Call Admission Control and Routing in Integrated Services Networks Using Neuro-Dynamic Programming

Peter Marbach, Oliver Mihatsch, and John N. Tsitsiklis

***Abstract*—We consider the problem of call admission control (CAC) and routing in an integrated services network that handles several classes of calls of different value and with different resource requirements. The problem of maximizing the average value of admitted calls per unit time (or of revenue maximization) is naturally formulated as a dynamic programming problem, but is too complex to allow for an exact solution. We use methods of neuro-dynamic programming (NDP) [reinforcement learning (RL)], together with a decomposition approach, to construct dynamic (state-dependent) call admission control and routing policies. These policies are based on state-dependent link costs, and a simulation-based learning method is employed to tune the parameters that define these link costs. A broad set of experiments shows the robustness of our policy and compares its performance with a commonly used heuristic.**

***Index Terms*—ART neural networks, communication system control, communication system routing, dynamic programming, Markov processes.**

I. Introduction

WE CONSIDER a communication network consisting of a set of nodes $\mathcal{N} = \{1, \ldots, N\}$ and a set of unidirectional links $\mathcal{L} = \{1, \ldots, L\}$, where each link l has a total capacity of $B(l)$ units of bandwidth. There is a set $\mathcal{M} = \{1, \ldots, M\}$ of different service classes, where each class m is characterized by its bandwidth requirement $b(m)$, its average call holding time $1/\nu(m)$, and the immediate reward (or value) $c(m)$ obtained whenever such a call is accepted. The bandwidth requirement $b(m)$ may reflect either the peak transmission rate requested by class m calls, or their "effective bandwidth" as defined and extensively studied in the context of ATM networks [25]. Furthermore, the reward $c(m)$ is not necessary a monetary one, but may reflect the importance of different classes and their desired quality-of-service (QoS) (blocking probabilities). We assume that the calls arrive according to independent Poisson processes with known rates $\lambda_{ij}(m)$ for class m calls with origin $i \in \mathcal{N}$ and destination $j \in \mathcal{N}$. We also assume that the holding times of the calls are independent, exponentially distributed, with finite mean $1/\nu(m)$, $m = 1, \ldots, M$ and independent of the arrival processes.

When a new call of class m, with origin i and destination j arrives, it can be either rejected (with zero reward) or it can be admitted (with reward $c(m)$). In order to accept it, we need to choose a route out of a predefined list of possible routes from i to j. Furthermore, at the time that the call is accepted, each link along the chosen route must have at least $b(m)$ units of unoccupied bandwidth. The objective is to exercise call admission control (CAC) and routing in such a way that the long term average reward is maximized. Ideally, this maximization should take place within the most general class of state-dependent policies, whereby the admission decision and the route choice are allowed to depend on the current state of the network.

The CAC defined earlier and routing problem have been studied extensively; see e.g., [10], [18] and the references therein. It is naturally formulated as an average reward dynamic programming problem, but is too complex to be solved exactly, and suitable approximations have to be employed to compute control policies. One proposed approach in this context is the reduced load approximation (also called Erlang fixed point method) [5], [10]. It relies on link independence and Poisson assumptions which allow to decompose the network into link processes where calls arrive according to independent Poisson processes. The corresponding arrival rates model the thinned (by blocking on other links) external traffic and are computed by iteratively solving a system of fixed point equations. This approach has been used to analyze routing schemes such as probabilistic routing (also called proportional routing) [5], [9], [16] and dynamic alternative routing with trunk reservation [7], [11], [12]. As its name suggests, state independent probabilistic routing assigns routes to calls at random according to a given probability distribution. Using the concept of a state independent link cost (link shadow price), gradient methods for tuning the routing probabilities can be devised [5], [9], [16]. Probabilistic routing can be shown to be asymptotically optimal, however in a "coarse sense": optimal routing schemes are sensitive to the model parameters, i.e., small modeling errors can severely degrade performance [26]. More robust, but also more difficult to analyze and optimize, is the state-dependent dynamic alternative routing with trunk reservation. In the case of a single service class, a decomposition

Manuscript received February 15, 1999; revised August 1, 1999. This research was supported in part by Siemens AG, Germany, Alcatel Bell, Belgium, and by the NSF under contract ECS-9873451. A preliminary version of this paper was presented at the 37th IEEE Conference on Decision and Control, Tampa, FL, December 1998.

P. Marbach was with the Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA 02139 USA. He is now with the Centre for Communication Systems Research, Cambridge University, Cambridge CB2 3O5, U.K. (e-mail: p.marbach@ccsr.cam.ac.uk).

O. Mihatsch is with the Siemens AG, Corporate Technology, Information and Communications 4, Munich D-81730, Germany (e-mail: oliver.mihatsch@mchp.siemens.de).

J. N. Tsitsiklis is with the Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA 02139 USA (email: jnt@mit.edu).

Publisher Item Identifier S 0733-8716(00)00507-2.

approach, that splits the reward associated with a call into link rewards, can be employed to compute state-dependent link costs (shadow prices) and to tune the trunk reservation parameters [11]. However, in the case of multiple service classes, a judicious choice of the trunk reservation parameters, that lead to near optimal performance, can be difficult. An application of this approach is described in [13], [14], for a relatively small problem, but can easily become intractable for larger networks. A variant of this approach was proposed which uses measurements in the network to determine the arrival rates associated with each link, thus avoiding the computational burden of solving fixed point equations. Similar to [11], a decomposition approach of the call rewards can be employed to compute state-dependent link costs and to optimize the policy. This method can again become intractable unless further approximations, such as link state aggregations, are employed. An application of this approach is given in [6].

The link independence and Poisson assumptions play an important role in the methods described earlier and allow one to construct a simpler model of the network process and to compute implied link costs (shadow prices). These costs are then used to obtain an approximation of the true implied network costs (derived from the differential reward function of dynamic programming) and to optimize and implement a CAC and routing policy. In this paper, we develop a new approach which allows us to avoid the use of a reduced model, i.e., explicitly decomposing the network process into independent link processes. We start with a dynamic programming formulation (Section II) and then use simulation based approximate dynamic programming [also called reinforcement learning (RL) or neuro-dynamic programming (NDP)] [2], [21] to construct an approximate differential reward function and to optimize the policy (Section III). In the following, we will use the term NDP for simulation-based approximate dynamic programming. For these methods, performance guarantees exist only for special cases (see [2]); however, recent case studies illustrate their ability to successfully address large scale problems. In particular, they have been applied to resources allocation problems in telecommunication systems, such as the channel assignment problem in cellular telephone systems [19], the link allocation problem [17], and the single link admission control problem with self similar traffic [4] or with statistical quality of service constraints [3]. A successful application of NDP relies crucially on the choice of a suitable (parametric) architecture for the approximation of the differential reward function: it should be rich enough (i.e., involve enough parameters) to approximate closely the differential reward function, but also simple (i.e., involve not too many parameters) to limit the "training time" to obtain a good approximation. Typically, an approximation architecture is chosen by a combination of analysis, engineering insight, and trial and error. Motivated by the analysis carried out in connection with the reduced load approach and its variants, we rely on a function which depends quadratically on the number of active calls of each class on each link and which leads to policies that rely on "trained" state-dependent link costs. Furthermore, we decompose the call reward into link rewards to allow a decentralized implementation of the optimization method and the resulting policies. We apply this approach to a large network involving 62 links and with 992 tunable parameters in our differential reward function approximator. To assess the method, we compare our CAC and routing policies with the "open shortest path first" (OSPF) heuristic (Section IV-B and Section IV-C). We show that the performance of our NDP policy is very robust with respect to changing arrival statistics. To investigate the accuracy of the quadratic approximator, we also provide a case study involving a single link (Section IV-A).

The main contributions of the paper are the following.

a) We show that NDP can be applied to the CAC problem in a manner that supports decentralized training and decentralized decision making.

b) By using NDP, we are able to avoid the use of a reduced model, as it was introduced in previous approaches through the link independence and Poisson assumption.

c) We avoid the computational burden associated with the evaluation of the link reward functions, as it was encountered in [6], [11].

