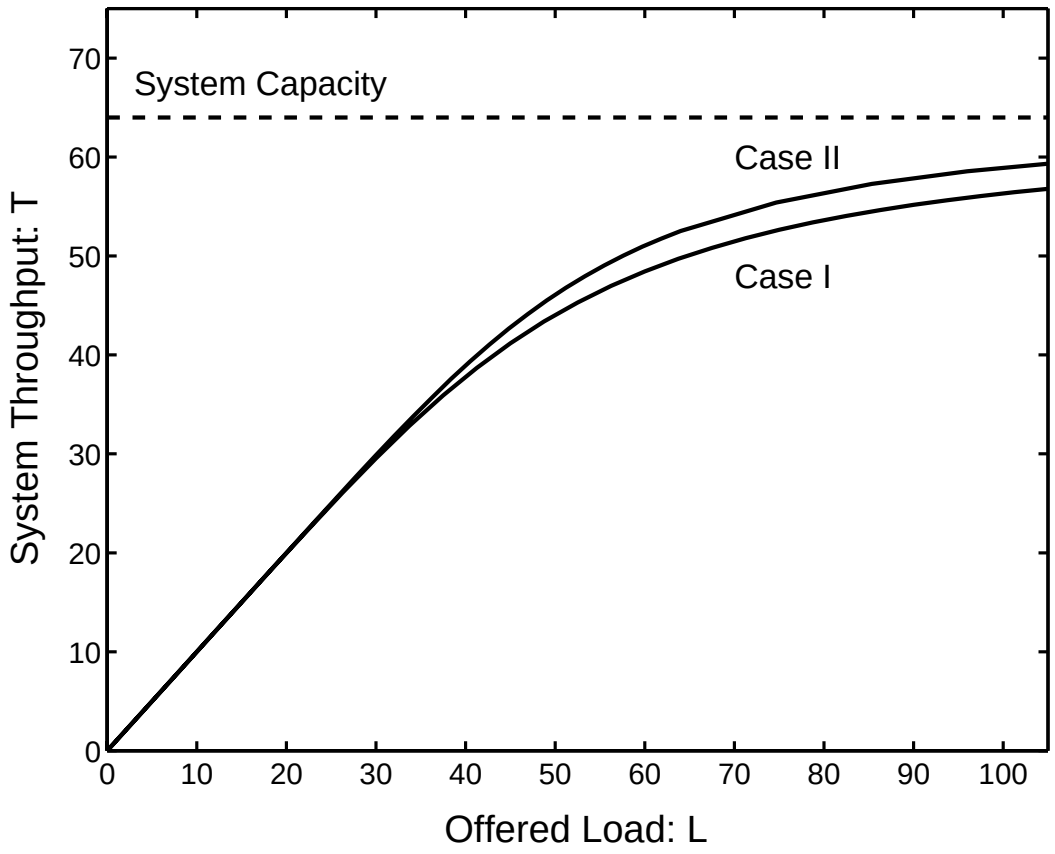

Figure 9: System Throughput of RCA, Analytical Results.

