LETTER
Algorithmic Computation of Optimal Capacity in Multiservice Mobile Wireless Networks*

Vicent PLA†, Member, Jorge MARTÍNEZ†, Nonmember, and Vicente CASARES-GINER†, Member

SUMMARY In this paper we propose a new algorithm for computing the optimal configuration of the Multiple Fractional Guard Channel (MFGC) admission policy in multiservice mobile wireless networks. The optimal configuration maximizes the offered traffic that the system can handle while meeting certain QoS requirements. The proposed algorithm is shown to be more efficient than previous algorithms appeared in the literature.

key words: mobile wireless network, multiservice, admission control, capacity, algorithms

1. Introduction

The Multiple Fractional Guard Channel (MFGC) strategy [1, 2] is an efficient admission policy for multiservice mobile wireless networks. We seek to find the optimal configuration of the MFGC admission policy, i.e. the one that maximizes the offered traffic that the system can handle while meeting the QoS requirements. To the best of our knowledge only one algorithm for this purpose has been proposed [2] in the literature. Ours offers substantial computational advantages. Besides, we observe that a further enhancement of the algorithm is possible by eliminating the iterative procedure of computing the handoff call arrival rates.

2. Model Description

The system has a total of $C$ resource units. The physical meaning of a unit of resources will depend on the specific technological implementation of the radio interface. The system offers $N$ different classes of services. For each type of service new and handoff call arrivals are distinguished so that there are $N$ types of services and $2N$ types of arrivals. Arrivals are numbered in such a manner that for service $i$ new call arrivals are referred to as arrival type $i$, whereas handoff arrivals are referred to as arrival type $N + i$. For the sake of mathematical tractability we make the common assumptions of Poisson arrival processes and exponentially distributed random variables for cell residence time and call duration. The arrival rate for new (handoff) calls of service $i$ is $\lambda_i^N (\lambda_i^H)$. A request of service $i$ consumes $b_i$ resource units, $b_i \in \mathbb{N}$. The call duration of service $i$ is exponentially distributed with rate $\mu_i^c$. The cell residence time of a service $i$ customer is exponentially distributed with rate $\mu_i^r$. Hence, the resource holding time in a cell for service $i$ is exponentially distributed with rate $\mu_i = \mu_i^c + \mu_i^r$. If we denote by $\mathbf{p} = (P_1, \ldots, P_{2N})$ the blocking probabilities for each of the $2N$ arrival streams, the new call blocking probabilities is $P_i^N = P_i$, the handoff failure probability is $P_i^H = P_{N+i}$ and the forced termination probability [3] $P_i^{ft} = P_i^H/(\mu_i^c/\mu_i^r + P_i^H)$.

If the system is in statistical equilibrium the handoff arrival rates are related to the new call arrival rates and the blocking probabilities ($P_i$) through the expression

$$\lambda_i^N = \lambda_i^H (1 - P_i^N)/(\mu_i^c/\mu_i^r + P_i^H) \quad [3].$$

The blocking probabilities do in turn depend on the handoff arrival rates yielding a system of non-linear equations which can be solved using a fixed point iteration method as described in [3]. The system state is described by an $N$-tuple $\mathbf{x} = (x_1, \ldots, x_N)$, where $x_i$ represents the number of type $i$ calls in the system, regardless they were initiated as new or handoff calls. Let $b(x)$ represent the amount of occupied resources at state $x$,

$$b(x) = \sum_{i=1}^{N} x_i b_i.$$

The MFGC policy operates in a manner that the number of resource unit that stream $i$ can dispose of is, on average, $t_i$. In order to decide on the acceptance of a request of type $i$, upon arrival the amount of resources that will be occupied if it is accepted is compared with the corresponding threshold $t_i$: if $b(x) + b_i \leq [t_i], $ accept; if $b(x) + b_i > [t_i] + 1, $ accept with probability $t_i - [t_i]; $ if $b(x) + b_i > [t_i] + 1, $ reject.