II. Dynamic Programming Formulation

We will now formulate the problem of CAC and routing as a continuous time, average reward, finite-state dynamic programming problem [1]. For any time t, let $n_t(r, m)$ be the number of class m calls that are currently active (have been admitted and have not yet terminated) and which have been routed along route r. The state x_t of the network at time t consists of a list of the numbers $n_t(r, m)$, for each r and m. The state space S (the set of all possible states) is defined implicitly by the requirements that each $n_t(r, m)$ be a nonnegative integer and that

$$\sum_{r \in R(l)} \sum_{m \in \mathcal{M}} n_t(r, m) b(m) \leq B(l), \qquad \forall l \in \mathcal{L}$$

where $R(l)$ is the set of routes that use link l. Even though the process evolves in continuous time, we only need to consider the state of the network at the times when certain events take place. The events of interest are the arrivals of new call requests and the terminations of existing calls. Note that the nature of an event is completely specified by the class m, origin-destination pair (i, j), and if it corresponds to a call termination, the route r occupied by the call. We denote by Ω the (finite) set of all possible events.

If the state of the system is x and event ω occurs, a decision u has to be made. If ω corresponds to an arrival, the set of possible decisions $U(\omega, x)$ consists of the possible routes (subject to the capacity constraints and the current state of the network) and of the rejection decision. If ω corresponds to a departure, there are no decisions to be made, which amounts to letting $U(x, \omega)$ be a singleton. Given the present state of the network x, an event ω, and a decision $u \in U(x, \omega)$, the network moves to a new state which will be denoted by x'. The resulting reward will be denoted by $g(x, \omega, u)$: if ω corresponds to a class m arrival and u is a decision to admit along some route, then $g(x, \omega, u) = c(m)$; otherwise, $g(x, \omega, u) = 0$.

We define a policy to be a mapping μ whose domain is the set $S \times \Omega$ and which satisfies

$$\mu(x, \omega) \in U(x, \omega), \qquad \forall x \in S, \omega \in \Omega.$$

We note that under any given policy μ, the state x_t evolves as a continuous time finite state Markov process. Let t_k be the time of the kth event, and let x_{t_k} be the state of the system just prior to that event. (This notation is equivalent to assuming that x_t is a left continuous function of time.) We then define the average reward associated with a policy μ to be

$$v(\mu) = \lim_{N \to \infty} \frac{1}{t_N} \sum_{k=0}^{N-1} g(x_{t_k}, \omega_k, u_{t_k}) \tag{1}$$

where $u_{t_k} = \mu(x_{t_k}, \omega_{t_k})$. Under the assumption that for all service classes the average call holding time is finite, the state corresponding to an empty system, to be denoted by $\hat{x}$, is recurrent. For this reason, the limit in (1) exists, is independent of the initial state and is equal to a deterministic constant with probability one.

A policy μ^* is said to be optimal if

$$v(\mu^*) \geq v(\mu)$$

for every other policy μ. We denote the average reward associated with an optimal policy μ^* as v^*.

An optimal policy can be obtained, in principle, by solving the Bellman optimality equation for average reward problems, which takes the form

$$\begin{aligned} &v^* E_\tau\{\tau | x\} + h^*(x) \\ &\quad = E_\omega\{\max_{u \in U(x,\omega)} [g(x, \omega, u) + h^*(x')]\}, \; x \in S \end{aligned} \tag{2}$$

$$h^*(\hat{x}) = 0. \tag{3}$$

Here, τ stands for the time until the next event occurs and $E_\tau\{\tau | x\}$ is the expectation of τ given that the current state is x. Furthermore, $E_\omega\{\cdot\}$ stands for the expectation with respect to the next event ω, and x' stands for the state right after the event, which is a deterministic function of x, ω, and the chosen decision u. If $|S|$ is the cardinality of the state space, the Bellman equation is a system of $|S| + 1$ nonlinear equations in the $|S| + 1$ unknowns $h^*(x)$, $x \in S$, and v^*. Because the state $\hat{x}$ is recurrent under every policy, the Bellman equation has a unique solution and the function $h^*(\cdot)$, called the optimal differential reward, admits the following interpretation: if we operate the system under an optimal policy, then $h^*(x) - h^*(y)$ is equal to the expectation of the difference of the total rewards (over the infinite horizon) for a system initialized at x, compared with a system initialized at y.

Once the optimal differential reward function $h^*(\cdot)$ is available, an optimal admission control and routing policy μ^* is given by

$$\mu^*(x, \omega) = \arg \max_{u \in U(x,\omega)} [g(x, \omega, u) + h^*(x')]. \tag{4}$$

This amounts to the following: whenever a new class m call requests a connection, consider admitting it along a permissible route, and let x' be the resulting successor state. We compute the value of such a decision by adding the immediate reward $g(x, \omega, u) = c(m)$ to the merit $h^*(x')$ of x'. We pick a route that results in the highest value and route the call accordingly if that value is higher than the value $h^*(x)$ of the current state; otherwise, the call is rejected.

However, the dynamic programming approach is impractical because the state space S is typically so large that it is impossible to compute, or even store, the optimal differential reward $h^*(x)$ for each state $x \in S$. This leads us to consider methods that work with approximations to the function h^*.

III. NDP SOLUTION

NDP is a simulation based approximate dynamic programming methodology for producing near optimal solutions to large scale dynamic programming problems. The central idea is to approximate v^* and the function $h^*(\cdot)$ by a tunable scalar $\tilde{v}$ and an approximating function $\tilde{h}(\cdot, \theta)$, respectively, where θ is a tunable parameter vector. The structure of the function $\tilde{h}$ is chosen so that for any given x and θ, $\tilde{h}(x, \theta)$ is easy to compute. Once the general form of the function $\tilde{h}(\cdot, \cdot)$ is fixed, the next step is to set θ and $\tilde{v}$ so that the resulting function $\tilde{h}(\cdot, \theta)$ provides an approximate solution to Bellman's equation. Any particular choice of θ leads immediately to a policy μ^θ, given by

$$\mu^\theta(x, \omega) = \arg \max_{u \in U(x,\omega)} [g(x, \omega, u) + \tilde{h}(x', \theta)]. \tag{5}$$

This is similar to (4), which defines an optimal policy, except that the approximation $\tilde{h}(x', \theta)$ is used instead of $h(x')$.

There are two main ingredients in this methodology, to be discussed separately in the sections that follow:

a) defining an "approximation architecture," that is, the general form of the function $\tilde{h}(\cdot, \cdot)$;

b) developing a method, usually simulation based, for tuning θ and $\tilde{v}$.

A. Approximation Architecture

In defining suitable approximation architectures, one usually starts with a process of feature extraction. This involves a feature vector $f(x)$, which is meant to capture those "features" of the state x that are considered most relevant to the decision making process. Usually, the feature vector is handcrafted based on available insights on the nature of the problem, prior experience with similar problems, or experimentation with simple versions of the problem. Our choice of a feature vector will be described shortly.

Given the choice of the feature vector, a commonly used approximation architecture is of the form $\tilde{h}(f(x), \theta)$, where $\tilde{h}$ is a multilayer perceptron with input $f(x)$ and internal tunable weights θ (see, e.g., [8]). This architecture is powerful because it can approximate arbitrary functions of $f(x)$. The drawback, however, is that the dependence on θ is nonlinear, and tuning θ can be time consuming and unreliable.

An alternative is provided by a linear feature-based approximation architecture, in which we set

$$\tilde{h}(x, \theta) = \theta^T f(x).$$

Here, the superscript T stands for transpose, and the dimension of the parameter vector θ is set to be equal to the number of features—the dimension of the feature vector $f(x)$. Because of the linear dependence on θ, the problem of tuning θ resembles the linear regression problem, and is generally much more reliable.

Let $n_{l,\,m}$ be the number of class m calls that are active and which have been assigned to routes that go through link l. We view the variables $n_{l,\,m}$ and the products of the form $n_{l,\,m}n_{l,\,m'}$ as features, and we will work with a linear approximation architecture of the form

$$\tilde{h}(x,\,\theta) = \sum_{l\in\mathcal{L}} \left[\theta(l) + \sum_{m} \theta(l,\,m)n_{l,\,m} + \sum_{(m,\,m'):m'\le m} \theta(l,\,m,\,m')n_{l,\,m}n_{l,\,m'}\right]. \quad (6)$$

Note that for this architecture the number of tunable parameters is equal to

$$L(2 + 1.5M + 0.5M^2)$$

where L is the number of unidirectional links in the network and M is the number of service classes, i.e., the "complexity" of the architecture grows linear in the number of links and quadratic in the number of service classes.

