A Recursive Algorithm for Bandwidth Partitioning

Scott Jordan, *Member, IEEE,* Sam Charrington, and Pruttipong Apivatanagul

***Abstract*—We consider complete partitioning of bandwidth among multiple services. When class bandwidth is an integer multiple of the next lower class and total bandwidth is an integer multiple of the largest class bandwidth, we develop a recursive algorithm that determines the optimal complete partitioning policy with a significantly lower complexity than that of known dynamic programming or mixed integer programming approaches.**

***Index Terms*—Resource management, access control.**

I. Introduction

THE problem of allocation of a network resource to multiple services arises in many areas of networking. Often, multiple services share a single resource. Allocation of this resource is generally determined by a combination of connection access control and packet scheduling policies. Although cross-layer approaches may achieve superior results, in connection-oriented packet-switched networks connection access control and packet scheduling are often segmented by analyzing them on different time scales in order to maintain modularity, see e.g. [1], [2]. In this segmented approach, packet scheduling is considered on the packet time scale, and each connection's resource demands are described by averaging across the connection duration. Connection access control uses these averaged resource requirements to allocate a shared resource among multiple service classes. Multiplexing gains are thus modeled separately on packet and connection time scales, and packet level QoS and connection level QoS are treated separately.

Such a segmentation reduces the connection access control problem to a classical problem of the sharing of bandwidth on a single link in a circuit-switched network that supports multiple service classes defined by the bandwidth required per connection, see e.g. [1]. This classical problem also appears in a wavelength division multiplexed network, see e.g. [3]. The two simplest admission control policies are *complete sharing* (in which all connections share the entire link bandwidth and a connection is admitted if there is sufficient free bandwidth on the link) and *complete partitioning* (in which the link bandwidth is partitioned among classes and a connection is admitted if there is sufficient free bandwidth in that class's allocation). Intermediate policies include trunk reservation (in which a reservation parameter is set for each class and a connection is admitted if the free bandwidth on the link exceeds the reservation parameter) and coordinate convex policies (in which a connection is admitted if the resulting vector of the number of active connections in each class remains within a specified convex subset of the state space), see e.g. [4]. Although trunk reservation and coordinate convex policies can usually outperform complete sharing and complete partitioning policies, complete partitioning is often used for simplicity. Complete partitioning policies continue to arise in many areas in networking, see e.g. [5] for use in virtual private networks (VPNs), [6], [7] for use in Multiple Packet Label Switching (MPLS) and/or Integrated Services (IntServ), [8]–[10] for use in wireless networks, and [11] for use in application composition.

In this paper, we consider connection access control policies that use complete partitioning of a single resource, here considered to be bandwidth. We presume that the system attempts to either maximize the revenue generated by admitted connections or to minimize a weighted sum of blocking probabilities. Although these optimization metrics are the most commonly considered in this problem, we note that other metrics and constraints are sometimes considered, see e.g. [12]. In addition, often connection access control policies consider allocation of multiple resources, see e.g. [1], [4]. Complete partitioning can be used as a static resource allocation strategy, in which case the computational complexity may not be an issue. However, complete partitioning is often considered as a dynamic resource allocation strategy, by recomputing the optimal complete partition whenever the traffic load changes significantly [12]. In this case, the time required to compute the optimal policy can be critical. A dynamic programming approach to the determination of the optimal solution was provided in [13], and a mixed integer programming approach was provided in [14]. Let K denote the number of classes and m denote the number of resources. The complexity of the dynamic programming approach is $O(Km^2)$. The complexity of the mixed integer programming approach has not been analytically determined; numerical evaluations show that it is lower than that of the dynamic programming approach, but that it can still be unacceptably high for systems with a large number of shared resources.

We consider complete partitioning in the case in which the resource demand of a class is an integer multiple of the demand of the next lower class. This integer relationship holds in most systems in which complete partitioning is used. In addition, we assume that the total bandwidth is an integer multiple of the largest class bandwidth. Under these assumptions, we develop a recursive algorithm that determines the optimal complete partitioning policy with a

Paper approved by T. T. Lee, the Editor for Switching Architecture Performance of the IEEE Communications Society. Manuscript received March 13, 2009; revised August 18, 2009.