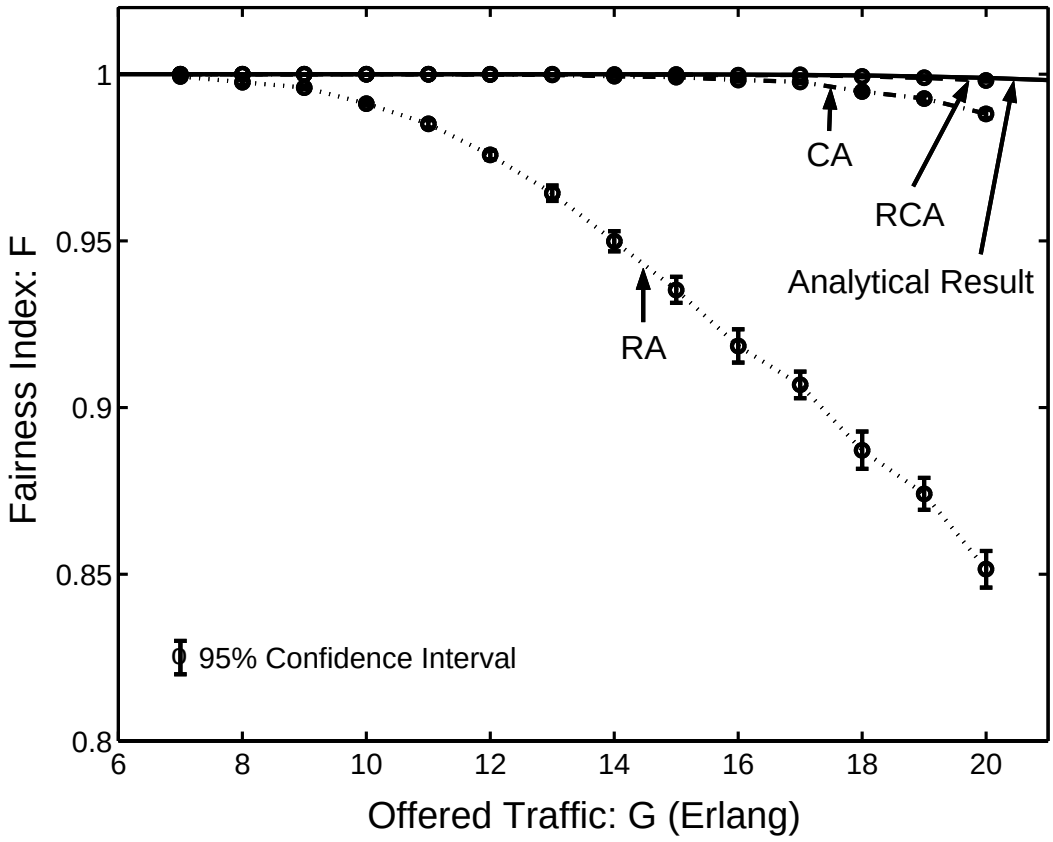

Figure 10: Fairness Index, *Case II*.

Appendix *A*

Based on the definition of $|\theta_i|$ given in (21), equation (23) can be derived as follows.

$$
\begin{aligned}
|\Phi_2(J,K)| &= \sum_{i=0}^{2^K} |\theta_i| \\
&= |\theta_0| + \sum_{i=1}^{2^K} |\theta_i| \\
&= |\theta_1| + \sum_{i=1}^{2^K} |\theta_i| \\
&= |\theta_1| + |\theta_1| + \sum_{i=2}^{2^K} |\theta_i| \\
&= |\theta_2| + \sum_{i=2}^{2^K} |\theta_i| \\
&= |\theta_3| + \sum_{i=2}^{2^K} |\theta_i| \\
&= |\theta_3| + |\theta_2| + \sum_{i=3}^{2^K} |\theta_i| \\
&= |\theta_4| + \sum_{i=3}^{2^K} |\theta_i| \\
&= |\theta_5| + \sum_{i=3}^{2^K} |\theta_i| \\
&= |\theta_5| + |\theta_3| + \sum_{i=4}^{2^K} |\theta_i| \\
&\vdots \\
&= |\theta_{2^{K+1}-1}| + |\theta_{2^K}| \\
&= |\theta_{2^{K+1}}| \\
&= |\theta_{2^J}|, \qquad\qquad J = K+1,\ K \geq 1
\end{aligned}
$$

References

[1] 3GPP TS 25.213 (V4.2.0), "Spreading and modulation (FDD)," Technical Specification (Release 4), Technical Specification Group Radio Access Network, 3GPP, Dec.

2001.

[2] 3GPP TS 25.223 (V4.3.0), "Spreading and modulation (TDD)," Technical Specification (Release 4), Technical Specification Group Radio Access Network, 3GPP, Dec. 2001.

[3] R. G. Cheng and P. Lin, "OVSF code channel assignment for IMT-2000," in *Proceedings of IEEE VTC 2000*, vol. 3, pp. 2188–2192, Spring 2000.

[4] F. Shueh, Z. E. P. Liu, and W. S. E. Chen, "A fair, efficient, and exchangeable channelization code assignment scheme for IMT-2000," in *Proceedings of IEEE ICPWC 2000*, pp. 429–433, 2000.