A main reason for choosing a quadratic function of the variables $n_{l,\,m}$ is that it led to essentially optimal solutions to single link problems (see Section IV-A). Note that we have only included those products $n_{l,\,m}n_{l',\,m'}$ associated with a common link $(l = l')$. There are two reasons behind this choice: it opens up the possibility of a decomposable training algorithm (cf. Section III-C). In addition, it results in policies with an appealing decentralized structure, which we now discuss.

Let $n_{l,\,m}$ be the variables associated with the current state x of the network and suppose that ω corresponds to an arrival of class m^*. Let us focus on a particular decision $u \in U(x,\,\omega)$, which assigns this call to route r, resulting in a new state x' and variables $n'_{l,\,m}$. Note that $n'_{l,\,m} = n_{l,\,m} + 1$ if $l \in r$, $m = m^*$, and $n'_{l,\,m} = n_{l,\,m}$, otherwise. With some straightforward algebra, the merit $Q(x,\,\omega,\,r,\,\theta)$ of this decision, in comparison to rejection, is given by

$$\begin{aligned} &Q(x,\,\omega,\,r,\,\theta) \\ &\quad = g(x,\,\omega,\,u) + \tilde{h}(x',\,\theta) - \tilde{h}(x,\,\theta) \\ &\quad = c(m^*) + \sum_{l\in r}\left[\theta(l,\,m^*) + \theta(l,\,m^*,\,m^*)(2n_{l,\,m^*} + 1) + \sum_{m<m^*}\theta(l,\,m^*,\,m)n_{l,\,m} + \sum_{m>m^*}\theta(l,\,m,\,m^*)n_{l,\,m}\right]. \end{aligned}$$

The corresponding policy $\mu^\theta(\cdot,\,\cdot)$ [cf. (5)] amounts to choosing a route r^* for which $Q(x,\,\omega,\,r,\,\theta)$ is largest, using this route if $Q(x,\,\omega,\,r^*,\,\theta) > 0$, and rejecting the call if $Q(x,\,\omega,\,r^*,\,\theta) \le 0$. This is equivalent to assigning a link cost (or shadow price)

$$\theta(l,\,m^*) + \theta(l,\,m^*,\,m^*)(2n_{l,\,m^*} + 1) + \sum_{m<m^*}\theta(l,\,m^*,\,m)n_{l,\,m} + \sum_{m>m^*}\theta(l,\,m,\,m^*)n_{l,\,m} \quad (7)$$

to each link, and using these link costs for admission control and shortest path routing. Note that these link costs (shadow prices) (7) are state dependent and reflect the instantaneous congestion on each link which is in the spirit of [6], [11]. However, the notion of a link cost results here from a specific choice of a approximation architecture, and not from an explicit decomposition of the network process into independent link processes as in [6], [11].

The family of policies μ^θ resulting from our approximation architecture can provide a fair amount of flexibility. It remains to assess:

a) whether there are systematic methods for finding good policies within this family; this is the subject of the next section;

b) whether they lead to significant performance improvement in comparison to more restricted families of policies; this is to be assessed experimentally in Section IV.

B. The Training Algorithm

There are several methods that can be used to tune the parameter θ, most of which rely on simulation runs (or on online observations of an actual system). We will use a variant of one of the most popular methods, namely, Sutton's TD(0) ("temporal differences") algorithm [20]. The standard TD(0) algorithm has been designed for discrete time problems with a discounted criterion (or for an undiscounted total reward criterion in systems that eventually terminate), where the goal is to maximize the so called discounted reward-to-go of state x, given by

$$E\left[\sum_{k=0}^{\infty} e^{-\beta t_k} g(x_{t_k},\,\omega_k,\,u_{t_k}) | x_0 = x\right]$$

simultaneously for every state of the system. Here, $\beta > 0$ is a discount factor. So, some modifications are necessary to apply TD(0) to our problem. The first one, going from discrete to continuous time is fairly straightforward. The second one, going from a discounted to an average reward criterion, is much more substantial, since average reward dynamic programming theory and algorithms are generally more complex. We will use the recently developed temporal difference method for average reward problems [24], which preserves the same convergence properties and error guarantees of its discounted counterpart [23]. It should be noted that this is the first time that this method is applied to an engineering problem.

In the simplest version of TD(0), the controlled Markov process x_t is simulated under a fixed policy μ. Let t_k be the time of the kth event ω_k, which finds the system at state x_{t_k}, and let $u_{t_k} = \mu(x_{t_k},\,\omega_k)$ be the resulting decision. At such an event time, the vector θ and the scalar $\tilde{v}$ are updated according to

$$\theta_k = \theta_{k-1} + \gamma_k d_k \nabla_\theta \tilde{h}(x_{t_{k-1}},\,\theta_{k-1}) \quad (8)$$

$$\tilde{v}_k = \tilde{v}_{k-1} + \eta_k(g(x_{t_{k-1}},\,\omega_{k-1},\,u_{t_{k-1}}) - (t_k - t_{k-1})\tilde{v}_{k-1}) \quad (9)$$

where the "temporal difference" d_k is defined by

$$\begin{aligned} d_k = {} & g(x_{t_{k-1}}, \omega_{k-1}, u_{t_{k-1}}) - (t_k - t_{k-1})\tilde{v}_{k-1} \\ & + \tilde{h}(x_{t_k}, \theta_{k-1}) - \tilde{h}(x_{t_{k-1}}, \theta_{k-1}) \end{aligned}$$

and where γ_k and η_k are small step size parameters. The only difference from discrete time average reward TD(0) is in the factor of $t_k - t_{k-1}$ that multiplies $\tilde{v}_{k-1}$ and which, in turn, is due to the factor $E_\tau\{\tau|x\}$ in Bellman's equation. Note that with our linear approximation architecture $\tilde{h}(x, \theta) = \theta^T f(x)$, we have $\nabla_\theta \tilde{h}(x, \theta) = f(x)$.

Under a fixed policy, and under the standard diminishing step size conditions, $\tilde{v}_k$ converges to the average reward $v(\mu)$, and θ_k converges to a limiting vector θ such that $\tilde{h}(\cdot, \theta)$ provides a "good" approximation of $h^\mu(\cdot)$. Here, $h^\mu(\cdot)$ is a function defined similar to $h^*(\cdot)$, but in a context in which there is a single possible decision at each state, the one prescribed by the policy μ. Furthermore, the approximation is "good" in the sense that the approximation error $\tilde{h}(\cdot, \theta) - h^\mu(\cdot)$, measured under a suitable norm, is of the same order of magnitude as the best possible approximation error under the given approximation architecture [24].

One can start with a policy μ, run TD(0) until it converges, use the resulting limiting value of θ to define a new policy according to (5), and then repeat. This method has some (weak) theoretical guarantees [2], but it is common practice to keep changing the underlying policy with each update of the parameter vector θ_k. This optimistic TD(0) method is completely described by the update rule (8) together with

$$u_{t_k} = \arg \max_{u \in U(x_{t_k}, \omega_k)} [g(x_{t_k}, \omega_k, u) + \tilde{h}(x', \theta_k)] \qquad (10)$$

where x' is the successor state that results from x_{t_k}, ω_k, and u.

Even though *optimistic* TD(0) has no convergence guarantees, its discounted variant has been found to perform well in a variety of contexts [19], [22], [27].

C. A Decomposition Approach

The algorithm described in the preceding section can be very slow to converge, especially for networks with a substantial number of links. This led us to consider a decomposition approach that breaks the reward associated with a call into link rewards in the spirit of [6], [11], and which led to much shorter training times. This improvement in terms of training time is essential for applying NDP to large networks (see Section IV-C).

For any link l, consider the local "state" $x^{(l)} = (n_{l,m}: \forall m)$. Of course, this is not a state in the true sense of the word because, in general, it does not evolve as a Markov process, but will be treated to some extent as if it were. We decompose the immediate reward $g(x_{t_k}, \omega_k, u_k)$ associated with the kth event, into a sum of rewards attributed to each link

$$g(x_{t_k}, \omega_k, u_{t_k}) = \sum_{l \in \mathcal{L}} g^{(l)}(x_{t_k}, \omega_k, u_{t_k}).$$

In particular, whenever a new call (say, of class m) is routed over a route r that contains the link l, the immediate reward $g^{(l)}$ associated with link l is set to $c(m)/\#r$, where $\#r$ is the number of links along route r. For all other events, the immediate reward associated with link l is equal to zero.

