LETTER

Algorithmic Computation of Optimal Capacity in Multiservice Mobile Wireless Networks*

Vicent PLA[†a)], ***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, Algorithmics*

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 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 λ_i^n (λ_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 μ_i^c. The cell residence time of a service i customer is exponentially distributed with rate μ_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 $\boldsymbol{p} = (P_1, \ldots, P_{2N})$ 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 $\lambda_i^h = \lambda_i^n(1 - P_i^n)/(\mu_i^c/\mu_i^r + P_i^h)$ [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 $\boldsymbol{x} = (x_1, \ldots, x_N)$, where x_i represents the number of type i calls in the system, regardless they were initiated as new or handoff calls. Let $b(\boldsymbol{x})$ represent the amount of occupied resources at state $\boldsymbol{x}$, $b(\boldsymbol{x}) = \sum_{i=1}^{N} x_i b_i$.

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 $b(\boldsymbol{x}) + b_i \leq \lfloor t_i \rfloor$, accept; if $b(\boldsymbol{x}) + b_i = \lfloor t_i \rfloor + 1$, accept with probability $t_i - \lfloor t_i \rfloor$; if $b(\boldsymbol{x}) + b_i > \lfloor t_i \rfloor + 1$, 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 $\lambda^T = \sum_{1 \leq i \leq N} \lambda_i^n$ be the aggregated call arrival rate and let f_i ($0 \leq f_i < 1, \sum_{1 \leq i \leq N} f_i = 1$) represent the fraction of λ^T that correspond to service i, i.e. $\lambda_i^n = f_i \lambda^T$. The capacity

Manuscript received August 30, 2004.

[†]The authors are with the Department of Communications, Universidad Politécnica de Valencia (UPV), ETSIT Camí de Vera s/n, 46022 Valencia, Spain.

a) E-mail: vpla@dcom.upv.es

*This work has been supported by the *Spanish Ministry of Science and Technology* under projects TIC2001-0956-C04-04 and TIC2003-08272.

optimization problem can be formally stated as follows

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

We propose an algorithm to work out this capacity optimization 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 $\boldsymbol{p_{max}} = (B_1^n, \ldots, B_N^n, B_1^h, \ldots, B_N^h)$ as the upper-bounds vector for the blocking probabilities, where the value of B_i^h is given by $B_i^h = (\mu_i^c/\mu_i^r)B_i^{ft}/(1 - B_i^{ft})$.

Algorithm:

$(\lambda_{max}^T, \boldsymbol{t}_{opt})$=capacity$(\boldsymbol{p_{max}}, \boldsymbol{f}, \boldsymbol{\mu^c}, \boldsymbol{\mu^r}, \boldsymbol{b}, C)$

$\varepsilon_1 :=<$ precision $>$; $L := 0$; $U :=<$ high value $>$
(possible, $\boldsymbol{t}$) := solve_MFGC$(\boldsymbol{p_{max}}, U\boldsymbol{f}, \boldsymbol{\mu^c}, \boldsymbol{\mu^r}, \boldsymbol{b}, C)$
atLeastOnce:=FALSE;
while possible **do**
  $L := U$; $\boldsymbol{t}_L := \boldsymbol{t}$; atLeastOnce:=TRUE ; $U := 2U$
  (possible, $\boldsymbol{t}$) := solve_MFGC$(\boldsymbol{p_{max}}, U\boldsymbol{f}, \boldsymbol{\mu^c}, \boldsymbol{\mu^r}, \boldsymbol{b}, C)$
end while /* it makes sure that $U > \lambda_{max}^T$ */
repeat
  $\lambda := (L + U)/2$
  (possible, $\boldsymbol{t}$) := solve_MFGC$(\boldsymbol{p_{max}}, \lambda\boldsymbol{f}, \boldsymbol{\mu^c}, \boldsymbol{\mu^r}, \boldsymbol{b}, C)$
  if possible **then** $L := \lambda$; $\boldsymbol{t}_L := \boldsymbol{t}$; atLeastOnce:=TRUE;
  else $U := \lambda$
until $(U - L)/L \leq \varepsilon_1$ AND atLeastOnce
$\lambda_{max}^T := L$; $\boldsymbol{t} := \boldsymbol{t}_L$

Procedure:

(possible,$\boldsymbol{t}$)=solveMFGC$(\boldsymbol{p_{max}}, \boldsymbol{\lambda_n}, \boldsymbol{\mu^c}, \boldsymbol{\mu^r}, \boldsymbol{b}, C)$