S. Jordan is with the Department of Computer Science, University of California, Irvine, CA 92697 (e-mail: sjordan@uci.edu).

S. Charrington contributed to this project while he was with the Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL 60208.

P. Apivatanagul contributed to this project while he was with the Institute of Transportation Studies, University of Calfornia, Irvine, CA 92697.

Additional work on this project was provided by Jie Hu.

Digital Object Identifier 10.1109/TCOMM.2010.04.090041

complexity that is $O(Km)$, namely $O(m)$ less than that of the dynamic programming approach. In addition, through numerical evaluations we show that the complexity can be a few orders of magnitude lower than that of the mixed integer programming approach. It should noted, however, that the dynamic programming and mixed integer programming approaches can model systems in which the integer multiple assumptions do not hold. The partitioning problem is presented in section II, the recursive algorithm is developed in section III, and its complexity is analyzed in section IV.

II. The Partitioning Problem

Consider a single link of bandwidth m shared between K classes of service. Connections of class i arrive as a Poisson process with rate λ_i, independent of arrivals of other classes. Each arrival, if admitted to the link, occupies a bandwidth of ν_i for an Exponentially distributed length of time with rate μ_i, independent of the duration of other connections. Suppose that each admitted connection of class i generates a revenue r_i per unit time in the system. Classes are thus characterized by the combination of resource demand, mean duration, and revenue rate. Denote the set of class bandwidths by $\vec{\nu} = (\nu_1, \cdots, \nu_K)$ and the set of class loads by $\vec{\rho} = (\rho_1, \cdots, \rho_K)$ where $\rho_i = \lambda_i/\mu_i$.

We consider the class of resource allocation policies known as *complete partitioning*, in which the bandwidth m is partitioned among the K classes, and connections of class i are admitted if and only if there is sufficient bandwidth within the bandwidth dedicated to class i. Denote by y_i the bandwidth dedicated to class i, where $\sum_{i=1}^{K} y_i \leq m$. The maximum number of simultaneous class i connections is thus $n_i = y_i/\nu_i$. Denote a complete partitioning policy by $\vec{n} = (n_1, \cdots, n_K) \in \mathbb{Z}^{+K}$.

Assume that classes are ordered by their bandwidths so that $\nu_i > \nu_{i-1}\ \forall i > 1$. Classes are usually defined so that the ratios of adjacent class bandwidths, denoted by $a_i = \nu_i/\nu_{i-1} > 1\ \forall i$, are integers, and we consider this case here. Normalizing the units of bandwidth so that $\nu_1 = 1$, we have $\nu_i = \prod_{j=2}^{i} a_j$. In addition, we assume that the total bandwidth m is an integer multiple c of the largest class bandwidth, i.e. $m = c\nu_K$.

Under complete partitioning, class i acts as multiserver loss queue and the blocking probability of class i is thus given by

$$PB_i(n_i) = \frac{\rho_i^{n_i}/n_i!}{\sum_{l=0}^{n_i} \rho_l^{n_l}/n_l!}. \tag{1}$$

The total revenue per unit time earned by the link under complete partitioning policy $\vec{n}$ is thus

$$R(\vec{n}) = \sum_{i=1}^{K} r_i \lambda_i [1 - PB_i(n_i)]. \tag{2}$$

The complete partitioning problem can then be stated as

$$\max_{\vec{n}} R(\vec{n}) \text{ s.t. } \vec{n} \cdot \vec{\nu} \leq m.$$

Using the substitution for $R(\vec{n})$ in (2), the objective can be equivalently stated as minimizing a weighted sum of blocking probabilities, $\min_{\vec{n}} \sum_{i=1}^{K} \omega_i PB_i(n_i)$, where $\omega_i = r_i\lambda_i$ is the weight for class i. In this form, the problem was given in [14] as *Problem APUT*. This problem can be transformed into a mixed integer programming problem by treating $\vec{n}$ as a set of integer decision variables. (Alternate problems are often considered that place constraints on blocking probabilities.)