Let us fix a policy μ, let $v^{(l)}(\mu)$ be the average reward attributed to link l, and note that

$$v(\mu) = \sum_{l \in \mathcal{L}} v^{(l)}(\mu).$$

For each link, we introduce a scalar $\tilde{v}^{(l)}$, which is meant to be an estimate of $v^{(l)}(\mu)$, as well an approximation architecture $\tilde{h}^{(l)}(x, \theta^{(l)})$ of the form

$$\begin{aligned} \tilde{h}^{(l)}(x^{(l)}, \theta^{(l)}) = {} & \theta(l) + \sum_m \theta(l, m) n_{l,m} \\ & + \sum_{(m, m'):\, m' \leq m} \theta(l, m, m') n_{l,m} n_{l,m'} \end{aligned}$$

where $\theta^{(l)}$ is the vector of parameters $\theta(l)$, $\theta(l, m)$, and $\theta(l, m, m')$, associated with link l. Note that

$$\tilde{h}(x, \theta) = \sum_{l \in \mathcal{L}} \tilde{h}^{(l)}(x^{(l)}, \theta^{(l)})$$

and we are therefore dealing with the same approximation architecture as in Section III-A. The key difference is that we will not update θ according to (8), but will use an update rule which is local to each link. The local TD(0) algorithm, for link l is given by

$$\begin{aligned} \theta_k^{(l)} = {} & \theta_{k-1}^{(l)} + \gamma_k^{(l)} d_k^{(l)} \nabla_{\theta^{(l)}} \tilde{h}^{(l)} \left(x^{(l)}_{t^{(l)}_{k-1}}, \theta^{(l)}_{k-1} \right) \\ \tilde{v}_k^{(l)} = {} & \tilde{v}_{k-1}^{(l)} + \eta_k^{(l)} \left(g^{(l)} \left(x^{(l)}_{t^{(l)}_{k-1}}, \omega^{(l)}_{k-1}, u_{t^{(l)}_{k-1}} \right) \right. \\ & \left. - (t_k^{(l)} - t_{k-1}^{(l)}) \tilde{v}^{(l)}_{k-1} \right) \end{aligned}$$

where

$$\begin{aligned} d_k^{(l)} = {} & g^{(l)} \left(x^{(l)}_{t^{(l)}_{k-1}}, \omega^{(l)}_{k-1}, u_{t^{(l)}_{k-1}} \right) - (t_k^{(l)} - t_{k-1}^{(l)}) \tilde{v}^{(l)}_{k-1} \\ & + \tilde{h}^{(l)} \left(x^{(l)}_{t^{(l)}_k}, \theta^{(l)}_{k-1} \right) - \tilde{h}^{(l)} \left(x^{(l)}_{t^{(l)}_{k-1}}, \theta^{(l)}_{k-1} \right) \qquad (11) \end{aligned}$$

$\gamma_k^{(l)}$ and $\eta_k^{(l)}$ are small step size parameters, and $t_k^{(l)}$ is the time of the kth event $\omega_k^{(l)}$ associated with link l. Here, we say that an event is associated with link l if it can potentially result in a change of $x^{(l)}$; this is the case if we have a departure of a call that was using link l, or if link l is part of a route in the predefined list of possible routes connecting the current origin-destination pair. This update rule is identical to the ordinary TD(0) update under the assumption that $x_t^{(l)}$ is a Markov process that receives rewards $g^{(l)}(x^{(l)}_{t^{(l)}_k}, \omega^{(l)}_k, u_{t^{(l)}_k})$ at the times $t_k^{(l)}$ of events associated with link l. Of course, $x_t^{(l)}$ is not Markov because its transitions are affected by the global state x_t. Although the update rules for different links are decoupled, they are to be carried out in the course of a single simulation of the entire system, which accurately reflects all dependencies involved. This is to be compared with [6], [11], where the entire system was explicitly decomposed into independent link processes, making $x_t^{(l)}$

truly Markov, however, at the expense of ignoring certain dependencies and introducing an additional modeling error.

IV. Experimental Results

In this section, we report the results obtained in a broad set of experiments. We compare the policy obtained through NDP with the commonly used heuristic OSPF. For every pair of source and destination nodes, OSPF orders the list of predefined routes. When a new call arrives, it is routed along the first route in the corresponding list that does not violate the capacity constraint. If no such route exists, the call is rejected. For a single link problem, OSPF reduces to the naive policy that always accepts an incoming call, as long as the required bandwidth is available.

A. Single Link Problems

Our first set of experiments involved multiple classes, but a single link. They were carried out in order to identify potential difficulties with this approach and to validate the promise of the quadratic approximation architecture. Naturally, with a single link, no decomposition had to be introduced. Two case studies were carried out involving three and ten service classes, respectively. For the latter case, three different scenarios were considered corresponding to a highly, medium, and lightly loaded link, respectively. A more detailed account of these experiments and the results obtained can be found in [15].

The experiments were carried out using TD(0) for discounted problems. The performance of the resulting policies was evaluated on the basis of the average reward criterion. The discount factor was chosen to be very small, which makes the discounted problem essentially equivalent to an average reward problem. The evaluation of the average reward is based on a trajectory of 4×10^6 time steps.

Besides TD(0) with a quadratic approximation architecture, we also used TD(0) with a multilayer perceptron (MLP) [8]. Furthermore, for the smaller problem, which only involved three classes, we also obtained an optimal policy through exact dynamic programming (DP) and used it as a basis of comparison. A comparison was also made with a naive policy that always accepts an incoming call, as long as the required bandwidth is available. By inspecting the nature of the best policy obtained using NDP, we observed that only some of the customer classes were ever deliberately rejected, and we were then able to use this knowledge to handcraft a trunk reservation (threshold) policy that attained comparable performance. However, in the absence of adequate tools for tuning trunk reservations parameters (as it is the case for large networks), the use of NDP can become very attractive. In addition, this illustrates that the quadratic approximator provides an adequate architecture for the differential reward function of a single link.

The parameters and results of the case studies are given in Tables I–V. One conclusion from these experiments is that NDP led to significantly better results than the heuristic "always accept" policy, except for the case of a lightly loaded link and ten classes, where the performance of both approaches was the same. (This is understandable because for a lightly loaded system "interesting" events such as blocking are too rare to be able to fine tune the policy.) In particular, for all cases, except for the one just mentioned, the rewards associated with calls that were blocked or deliberately rejected (these are the lost

TABLE I
Case Study for Three Service Classes and a Link with a Capacity of 12 Units

Service Class m	1	2	3
Bandwidth Demand $b(m)$	1.00	2.00	2.00
Average Holding Time $1/\nu(m)$	2.00	1.25	1.11
Arrival Rate $\lambda(m)$	3.00	2.00	2.50
Immediate Reward $c(m)$	4.00	15.00	12.00

	Performance	
	Average Reward	Lost Average Reward
Always Accept	40.09	31.86
Trunk Reservation	47.23	24.72
Dynamic Programming	47.23	24.72
TD(0): MLP	47.19	24.76
TD(0): Quadratic	47.19	24.76

TABLE II
Problem Data of the Case Study for Ten Service Classes on a Link with a Capacity of 600 Units

Service Class m	1	2	3	4	5	6	7	8	9	10
Bandwidth Demand $b(m)$	2.00	2.00	4.00	4.00	6.00	6.00	8.00	8.00	10.00	10.00
Average Holding Time $1/\nu(m)$	1.00	1.00	1.25	1.25	1.67	1.67	2.50	2.50	5.00	5.00
Immediate Reward $c(m)$	2.00	1.40	5.00	2.50	10.00	4.00	20.00	7.00	5.00	16.00
Arrival Rate $\lambda(m)$ (high load)	15.00	15.00	12.00	12.00	10.00	10.00	6.00	6.00	4.00	4.00
Arrival Rate $\lambda(m)$ (medium load)	15.00	15.00	10.00	10.00	7.00	7.00	3.00	3.00	1.80	1.80
Arrival Rate $\lambda(m)$ (light load)	16.00	16.00	12.00	12.00	7.00	7.00	2.40	2.40	1.10	1.10