[5] E. Dahlman and K. Jamal, "Wide-band services in a DS-CDMA based FPLMTS system," in *Proceedings of IEEE VTC 1996*, vol. 3, pp. 1656–1660, Apr. 1996.

[6] M. Dell'Amico, M. L. Merani, and F. Maffioli, "Efficient algorithms for the assignment of ovsf codes in wideband CDMA," in *Proceedings of IEEE ICC 2002*, vol. 5, pp. 3055–3060, 2002.

[7] R. Fantacci and S. Nannicini, "Multiple access protocol for integration of variable bit rate multimedia traffic in UMTS/IMT-2000 based on wideband CDMA," *IEEE Journal on Selected Areas in Communications*, vol. 18, pp. 1441–1454, Aug. 2000.

[8] C. E. Fossa Jr. and N. J. Davis IV, "Dynamic code assignment improves channel utilization for bursty traffic in third-generation wireless networks," in *Proceedings of IEEE ICC 2002*, vol. 5, pp. 3061–3065, 2002.

[9] W. T. Chen, Y. P. Wu, and H. C. Hsiao, "A novel code assignment scheme for W-CDMA systems," in *Proceedings of IEEE VTC 2001*, vol. 2, pp. 1182–1186, Fall 2001.

[10] M. Zhang and T.-S. P. Yum, "Comparisons of channel-assignment strategies in cellular mobile telephone systems," *IEEE Transactions on Vehicular Technology*, vol. 38, pp. 211–215, Nov. 1989.

[11] S. M. Elnoubi, R. Singh, and S. C. Gupta, "A new frequency channel assignment algorithm in high capacity mobile communication systems," *IEEE Transactions on Vehicular Technology*, vol. VT-31, Aug. 1982.

[12] R. Assarut, K. Kawanishi, U. Yamamoto, Y. Onozato, and M. Matsushita, "Region division assignment of orthogonal variable-spreading-factor codes in W-CDMA," in *Proceedings of IEEE VTC 2001*, vol. 3, pp. 1884–1888, Fall 2001.

[13] J. S. Engel and M. M. Peritsky, "Statistically-optimum dynamic server assignment in systems with interfering servers," *IEEE Transactions on Communications*, vol. COM-21, pp. 1287–1293, Nov. 1973.

[14] T. J. Kahwa and N. D. Georganas, "A hybrid channel assignment scheme in large-scale, cellular-structured mobile communication systems," *IEEE Transactions on Communications*, vol. COM-26, Apr. 1978.

[15] T. Minn and K. Y. Siu, "Dynamic assignment of orthogonal variable-spreading-factor codes in W-CDMA," *IEEE Journal on Selected Areas in Communications*, pp. 1429–1440, Aug. 2000.

[16] F. Adachi, M. Sawahashi, and K. Okawa, "Tree-structured generation of orthogonal spreading codes with different lengths for forward link of DS-CDMA mobile radio," *Electronics Letters*, vol. 33, pp. 27–28, Jan. 1997.

[17] J. S. Kaufman, "Blocking in a shared resource environment," *IEEE Transactions on Communications*, vol. COM-29, pp. 1474–1481, Oct. 1981.

[18] N. J. A. Sloane, "Sloane's on-line encyclopedia of integer sequences," *http://www.research.att.com/∼njas/sequences/index.html*.

[19] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. John Wiley and Sons, 1991.

[20] 3GPP TS 25.211 (V4.3.0), "Physical channels and mapping of transport channels onto physical channels (FDD)," Technical Specification (Release 4), Technical Specification Group Radio Access Network, 3GPP, Dec. 2001.

[21] 3GPP TS 25.221 (V4.3.0), "Physical channels and mapping of transport channels onto physical channels (TDD)," Technical Specification (Release 4), Technical Specification Group Radio Access Network, 3GPP, Dec. 2001.

[22] S. M. Ross, *Introduction to Stochastic Dynamic Programming*. Academic Press, 1983.

[23] T.-S. P. Yum and C. Dou, "Buffer allocation strategies with blocking requirements," *Performance Evaluation*, vol. 4, pp. 285–295, North-Holland 1984.