III. A Recursive Algorithm

A dynamic programming approach to the determination of the optimal solution was provided in [13], and a mixed integer programming approach was provided in [14]. While these techniques can be applied to classes with arbitrary bandwidth usage, we consider here only the subset of the complete partitioning problem in which class bandwidth is an integer multiple of the next lower class, i.e. $a_i \in \mathbb{Z}^+\ \forall i$, and total bandwidth is an integer multiple of the higher class bandwidth, i.e. $m = c\nu_K$. We will propose a algorithm that outperforms both previous algorithms for this restricted problem.

The algorithm takes advantage of the concavity of the revenue function with respect to the bandwidth assigned to each class.

Lemma 1: $R(\vec{n})$ is concave in $n_i\ \forall i$.

Proof: The Erlang blocking probability $PB_i(n_i)$ is decreasing and convex in n_i [15]. From (2), it follows that $R(\vec{n})$ is concave in $n_i\ \forall i$. ■

This lemma enables us to construct a recursive resource assignment algorithm. We note that bandwidth can be assigned in increments of the least common multiple of the set of class bandwidths $\{\nu_i\}$. In the case in which the class bandwidth ratios are integers, i.e. $a_i \in \mathbb{Z}^+\ \forall i$, this least common multiple is ν_K, and hence each set of ν_K bandwidth can be assigned to the combination of classes that will generate the highest incremental revenue. The concavity property dictates that this assignment need not be revisited when further bandwidth is assigned.

Next we note that each set of of ν_K bandwidth can either be assigned as a group to class K or can be optimally split among the remaining $K-1$ classes. In the latter case, we can consider this ν_K bandwidth in a_K subsets of ν_{K-1} each, with each subset optimally assigned according to the incremental revenue it will generate.

Similarly, each subset of ν_{K-1} bandwidth can be assigned as a group to class $K-1$ or can be optimally split among the remaining $K-2$ classes. This process can be continued by splitting each set into a_i subsets until the branching ends when class 1 is considered. The resulting exploration of potential assignments can be represented as a tree. In Figure 1, we display such a tree for a problem in which $K = 3$, $\vec{\nu} = (1, 2, 4)$, and $m = 4$; each box represents one of the 4 units of bandwidth, and the tree shows the sequence of incremental allocations considered.

Decisions are made on the way back up the decision tree. A bandwidth of ν_2 is tentatively assigned either to class 1 or to class 2, depending on which generates the higher incremental revenue. A bandwidth of ν_3 is tentatively assigned either to class 3 or to the combination of classes 1 and 2 (as dictated by the previous tentative decision). These tentative decisions continue until the final decision is made when choosing between assignment of ν_K bandwidth either to class K or to the optimal combination of classes 1 through $K-1$. This recursive process optimally assigns ν_K bandwidth. The

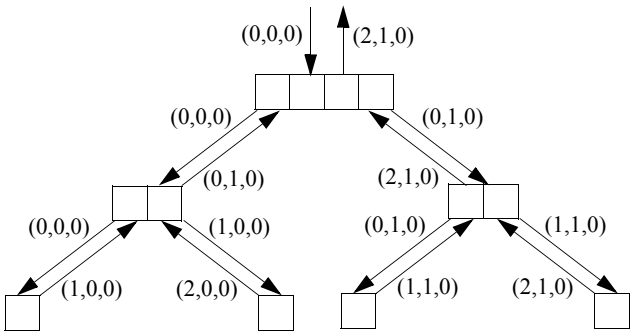

Fig. 1. Decision tree for $K = 3$, $\vec{\nu} = (1, 2, 4)$, and $m = 4$.

algorithm repeats this recursive process c times, resulting in optimal allocation of the entire m bandwidth.