TABLE III
Case Study for Ten Service Classes and a Highly Loaded Link with a Capacity of 600 Units

	Performance	
	Average Reward	Lost Average Reward
Always Accept	412.77	293.15
Trunk Reservation	518.17	187.67
TD(0): MLP	505.46	200.34
TD(0): Quadratic	511.45	194.71

TABLE IV
Case Study for Ten Service Classes and a Medium Loaded Link with a Capacity of 600 Units

	Performance	
	Average Reward	Lost Average Reward
Always Accept	389.53	34.31
Trunk Reservation	396.68	26.97
TD(0): MLP	393.68	29.87
TD(0): Quadratic	385.39	38.63

TABLE V
CASE STUDY FOR TEN SERVICE CLASSES AND A LIGHTLY LOADED LINK WITH A CAPACITY OF 600 UNITS

	Performance	
	Average Reward	Lost Average Reward
Always Accept	370.82	8.89
Trunk Reservation	372.40	7.70
TD(0): MLP	370.82	8.89
TD(0): Quadratic	370.82	8.89

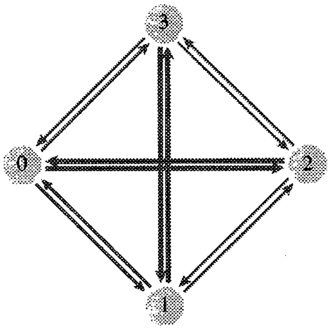

Fig. 1. Telecommunication network consisting of four nodes and 12 unidirectional links.

TABLE VI
SERVICE CLASSES AND ARRIVAL RATES FOR THE FOUR-NODE NETWORK

Service Class m	1	2	3
Bandwidth Demand $b(m)$	1	3	5
Average Holding Time $1/\nu(m)$	10	10	2
Immediate Reward $c(m)$	1	2	50

	Arrival Rates		
Service Class	1	2	3
Origin-Destination Pairs (0-2)(2-0)(1-3)(3-1)	4.40	2.64	0.44
All Other Origin-Destination Pairs	2.00	1.20	0.20

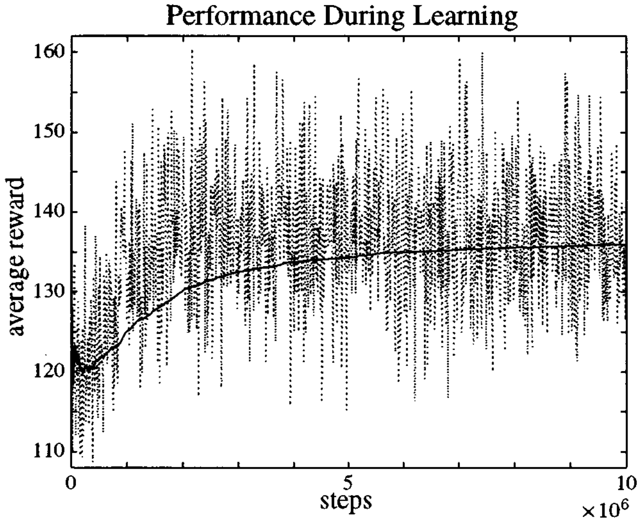

Fig. 2. “Empirical” average reward per time unit during the whole training phase of 10^7 steps (solid) and during shorter time windows of 10^5 steps (dashed).

rewards), were reduced by 10%–35%. For the case of three classes, essentially optimal performance was attained. It was also seen that the MLP architecture did not lead to performance improvements, and this was an important reason for not using it in larger problems.

B. A Four-Node Network

In this section, we present experimental results obtained for the case of an integrated services network consisting of four nodes and 12 unidirectional links. There are two different classes of links with a total capacity of 60 and 120 units of bandwidth, respectively (indicated by thick and thin arrows in Fig. 1). We assume a set $\mathcal{M} = \{1, 2, 3\}$ of three different service classes. The corresponding parameters are given in Table VI. Note that the calls of type 3 are much more valuable than the one of type 1 and 2. Furthermore, for each pair of source and destination nodes, the list of possible routes consists of three entries: the direct path and the two alternative two-hop routes.

This case study is characterized by a high traffic load and by calls of one service class having a much higher immediate reward than calls of the other types. Clearly, for this case, a good CAC and routing policy should give “priority” to calls of the service class with the highest reward. We chose this setting to determine the potential of our optimization algorithm, i.e., to find out if NDP indeed discovers a control policy which reserves bandwidth for calls of the most valuable service type.

This experiment was carried out using TD(0) for discounted problems combined with the decomposition approach. However, the performance of the resulting policies was evaluated on the basis of the average reward criterion. Our value function approximator contains 120 tunable parameters. There are approximately 1.6×10^{45} different link state (feature) configurations. Note that the cardinality $|S|$ of the underlying state space is even higher. We make the following observations.

a) Employing the decomposition approach did not affect the the performance of our final NDP policy and reduced the training time by a factor of two. (Note that the decomposed optimization updates the parameters corresponding to only five links instead of twelve at every time step.) This was an important reason for using it in larger problems (see Section IV-C).

b) In order to assure convergence of the discounted TD(0) method, we had to carefully handcraft some of the initial parameter values of our function approximator. In particular, the magnitude of the parameter $\theta(l)$ associated with each link turned out to be critical. This procedure becomes rapidly impractical as the number of links increases. Larger problems can be solved much easier using average reward algorithms which are less sensitive in this respect (see Section IV-C).

c) For this case study, we could significantly improve the performance of the resulting policy by enforcing an explicit exploration of the state space during the training. At each state, with probability $p = 0.5$, we apply a random action, instead of the action recommended by the current value function, to generate the next state in our training trajectory. However, the successor state $x^{(l)}_{t^{(l)}_k}$ that is used in update rule (11) is still chosen according to the greedy action given in (10). The importance of using a certain

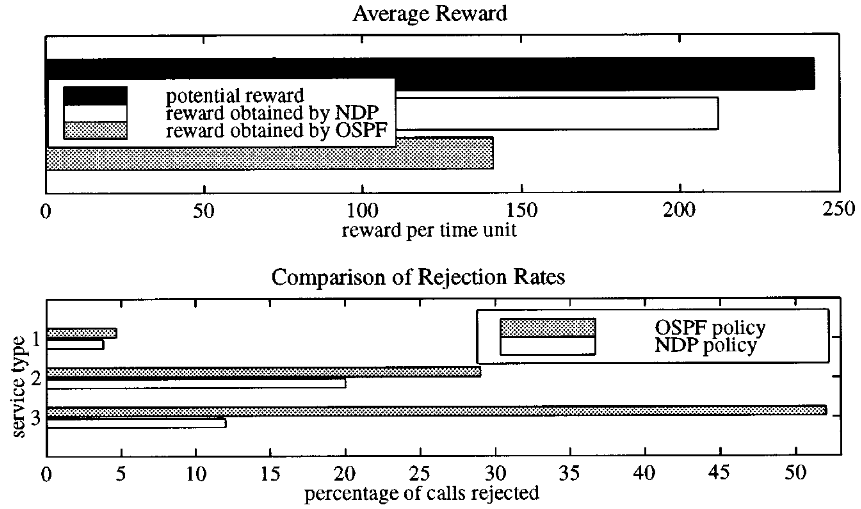

Fig. 3. Four-node network: Comparison of the average rewards and rejection rates of the NDP and OSPF polices.

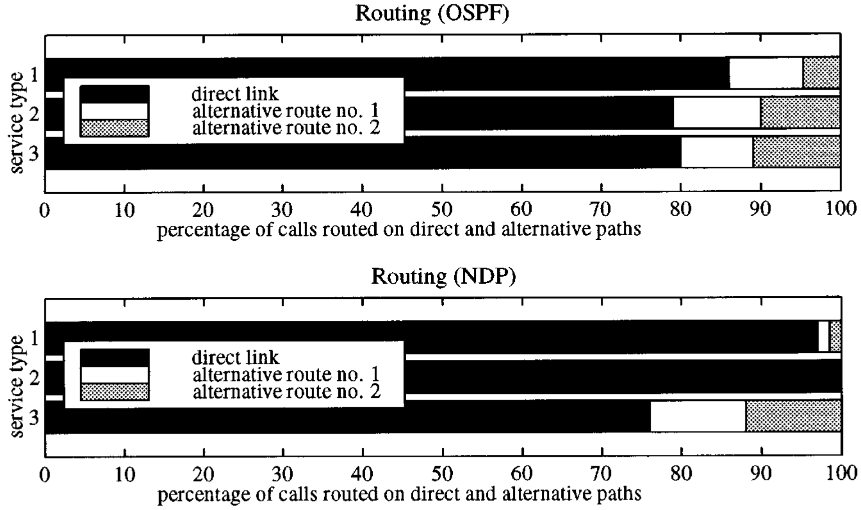

Fig. 4. Four-node network: comparison of the routing behavior of the NDP and OSPF policies.

amount of exploration in connection with NDP methods is well known (see for example [2]).