3. Optimal Capacity: Algorithm

The algorithm computes the system capacity, i.e. the maximum offered traffic that the network can handle while meeting certain QoS requirements. These QoS requirements are given in terms of upper-bounds for the new call blocking probabilities ($B_i^N$) and the forced termination probabilities ($B_i^{ft}$). Let $\lambda^T = \sum_{i \leq N} \lambda_i^N$ be the aggregated call arrival rate and let $f_i (0 \leq f_i < 1, \sum_{i \leq N} f_i = 1)$ represent the fraction of $\lambda^T$ that correspond to service $i$, i.e. $\lambda_i^N = f_i \lambda^T$. The capacity optimization problem can be formally stated as follows

Given: $C, b_i, f_i, \mu_i^c, \mu_i^r, B_i^N, B_i^{ft}; i = 1, \ldots, N$
Maximize: $\lambda^T$ by finding the appropriate MFGC parameters $t_i; i = 1, \ldots, 2N$
Subject to: $P_i^N \leq B_i^N, P_i^H \leq B_i^{ft}; i = 1, \ldots, N$

We propose an algorithm to work out this capacity op-

Copyright © 2005 The Institute of Electronics, Information and Communication Engineers
timization problem. Our algorithm has a main part (Algorithm capacity) from which the procedure solveMFGC is called. The procedure solveMFGC does, in turn, call another procedure (MFGC) that calculates the blocking probabilities. Let us introduce the 2N-tuple $p_{max} = (B_1^0, \ldots, B_N^0, B_1^* \ldots, B_N^*)$ as the upper-bounds vector for the blocking probabilities, where the value of $B_i^0$ is given by $B_i^0 = (\mu_i/\mu'_i)B_i^*/(1 - B_i^*)$.

Algorithm:

\[(\lambda_{max}^*, t_{opt}) = \text{capacity}(p_{max}, f, \mu^c, \mu', b, C)\]

\[e_1 := \text{precision} > 0; L := 0; U := \text{high value} > (possible, t) := \text{solve}_MFGC(p_{max}, U f, \mu^c, \mu', b, C)\]

\[\text{atLeastOnce} := \text{FALSE};\]

\[\text{while possible do}\]

\[L := U; t_L := t; \text{atLeastOnce} := \text{TRUE};\]

\[\text{if possible then} \quad L := \lambda; t_L := t; \text{atLeastOnce} := \text{TRUE};\]

\[\text{else} \quad U := \lambda; \text{atLeastOnce} := \text{TRUE};\]

\[\text{until } (U - L)/L \leq e_1 \text{ AND atLeastOnce}\]

\[\lambda_{max}^* := L; t := t_L;\]

Procedure:

\[(\text{possible, } t) = \text{solve}_MFGC(p_{max}, \lambda_n, \mu^c, \mu', b, C)\]

\[e_2 := \text{precision} > 0; \delta := \text{small value} > t := (\delta, \delta, \ldots, \delta)\]

\[p := \text{MFGC}(t, \lambda_n, \mu^c, \mu', b, C)\]

\[\text{repeat}\]

\[\text{if } \exists z > \text{optimal prioritization order is found. For each element the HCO algorithm uses} \]

\[\text{canConverge} := \text{TRUE}; i := 1;\]

\[\text{repeat}\]

\[\text{if } \exists z > \text{optimal prioritization order is found. For each element the HCO algorithm uses} \]

\[\text{canConverge} := \text{FALSE};\]

\[\text{else}\]

\[L := i(t); U := C\]

\[\text{repeat}\]

\[\text{if } \exists z > \text{optimal prioritization order is found. For each element the HCO algorithm uses} \]

\[\text{canConverge} := \text{FALSE};\]

\[\text{else}\]

\[L := i(t); U := C\]

\[\text{repeat}\]

\[\text{if } \exists z > \text{optimal prioritization order is found. For each element the HCO algorithm uses} \]

\[\text{canConverge} := \text{FALSE};\]

\[\text{else}\]

\[L := i(t); U := C\]

\[\text{repeat}\]

\[\text{if } \exists z > \text{optimal prioritization order is found. For each element the HCO algorithm uses} \]

\[\text{canConverge} := \text{FALSE};\]

\[\text{else}\]

\[L := i(t); U := C\]

\[\text{until } (1 - e_2)p_{max}(i) \leq p(i) \leq p_{max}(i)\]

\[\text{end if}\]

\[\text{end if}\]

\[i := i + 1\]

\[\text{until } (i > 2N) \text{ OR (NOT(canConverge))}\]

\[\text{if canConverge then}\]

\[\text{if } \exists z > \text{optimal prioritization order is found. For each element the HCO algorithm uses} \]

\[\text{possible} := \text{TRUE}; \text{exit} := \text{TRUE};\]

\[\text{else}\]

\[\text{exit} := \text{FALSE};\]

\[\text{else}\]

\[\text{possible} := \text{FALSE}; \text{exit} := \text{TRUE};\]

\[\text{until exit}\]