In Figure 1, the first two units of bandwidth are tentatively assigned to class 1. This tentative decision ($\vec{n} = (2, 0, 0)$) is passed up to the second level of the tree, where the decision is to reassign these two units of bandwidth to class 2 since this alternative allocation generates higher revenue. This tentative decision ($\vec{n} = (0, 1, 0)$) is passed up to the first level of the tree and down to the right two lowest leaves, where the second two units of bandwidth are then tentatively assigned to class 1. This tentative decision ($\vec{n} = (2, 1, 0)$) is passed up to the second level of the tree, where it is compared to the alternative allocation $\vec{n} = (0, 2, 0)$; the former allocation generates higher revenue, and this tentative decision ($\vec{n} = (2, 1, 0)$) is passed up to the root of the tree. A final comparison is made to the policy $\vec{n} = (0, 0, 1)$; since the former allocation is superior, it is the final allocation and the algorithm terminates.

The algorithm is described by the pseudo-code in Table I and flowcharted in Figure 2. The *Main* routine calls *Initialize* once, and then repeatedly calls a recursive routine *Allocate* to assign each set of ν_K bandwidth. The *Initialize* routine calculates and stores all potential blocking probabilities $PB_i(n_i)$ according to (1). The *Allocate* routine implements a recursive allocation for a set (or subset) of bandwidth. It recursively creates a tree of allocation options and then chooses the best of these options on the way back up the tree. At each recursion level, the algorithm asks the question "Should I assign a bandwidth of ν_i as a whole to class i or should I split up this bandwidth and distribute optimally to classes 1 through $i - 1$?". (We define the indicator vector $\vec{e_i} = (e_1, \cdots, e_K)$ where $e_k = 1$ if $i = k$ and $e_k = 0$ otherwise, and use it to denote the allocation of additional bandwidth ν_i to class i.) Decisions are made on the basis of which option generates the higher incremental revenue.

Convex optimization theory tells us that the algorithm is optimal:

Theorem 1: The proposed algorithm finds an allocation that generates the optimal revenue.

Proof: It is well known that the maximization of a separable concave function subject to a sum constraint of the integer decision variables can be solved by a greedy algorithm, see e.g. [16]. Define integer variables b_0 and b_1, and let $b_1 = n_K$ denote the number of sets of ν_k bandwidth assigned to class K and $b_0 = (m/\nu_K - b_1)$ denote the number of sets of ν_k bandwidth assigned to classes $1 : K - 1$. Since the revenue function can be written as a separable and concave function of b_0 and b_1, it follows that a greedy algorithm that incrementally assigns each set of ν_k bandwidth to either class K or to classes $1 : K - 1$ will achieve the optimal allocation. Similar reasoning can be used to conclude that at each level of the recursion, the binary decision to allocate

TABLE I
PSEUDOCODE FOR RECURSIVE ALGORITHM

***Main*:**
Initialize;
for (j in 1:c) {
/* Allocate batch of ν_K bandwidth */
$\vec{n}$ = Allocate($\vec{n}$,K); }

***Initialize*:**
compute $PB_i(y_i)\ \forall\ n_i = 1 : (m/\nu_i)$ according to (1);
$\vec{n} = \vec{0}$;

***Allocate*($\vec{n}$,i):**
/* Determine optimal allocation of bandwidth ν_i to classes $1 : i$ and return corresponding $\vec{n}$ */
if ($i > 1$) {
$\vec{n'} = \vec{n}$;
for (j in 1:a_k) {
/* Determine optimal allocation of bandwidth ν_{i-1} to classes $1 : i - 1$ */
$\vec{n'}$ = Allocate($\vec{n'}$,i-1); }
if ($R(\vec{n} + \vec{e_i}) > R(\vec{n'})$ {
/* Assign bandwidth ν_i to class i */
return $\vec{n} + \vec{e_i}$; }
else {
/* Assign bandwidth ν_i to classes $1 : i - 1$ */
return $\vec{n'}$; } }
else return $\vec{n} + \vec{e_1}$; }