The results of the case study is given in Fig. 2 (training phase), Fig. 3 (performance), and Fig. 4 (routing behavior). We give here a summary of the results.

1) Training Phase: Fig. 2 shows the performance improvement during the optimization phase. Here, the "empirical" average reward of the NDP policy (computed by averaging the rewards obtained during the whole training and during shorter time window of 10^5 steps) is depicted as a function of the training steps. Although this average reward increases during the training, it does not exceed 141, the average reward of the heuristic OSPF. This is due to the high amount of exploration in the training phase. We obtained the final control policy after 10^7 iteration steps.

2) Performance Comparison: We used simulated trajectories of 10^7 time steps to evaluate our policies. The policy obtained through NDP gives an average reward of 212, which as about 50% higher than the one of 141 achieved by OSPF. Furthermore, the NDP policy reduces the number of rejected calls for all service classes. The most significant reduction is achieved for calls of service class 3, the service class, which has the highest immediate reward. Fig. 3 also shows that the average reward of the NDP policy is close to the potential average reward of 242, which is the average reward we would obtain if all calls were accepted. This leaves us to believe that the NDP policy is close to optimal. Fig. 4 compares the routing behavior of the NDP control policy and OSPF. While OSPF routes about 15%–20% of all calls along one of the alternative two-hop routes, the NDP policy uses alternate routes for calls of type 3 (about 25%) and routes calls of the other two service classes almost exclusively over the direct route. This indicates, that the NDP policy uses a routing scheme, which avoids two-hop routes for calls of service class 1 and 2, and which allows us to use network resources more efficiently.

C. A 16-Node Network

In this section, we present experimental results obtained for a network consisting of 16 nodes and 62 unidirectional links (see Fig. 5). The network topology is taken from [7]. The network consists of three different classes of links with a capacity of 60, 120, and 180 units of bandwidth, respectively. We assume four different service classes. Table VII summarizes the

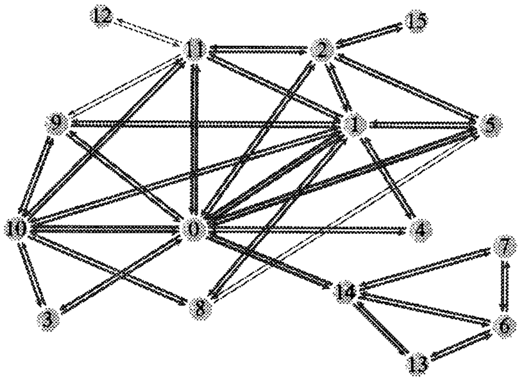

Fig. 5. Telecommunication network consisting of 16 nodes and 62 unidirectional links.

TABLE VII
SERVICE CLASSES FOR THE 16-NODE NETWORK

Service Class m	1	2	3	4
Bandwidth Demand $b(m)$	1.00	2.00	3.00	4.00
Average Holding Time $1/\nu(m)$	1.25	1.25	1.25	1.25
Immediate Reward $c(m)$	0.25	1.00	6.00	15.00

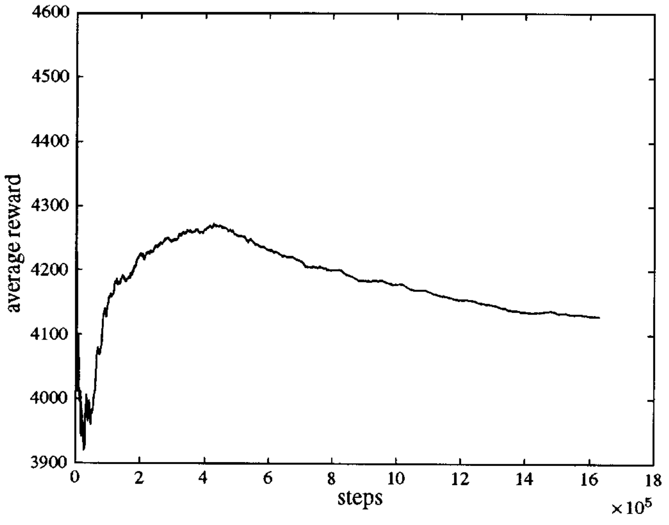

Fig. 6. "Empirical" average reward obtained during the training as a function of training steps. The performance initially improves and then suddenly deteriorates.

corresponding bandwidth demands, average holding times, and immediate rewards. The table of arrival rates is also taken from [7]. However, for our experiments we rescaled them by a factor of two. The list of accessible routes consists of a maximum of six minimal hop routes for each pair of source and destination nodes. Routes with an equal number of hops are ordered by their absolute path length (in miles) which is also reported in [7].

For this experiment, there are approximately 1.4×10^{256} different link state (feature) configurations and 992 tunable parameters. The results of the case study are summarized by Figs. 6 (training), 7 (performance), 8 (routing), and 9 (robustness).

We make the following observations.

a) Without using the decomposition approach, no substantial improvement over the initial policy is achieved within a reasonable amount of computation time (24 h, say). This illustrates the importance of the decomposition approach in applying NDP to the CAC and routing problem.

b) Discounted reward algorithms failed due to their critical dependence on initial parameter values (see Section IV-B). This difficulty does not arise with *average* reward algorithms.

c) Instabilities can occur during the training phase, even when exploration is employed (see the discussion below).

d) Our NDP policies are very robust with respect to changes of the underlying arrival statistics.

1) Training Phase: Fig. 6 shows the "empirical" average reward of the NDP policy (computed by averaging the rewards obtained during the simulation run) as a function of the training steps. In contrast to the four-node example, the NDP policy does not converge towards a final policy better than OSPF, although the average reward significantly improved during the first $4{\cdot}10^5$ training steps. Afterwards, a sudden performance breakdown occurs, from which the system never recovers. This loss of stability did not disappear, even if we introduce explicit exploration during the training. For the subsequent performance comparison between NDP and OSPF, we pick the best policy (given by the parameter values just before the loss of stability) generated in the course of the algorithm, not the last one.

2) Performance Comparison: The policies are empirically evaluated based on simulated trajectories of 10^7 time steps. The OSPF policy almost exclusively routes all calls over the shortest path. This leads to an average reward of about 4254. The rate of rejected calls is positive for all service classes. The two most valuable service classes 3 and 4 receive the highest rejection rate. In contrast, the NDP policy comes up with a very different routing scheme that uses alternative paths for all types of services. Now, the rejection rates for calls of type 1, 3 and 4 vanish whereas that for service class 2 increases. The NDP policy rejects these calls in a strategic way, i.e., NDP is not forced to do so by the capacity constraint. Instead, it explicitly reserves bandwidth for the most valuable calls of type 3 and 4. The average reward of 4349 obtained through the NDP policy is about 2.2% higher than the one achieved by OSPF. While this might appear to be a small improvement, it has to be viewed in perspective: even if we could achieve the potential average reward (which is 4438) by accepting every arriving call, the reward would only increase by 4.3%. Thus, the 2.2% improvement in rewards, is a substantial fraction of the best possible improvement. In fact, NDP reduces the lost average reward (potential average reward minus actual average reward) by about 52% compared with OSPF. Note that for this type of problems, the lost average reward is a more meaningful performance measure than the average reward. For example, if we have a single link and a single service class, it coincides with the blocking probability (rejection rate), which is the generally accepted performance metric. Blocking probabilities in well-designed systems are generally small, and an improvement from, say, 4% to 2% is generally viewed as substantial, even though it only represents a 2% increase of calls accepted.