$\varepsilon_2 :=<$ precision $>$; $\delta :=<$ small value $>$
$\boldsymbol{t} := (\delta, \delta, \ldots, \delta)$
$\boldsymbol{p} := \text{MFGC}(\boldsymbol{t}, \boldsymbol{\lambda_n}, \boldsymbol{\mu^c}, \boldsymbol{\mu^r}, \boldsymbol{b}, C)$
repeat
  canConverge:=TRUE; $i := 1$;
  repeat
    if $\boldsymbol{p}(i) > \boldsymbol{p_{max}}(i)$ **then**
      $\boldsymbol{t}' := \boldsymbol{t}; \boldsymbol{t}'(i) := C$
      $\boldsymbol{p}' := \text{MFGC}(\boldsymbol{t}', \boldsymbol{\lambda_n}, \boldsymbol{\mu^c}, \boldsymbol{\mu^r}, \boldsymbol{b}, C)$
      if $\boldsymbol{p}'(i) > \boldsymbol{p_{max}}(i)$ **then**
        canConvege:=FALSE;
      else
        $L := \boldsymbol{t}(i); U := C$
        repeat
          $\boldsymbol{t}(i) := (L + U)/2$
          $\boldsymbol{p} := \text{MFGC}(\boldsymbol{t}, \boldsymbol{\lambda_n}, \boldsymbol{\mu^c}, \boldsymbol{\mu^r}, \boldsymbol{b}, C)$
          if $\boldsymbol{p}(i) > \boldsymbol{p_{max}}(i)$ **then** $L := \boldsymbol{t}(i)$
          else $U := \boldsymbol{t}(i)$
        until $(1 - \varepsilon_2)\boldsymbol{p_{max}}(i) \leq \boldsymbol{p}(i) \leq \boldsymbol{p_{max}}(i)$
      end if
    end if
    $i := i + 1$
  until $(i > 2N)$ OR (NOT(canConverge))
  if canConverge **then**
    if $\boldsymbol{p}(i) \leq \boldsymbol{p_{max}}(i) \quad \forall i$ **then**
      possible:=TRUE; exit:=TRUE;
    else exit:=FALSE;
  else possible:=FALSE; exit:=TRUE;
until exit

The procedure `MFGC`, which is invoked in the innermost loop of our algorithm, is used to obtain the blocking probabilities ($\boldsymbol{p} := \text{MFGC}(\boldsymbol{t}, \boldsymbol{\lambda_n}, \boldsymbol{\mu^c}, \boldsymbol{\mu^r}, \boldsymbol{b}, C)$). 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 $\boldsymbol{t}$ so that $\boldsymbol{p} = \boldsymbol{p_{max}}$ (within tolerance limit). Thus, in order to compute λ_i^h we use the expression $\lambda_i^h = \lambda_i^n(1 - B_i^n)/(\mu_i^c/\mu_i^r + B_i^h)$ in which λ_i^h 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 $\boldsymbol{t}$ is the policy setting for which the maximum capacity is achieved, the optimal prioritization order is the permutation $\boldsymbol{\sigma}^* \in \boldsymbol{\Sigma}$, $\boldsymbol{\Sigma} := \{(\sigma_i, \ldots, \sigma_{2N}) : \ \sigma_i \in \mathbb{N}, 1 \leq \sigma_i \leq 2N\}$, such that $t(\sigma_1^*) \leq t(\sigma_2^*) \leq \ldots \leq t(\sigma_{2N}^*) = C$. 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 converged, another prioritization order is tried.

Our algorithm does not require any *a priori* knowledge. Indeed, after obtaining the policy setting $\boldsymbol{t}$ 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

Table 1 Comparison of the HCO algorithm (with known prioritization order) and our algorithm (figures in Mflops)

		our algorithm	
C	HCO	—	speed-up
5	5.70	1.17	0.39
10	60.20	13.80	4.53
20	438.00	145.00	46.60

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 $\boldsymbol{b} = (1, 2)$, $\boldsymbol{f} = (0.8, 0.2)$, $\boldsymbol{\mu^c} = (1/180, 1/300)$, $\boldsymbol{\mu^r} = (1/900, 1/1000)$, $\boldsymbol{B}^n = (0.02, 0.02)$, $\boldsymbol{B}^{ft} = (0.002, 0.002)$; all tolerances have been set to $\varepsilon_{1,2} = 10^{-2}$.

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 $\boldsymbol{t}$.

5. Conclusions

We proposed a new algorithm for computing the optimal configuration of the of 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

[1] H. Heredia-Ureta, F.A. Cruz-Pérez, and L. Ortigoza-Guerrero, "Multiple fractional channel reservation for optimum system capcity in multi-service cellular networks," Electron. Lett., vol.39, no.1, pp.133–134, Jan. 2003.

[2] H. Heredia-Ureta, F.A. Cruz-Pérez, and L. Ortigoza-Guerrero, "Capacity optimization in multiservice mobile wireless networks with multiple fractional channel reservation," IEEE Trans. Veh. Technol., vol.52, no.6, pp.1519 – 1539, Nov. 2003.

[3] Y.B. Lin, S. Mohan, and A. Noerpel, "Queueing priority channel assignment strategies for PCS hand-off and initial access," IEEE Trans. Veh. Technol., vol.43, no.3, pp.704–712, Aug. 1994.