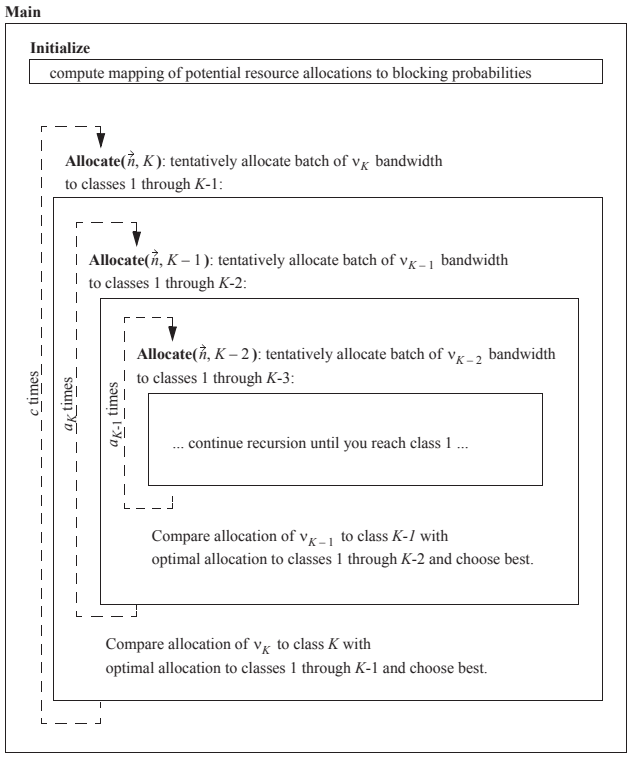

Fig. 2. Flowchart of recursive algorithm.

incremental bandwidth to either class i or to classes $1:i-1$ can be optimally achieved by a greedy algorithm. The theorem follows. ■

IV. Complexity

In this section, we compare the computational complexity of the proposed algorithm to that of the dynamic programming approach given in [13] and of the mixed integer approach given in [14].

The execution time of the *Initialize* routine is approximately c_1+c_2Km, where c_1 is the constant time required for various bookkeeping tasks and c_2Km is the time required for calculation of the blocking probabilities for every class and for every potential allocation of resources to that class. The execution time of the *Allocate* routine is more difficult to obtain. When called for i classes, denote the execution time by $z(i)$. Most of the time in the routine consists of a_i calls to *Allocate* for $i-1$ classes, each of which requires $z(i-1)$ time. It follows that $z(i)=a_iz(i-1)+c_3$, when $i>1$, where c_3 is the time for all statements outside the recursion. Expanding the recursion, it follows that $z(K)\approx\sum_{i=1}^{K}\nu_ic_3+\nu_Kc_4$, where c_4 is the time for the base case execution of the *Allocate* routine. The loop in the *Main* routine calls *Initialize* once and calls *Allocate(K)* c times, so the execution time of the entire algorithm is thus approximately $c_1+c_2Km+c[\sum_{i=1}^{K}\nu_ic_3+\nu_Kc_4]$. Expressing $c=m/\nu_K$, this gives an execution time of approximately $c_1+c_2Km+(\sum_{i=1}^{K}\frac{\nu_i}{\nu_K})c_3m+c_4m=O(Km)$.

The execution time of the dynamic programming approach is $O(Km^2)$ [13], which is $O(m)$ worse than the recursive algorithm. The mixed integer approach is presented in [14] as *Problem APUT-MIP*. It introduces decision variables x_{ij} that dictate the allocation of $x_{ij}j$ bandwidth to class i. The mixed integer approach can be used to find either the optimal or a suboptimal allocation. We focus here on the optimal allocation. In that case, the problem is expressed as:

$$\min_{\vec{n}}\sum_{i=1}^{K}\sum_{j=0}^{m/\nu_i}\omega_iPB_i(j)x_{ij}$$
$$\text{s.t.}\sum_{j=0}^{m/\nu_i}x_{ij}=1,\sum_{j=0}^{m/\nu_i}jx_{ij}=n_i,$$
$$\vec{n}\cdot\vec{\nu}\le m,x_{ij}\ge 0,\forall i,j.$$