3) Robustness: We applied our best policy obtained through training under the arrival statistics mentioned earlier to problems with randomly changed arrival rates in order to show the robustness of NDP policies. In particular, each arrival rate is

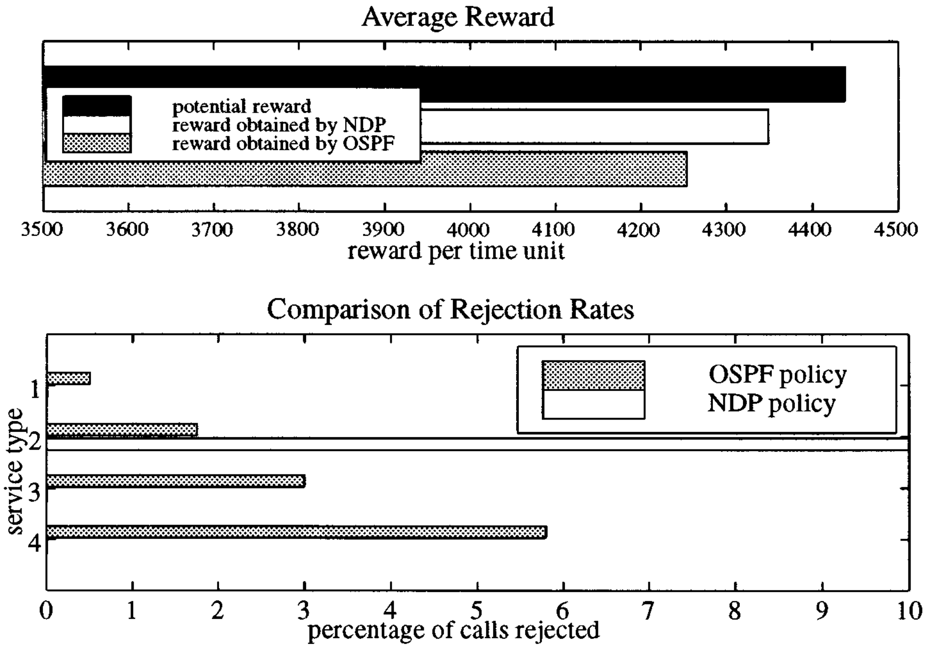

Fig. 7. 16-node network: comparison of the average rewards and rejection rates of the NDP and OSPF policies.

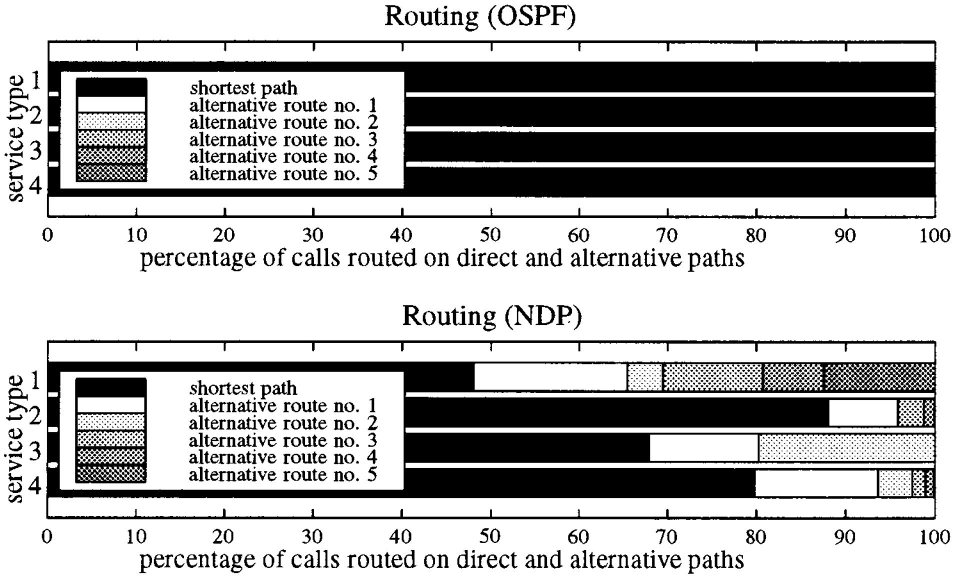

Fig. 8. 16-node network: comparison of the routing behavior of the NDP and OSPF policies.

multiplied by a factor $1 + \rho$, where $\rho \in [-\alpha, \alpha]$ is independently drawn from a uniform distribution. An arrival rate is set to zero, if $1 + \rho$ happens to be negative. We carried out a set of experiments by varying the magnitude $\alpha \in [0, 2]$ in steps of 0.1, which amounts to rather strong perturbations of the traffic statistics. Fig. 9 shows the result of these experiments. The magnitude α of the relative perturbations of the arrival rates is depicted against the relative lost reward defined as

$$\frac{v(\mu_{\mathrm{NDP}}) - v(\mu_{\mathrm{OSPF}})}{v_{\mathrm{potential}} - v(\mu_{\mathrm{OSPF}})}.$$

Here, $v_{\mathrm{potential}}$, μ_{NDP}, and μ_{OSPF} denote the potential average reward, the NDP policy and the OSPF policy, respectively. The experiments show, that our NDP policy is indeed very robust against changes in the arrival rates. There is only one out of twenty experiments where the NDP policy happened to be worse then OSPF. (We did not average several experiments with equal perturbation parameter α.) For all other arrival statistics the NDP policy still outperforms OSPF with a relative lost reward between 25% and 70%.

V. Conclusion

The CAC and routing problem for integrated service networks is naturally formulated as an average reward dynamic programming problem, but with a very large state space. Traditional dynamic programming methods are computationally infeasible for such large scale problems. We use neuro-dynamic programming, based on the average reward TD(0) method of

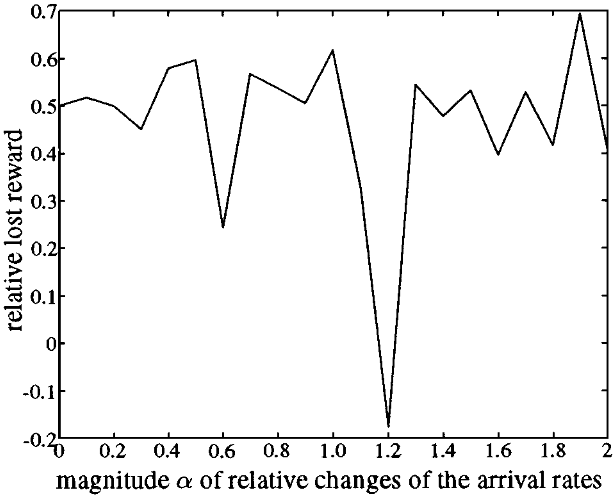

Fig. 9. Relative lost reward of the NDP policy applied to networks with randomly changed arrival statistics.

[24], combined with a decomposition approach that views the network as consisting of decoupled link processes. This decomposition has the advantage that it allows for decentralized decision making and decentralized training, which reduces significantly the training time. We have presented experimental results for several example problems of different sizes. The case study involving a 16-node network shows that NDP can lead to sophisticated control policies involving strategic call rejections, which are difficult to obtain through heuristics.

Compared with the heuristic OSPF, the NDP policy reduces the lost average reward by 50% (heavily loaded four node network), 52% (lightly loaded 16-node network), and (except for one out of twenty experiments) by 20%–70% (16-node network under different loads). This illustrates that NDP has the potential to significantly improve performance over a broad range of network loads.

Concerning the practical applicability of this general methodology, there are two somewhat distinct issues. The first is whether dynamic policies based on state-dependent costs (depending linearly on the variables $n_{l,m}$) can lead to significant performance improvements. Our results suggest that this is indeed the case, although a comparison with alternative policies (such as dynamic alternative routing with trunk reservation) remains to be made. A somewhat related issue is whether efficient performance evaluation tools are possible (based on ideas similar to the reduced load approximation, that do not involve simulation) which apply to policies of the form considered in this paper.

The second issue refers to computational requirements. Simulation based methods such as TD can be slow. For example, the computation times for our different experiments ranged from one to four hours of CPU time on a Sun Sparc 20 workstation. On the other hand, once we can see promise in an application domain, a variety of ways of improving speed can be considered. Besides optimizing the code, these could include batch linear least squares methods for tuning θ (to replace small step size incremental training), or the use of a smaller set of tunable parameters after identifying those "features" that are most critical for improved performance. Nevertheless, it seems that NDP is best suited as a tool for offline rather than online optimization of the CAC and routing policy.