The procedure MFGC, which is invoked in the innermost loop of our algorithm, is used to obtain the blocking probabilities $p := \text{MFGC}(t, \lambda_n, \mu^c, \mu', b, C)$. For this computation an iterative procedure is normally required in order to obtain the value of the handoff request rates. At each iteration a multidimensional birth-and-death process is solved which constitutes the most computationally expensive part of the algorithm. The following observation can be used to speed up the algorithm since it permits to eliminate the above mentioned iterative procedure. Each run of solveMFGC tries to find $t$ so that $p = p_{max}$ (within tolerance limit). Thus, in order to compute $\lambda_{opt}$ we use the expression $\lambda_{opt} = (1 - B_i^*)/(\mu_i/\mu'_i + B_i^*)$ in which $B_i^*$ is explicitly defined.

4. Numerical Evaluation

In this section we evaluate the numerical complexity of our algorithm. To this end we used the algorithm proposed by Heredia et al. in [1, 2] as a reference. Henceforth we refer to this algorithm as HCO after its authors’ initials.

The HCO algorithm requires the optimal prioritization order as input, i.e. a list of call types sorted by their relative priorities [2]. If $t$ is the policy setting for which the maximum capacity is achieved, the optimal prioritization order is the permutation $\sigma^* \in \Sigma, \Sigma := \{i_1, \ldots, i_{2N}\} : \sigma_i \in N, 1 \leq \sigma_i \leq 2N\}$, such that $t(\sigma_i^*) \leq t(\sigma_j^*) \leq \cdots \leq t(\sigma_{2N}^*) = C$. Selecting the optimal prioritization order is a complicated task as it depends on both QoS constraints and system characteristics as pointed out in [2]. In general there are a total of $(2N)!$ different prioritization orders. In [2] the authors give some guidelines to construct a partially sorted list of prioritization orders according to their likelihood of being the optimal ones. Then a trial and error process is followed using successive elements of the list until the optimal prioritization order is found. For each element the HCO algorithm is run and if after a large number of iterations it did not converge, another prioritization order is tried.

Our algorithm does not require any a priori knowledge. Indeed, after obtaining the policy setting $t$ for which the maximum capacity is achieved, the optimal prioritization order is automatically determined as a by-product of our algorithm. This constitutes by itself a significant advantage of our algorithm over the HCO algorithm. Moreover, in what follows we show through numerical examples that our algorithm is still more efficient than the HCO algorithm when the latter is provided with the optimal prioritization order.

For the numerical examples we considered a system with two services ($N = 2$) and the values of the parameters are $b = (1, 2), f = (0.8, 0.2), \mu^c = (1/180, 1/300), \mu' = (1/900, 1/1000), B_{n} = (0.02, 0.02), B_{r} = (0.002, 0.002); all tolerances have been set to $e_1 = 10^{-2}$.

A comparison of the number of floating point operations (flops) required by the HCO algorithm and our algorithm is shown in Table 1. We tried other configuration, which are not shown here due to the space constraints, and
similar results were obtained. The speed-up technique divides the flop count by a factor of about three.

It is worth noting that, as expected, the disagreement among the values obtained for the optimal capacity computed using the different methods was within tolerance in all tested cases. The same can be said for the policy setting $t$.

### 5. Conclusions

We proposed a new algorithm for computing the optimal configuration of the Multiple Fractional Guard Channel (MFGC) admission policy in multiservice mobile wireless networks. The optimal configuration maximizes the offered traffic that the system can handle while meeting certain QoS requirements. Compared to a recently published algorithm (HCO) ours offers the advantage of not needing a call prioritization order as input. We observed that a further enhancement of the algorithm is possible by eliminating the iterative procedure for computing the handoff call arrival rates. Numerical examples show that our algorithm is faster than the HCO algorithm even if the latter is provided with the optimal prioritization order.

### References