This mixed integer formulation has K integer and $K+\sum_{i=1}^{K}m/\nu_i$ real variables. Its complexity has not been analytically determined. Therefore, we compare the execution time of our proposed recursive algorithm to that of the MIP algorithm in two experiments. In both, we set $a_i=2\ \forall i>1,\lambda_i=2^{K-i}\lambda\ \forall i,\mu_i=1\ \forall i,r_i=\nu_i\ \forall i$, and λ is chosen so that the total normalized load $\vec{\rho}\cdot\vec{\nu}=0.8$. The recursive algorithm is coded entirely in C, while the mixed integer program is coded in C with calls to the lp_solve 5.5 branch-and-bound software library [17]. Both were run on the same 2.40 GHz Windows XP PC with 2GB RAM.

In the first experiment, we fix the number of classes $K=14$ and vary the bandwidth m by varying c from 1 to 96. The execution times (in seconds) are given in Table II. For the recursive algorithm, with K fixed, from the above analysis we expect the execution time to approximately follow $c_1+[c_2K+2c_3-c_3/2^{K-1}+c_4]m$, namely it should be approximately linear in m. Indeed, the numerical data follows this form extremely well, with time $\approx 3.11\cdot10^{-6}\ m$ with a correlation of $R^2=0.99$. The mixed integer approach requires an execution time that is also approximately linear in m, but with time $\approx 1600\cdot10^{-6}\ m$. As a result, mixed integer execution time is about 70 times longer than the recursive algorithm's execution time when $c=1$ and rises to roughly 500 times longer as m increases[1].

TABLE II
Execution Time Versus Bandwidth for a Fixed Number of Classes

m	Recursive execution time (secs)	MIP execution time (secs)
8192	0.03	2.50
16384	0.06	5.96
24576	0.08	11.60
32768	0.09	22.02
40960	0.11	23.46
49152	0.13	32.04
57344	0.18	45.18
65536	0.18	51.09
131072	0.21	155.58
196608	0.41	263.62
262144	0.62	389.5
327680	1.04	519.31
393216	1.25	600.97
458752	1.45	695.55
524288	1.66	875.21
655360	2.07	1029.71
786432	2.51	1328.63

TABLE III
Execution Time Versus Number of Classes for a Fixed Bandwidth

K	Recursive execution time (secs)	MIP execution time (secs)
2	0.13	0.84
3	0.15	1.03
4	0.18	1.17
5	0.17	1.37
6	0.18	1.53
7	0.18	6.79
8	0.19	19.69
9	0.19	33.49
10	0.20	45.37
11	0.19	53.89
12	0.20	57.39
13	0.18	57.77
14	0.20	51.40

In the second experiment, we fix the bandwidth $m=2^{16}$ and vary the number of classes K from 2 to 14. The execution times are given in Table III. For the recursive algorithm, with m fixed, from the above analysis we expect the execution time to approximately follow $[c_1+(2c_3+c_4)m]+(c_2m)K-c_3m/2^{K-1}$. The numerical data follows this form extremely well, with time $\approx 0.177+0.0013K-0.213/2^K$ with a correlation of $R^2=0.96$. The mixed integer approach's execution time varies widely with the number of classes K, as we expect since the mixed integer formulation has K integer and $K+\sum_{i=1}^{K}m/\nu_i$ real variables. The mixed integer execution time is about 7 times longer than the recursive algorithm's execution time at low values of K and rises to roughly 300 times longer as K increases.

[1]The MIP's execution time can be decreased by searching for sub-optimal solutions, but in [14] this only resulted in reduction by a factor of 2-3 for a 5% tolerance.

V. Conclusion

Complete partitioning of bandwidth among multiple services is a problem that arises in many types of networks. When complete partitioning is used as a dynamic resource allocation strategy, by recomputing the optimal complete partition whenever the traffic load changes significantly, the time required to compute the optimal policy can be critical. In this paper, we developed a recursive algorithm that can be used in the common case when class bandwidth is an integer multiple of the next lower class and when total bandwidth is an integer multiple of the largest class bandwidth. We derived the complexity of this algorithm, and showed in numerical comparisons that it is significantly lower than that of known dynamic programming or mixed integer programming approaches. It would be of value to consider whether similar recursive techniques could be applied to complete partitioning of *multiple* resources among multiple services, such as that which occurs when bandwidth is simultaneously allocated on multiple intersecting routes.