It should be noted that while the (offline) training time of the NDP policy can be in the order of minutes or hours, the "complexity" of implementing (online) a NDP policy (for a fixed parameter vector) is very similar to the one of OSPF, i.e., the "cost" of a route can be determined by simply adding up the corresponding "link shadow prices," which are given by a quadratic functions.

References

[1] D. P. Bertsekas, *Dynamic Programming and Optimal Control*. Belmont, MA: Athena Scientific, 1995.

[2] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*. Belmont, MA: Athena Scientific, 1996.

[3] T. X. Brown, H. Tong, and S. Singh, "Optimizing admission control while ensuring quality of service in multimedia networks via reinforcement learning," in *Advances in Neural Information Processing Systems 11*. Cambridge, MA: MIT Press, 1999.

[4] J. Carlström and E. Nordström, "Reinforcement learning for control of self-similar call traffic in broadband networks," in *Teletraffic Engineering in a Competitive World, Proc. 16th Int. Teletraffic Congress (ITC'16)*, P. Key and D. Smith, Eds., Elsevier/Edinburgh, U.K., 1999, pp. 571–580.

[5] S. Chung and K. W. Ross, "Reduced load approximations for multi-rate, multi-hop communication networks," *IEEE Trans. Commun.*, vol. 41, Aug. 1993.

[6] Z. Dziong and L. G. Mason, "Call admission and routing in multi-service loss networks," *IEEE Trans. Commun.*, vol. 42, Feb./Mar./Apr. 1994.

[7] A. Greenberg and R. Srikant, "Computational techniques for accurate performance evaluation in multirate, multihop communication Networks," *IEEE/ACM Trans. Networking*, vol. 5, Apr. 1997.

[8] S. Hayking, *Neural Networks: A Comprehensive Foundation*, New York: McMillian, 1994.

[9] F. P. Kelly, "Routing in circuit switched networks: Optimization, shadow prices and decentralization," *Adv. Appl. Prob.*, vol. 20, 1988.

[10] ——, "Loss networks," *Ann. Appl. Probab.*, vol. 1, pp. 319–378, 1991.

[11] P. B. Key, "Optimal control and trunk reservation in loss networks," *Prob. Eng.. Inform. Sci.*, vol. 4, pp. 203–242, 1990.

[12] C. N. Laws, "On trunk reservation in loss networks," in *Stochastic Networks*, F. P. Kelly and R. J. Williams, Eds, New York: Springer-Verlag, 1995, vol. 71, IMA Volumes in Mathematics and its Applications, pp. 187–198.

[13] M. Y. Liu, J. S. Baras, and A. Misra, "Performance evaluation in multi-rate, multi-hop communication networks with adaptive routing," in *Proc. ARL Federated Laboratory Second Annu. Conf.* College Park, MD, Feb. 1998.

[14] M. Y. Liu, "Performance Evaluation in Multi-Rate Multi-Hop Communication Networks," Master's thesis, Univ. Maryland, College Park, MD, 1997.

[15] P. Marbach and J. N. Tsitsiklis, "A Neuro-Dynamic Programming Approach to Call Admission Control in Integrated Service Networks: The Single Link Case,", Massachusetts Institute of Technology, Cambridge, MA, Lab. Inform. Decision Syst. Rep. LIDS-P-2402. [Online]. Available WWW: http:/web.mit.edu/jnt/www/publ.html, Nov. 1997.

[16] D. Mitra, J. A. Morrison, and K. G. Ramakrishnan, "ATM network design and optimization: A multirate loss network framework," *IEEE/ACM Trans. Networking*, vol. 4, Aug. 1996.

[17] E. Nordström and J. Carlström, "A reinforcement learning scheme for adaptive link allocation in ATM networks," in *Proc. Int. Workshop of Applications Neural Networks in Telecommunication 2 (IWANNT*95)*, J. Alspector, R. Goodman, and T. X. Brown, Eds. Stockholm, Sweden, 1995, pp. 88–95.

[18] K. W. Ross, *Multiservice Loss Models for Broadband Communication Networks*. Berlin, Germany: Springer-Verlag, 1995.

[19] S. Singh and D. P. Bertsekas, "Reinforcement learning for dynamic channel allocation in cellular telephone systems," in *Advances in Neural Information Processing Systems 9*, M. C. Mozer, M. I. Jordan, and T. Petsche, Eds. Cambridge, MA: MIT Press, 1997, pp. 974–980.

[20] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine Learning*, vol. 3, pp. 9–44, 1988.

[21] R. S. Sutton and A. B. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.

[22] J. Tesauro, “Practical issues in temporal difference learning,” *Machine Learning*, vol. 8, 1988.

[23] J. N. Tsitsiklis and B. Van Roy, “An analysis of temporal-difference learning with function approximation,” *IEEE Trans. Automat. Contr.*, vol. 42, pp. 674–690, May 1997.

[24] ——, “Average Cost Temporal-Difference Learning,” Massachusetts Institute of Technology, Cambridge, MA, Lab. Inform. Decision Syst. Report LIDS-P-2390, 1997.

[25] J. Walrand and P. Varaiya, *High-Performance Communication Networks*. San Francisco, CA: Morgan Kaufman, 1996.

[26] P. Whittle, “Approximation in large-scale circuit-switched networks,” *Prob. Eng. Inform. Sci.*, vol. 2, pp. 279–291, 1988.

[27] W. Zhang and T. G. Dietterich, “High performance job-shop scheduling with a time-delay TD(λ) network,” in *Advances in Neural Information Processing Systems 8*, D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, Eds. Cambridge, MA: MIT Press, 1996, pp. 1024–1030.

Peter Marbach was born in Lucerne, Switzerland. He received the Eidg. Dipl. El.-Ing. from the ETH Zurich, Switzerland, in 1993, the M.S. in electrical engineering from the Columbia University, NY, in 1994, and the Ph.D. in electrical engineering from the Massachusetts Institute of Technology, Cambridge, in 1998.

Since 1998, he has been with the Center for Communication Systems Research at the University of Cambridge, Cambridge, U.K. He has also been a Visiting Scientist at the Siemens Corporate Research Center, Munich, Germany, in 1996. His research interests are in the fields of communication networks, stochastic systems, optimization, and control.

Oliver Mihatsch was born in Munich, Germany, in 1967. He received the Dipl. and Ph.D. degrees in mathematics from the Technical University Munich (TUM), Germany, in 1993 and 1998, respectively.

During 1993–1995, he was a Teaching and Research Assistant at the Department of Mathematics, TUM. Since 1996, he has been with the Siemens AG, Corporate Technology, where he is currently working as a Scientist in the fields of neural computation, optimal control, and nonlinear dynamics.

John N. Tsitsiklis was born in Thessaloniki, Greece, in 1958. He received the B.S. degree in mathematics in 1980, and the M.S. and Ph.D. degrees in electrical engineering, in 1981 and 1984, respectively, all from the Massachusetts Institute of Technology (MIT), Cambridge.

During the academic year 1983-84, he was an acting Assistant Professor of electrical engineering at Stanford University, Stanford, CA. Since 1984, he has been with MIT, where he is currently Professor of electrical engineering and computer science. He has served as acting Codirector of the MIT Laboratory for Information and Decision Systems (Spring 1996 and 1997). He has also been a Visitor with the Department of Electrical Engineering and Computer Science at the University of California, Berkeley, and the Institute for Computer Science, Iraklion, Greece. His research interests are in the fields of systems, optimization, control, and operations research. He has written about 80 journal papers in these areas. He was an Associate Editor of *Automatica* and is an Associate Editor of *Applied Mathematics Letters*.

Dr. Tsitsiklis was a recipient of an IBM Faculty Development Award in 1983, an NSF Presidential Young Investigator Award in 1986, an Outstanding Paper Award by the IEEE Control Systems Society, the MIT. Edgerton Faculty Achievement Award in 1989, the Bodossakis Foundation Prize in 1995, and the INFORMS/CSTS prize in 1997. He was a Plenary Speaker at the 1992 IEEE Conference on Decision and Control. He has been an Associate Editor of the IEEE TRANSACTIONSON AUTOMATIC CONTROL.