References

[1] K. W. Ross, *Multiservice Loss Models for Broadband Telecommunication Networks*. Springer-Verlag, 1995.

[2] A. Arvidsson, J. de Kock, A. Krzesinski, and P. Taylor, "Cost-effective deployment of bandwidth partitioning in broadband networks," *Telecommun. Syst.*, vol. 25, pp. 33-49, 2004.

[3] K. Mosharaf, J. Talim, and I. Lambadaris, "Optimal resource allocation and fairness control in all-optical WDM networks," *IEEE J. Sel. Areas Commun.*, vol. 23, no. 8, pp. 1496-1507, Aug. 2005.

[4] S. Jordan and P. P. Varaiya, "Control of multiple service, multiple resource communication networks," *IEEE Trans. Commun.*, vol. 42, no. 11, pp. 2979-2988, Nov. 1994.

[5] D. Mitra, J. Morrison, and K. Ramakrishnan, "Virtual private networks: joint resource allocation and routing design," in *Proc 18th Annual Joint Conf. IEEE Computer Commun. Societies (Infocom 1999).*, vol. 2, pp. 480-490, Mar. 1999.

[6] V. Rakocevic, J. Griffiths, and G. Cope, "Dynamic partitioning of link bandwidth in IP/MPLS networks," in *Proc. IEEE International Conf. Commun. (ICC 2001)*, vol. 9, pp. 2918-2922.

[7] A. Elwalid, D. Mitra, I. Saniee, and I. Widjaja, "Routing and protection in GMPLS networks: from shortest paths to optimized designs," *J. Lightwave Technol.*, vol. 21, no. 11, pp. 2828-2838, Nov. 2003.

[8] S. Jordan and E. Schwabe, "Worst-case performance of cellular channel assignment policies," *ACM J. Wireless Netw.*, vol. 2, pp. 265-275, 1996.

[9] D. Niyato and E. Hossain, "A queuing-theoretic and optimization-based model for radio resource management in IEEE 802.16 broadband wireless networks," *IEEE Trans. Computers*, vol. 55, no. 11, pp. 1473-1488, Nov. 2006.

[10] J.-G. Choi, Y.-J. Choi, and S. Bahk, "Power-based admission control for multiclass calls in QoS-sensitive CDMA networks," *IEEE Trans. Wireless Commun.*, vol. 6, no. 2, pp. 468-472, Feb. 2007.

[11] H. Bannazadeh and A. Leon-Garcia, "Allocating services to applications using Markov decision processes," in *Proc. IEEE International Conf. Service-Oriented Computing Appl.*, June 2007, pp. 141-146.

[12] R. Bolla, F. Davoli, and M. Marchese, "Bandwidth allocation and admission control in ATM networks with service separation," *IEEE Commun. Mag.*, vol. 35, no. 5, pp. 130-137, May 1997.

[13] K. Ross and D. Tsang, "The stochastic knapsack problem," *IEEE Trans. Commun.*, vol. 37, no. 7, pp. 740-747, July 1989.

[14] G. Meempat and M. Sundareshan, "Optimal channel allocation policies for access control of circuit-switched traffic in ISDN environments," *IEEE Trans. Commun.*, vol. 41, no. 2, pp. 338-350, Feb. 1993.

[15] E. Messerli, "Proof of a convexity property of the Erlang B formula," *Bell Syst. Tech. J.*, vol. 51, pp. 951-953, 1972.

[16] D. G. Luenberger, *Linear and Nonlinear Programming*. Springer, 2008.

[17] *lp_solve, A mixed integer programming solver*. [Online]. Available: http://lpsolve.sourceforge.net/5.5/ under the GNU public license.