Predictive Gain Scheduling for Control of Self-Similar Call Traffic in Multi-Service Networks

Jakob Carlström

Department of Computer Systems, Information Technology, Uppsala University,
Box 325, SE - 751 05 Uppsala, Sweden
Email: jakobc@DoCS.UU.SE

Abstract

This paper presents and evaluates a gain scheduling approach to solving the admission control and routing problems for self-similar call arrival processes. The control problem is decomposed into two sub-problems: prediction of near-future call arrival rates and computation of control policies for Poisson arrival processes. At decision time, the predicted arrival rates are used to select one of the control policies. The rate predictions are made by neural networks, trained on-line, and the control policies are computed using standard techniques for Markov decision processes. In simulations, this method achieves higher link utilization than methods which do not exploit the memory of the arrival process. It also adapts to the network traffic considerably faster than a previously presented controller employing reinforcement learning without decomposition of the problem.

1 Introduction

In multi-service communications networks, such as Asynchronous Transfer Mode (ATM) networks, resource control is of crucial importance for the network operator as well as for the network users. The objective of resource control is to maintain the service quality while maximizing the network operator's revenue. At the call level, service quality (Grade of Service) is measured in terms of call blocking probabilities, and the key resource to be controlled is bandwidth. If a uniform call charging policy is used, the revenue is proportional to the long-term bandwidth utilization. Network routing and call admission control (CAC) are two important resource control problems in multi-service networks.

Theory for Markov decision processes (MDPs) offers theoretically well-established framework for optimal CAC and routing [1]. By modelling the dynamics of the network with traffic and computing control policies using dynamic programming [2, 3], resource control is optimized. A stan-

dard assumption in the derivations of such models is that calls arrive according to (constant-rate) Poisson processes. This makes the models of the dynamics relatively simple. Although the Poisson assumption still is valid for most user-initiated requests in communications networks, a number of studies [4, 5, 6, 7] indicate that many types of arrival processes in wide-area networks as well as in local area networks are statistically *self-similar* [8]. This makes it difficult to find models of the dynamics, and the models themselves become large and complex. If the number of system states grows too large, straightforward application of dynamic programming becomes unfeasible. Nevertheless, the "fractal" burst structure of self-similar traffic should be possible to exploit in the design of efficient resource control methods.

The solution to the CAC problem for self-similar call arrival processes presented and evaluated in this paper is referred to as *predictive gain scheduling*. It decomposes the control problem into two parts: prediction of near-future call arrival rates and selection among precomputed control policies for Poisson arrival processes based on these predictions. Thus, the self-similar arrival process is approximated by a quasi-stationary Poisson process. The rate predictions are made by neural networks, trained on-line. The control policies can be computed using dynamic programming or other *reinforcement learning* techniques [9, 10].

Control based on *temporal-difference learning*, a reinforcement learning technique [9], offers an alternative to predictive gain scheduling. In [11], we showed how to use temporal-difference (TD) learning for CAC of self-similar call traffic. The TD learning controller finds optimal solutions to Markov Decision Problems by iterative refinement of the control policy. It samples the immediate rewards (in terms of carried traffic) upon state transitions in a simulation model. Unlike dynamic programming, no model of system state transition probabilities and expected immediate rewards is needed. Our TD learning controller employs a state descriptor containing the number of active calls per service class along with features extracted from the history of the arrival processes. In this paper, this link admission controller is compared to the novel gain scheduling controller.

Although this paper concentrates on the link admission control problem, the link admission controller we describe may be used as a building block for optimal link allocation or routing, as shown in [12, 13 and 14].

Other work on approximate solutions to the CAC and routing problems include [12, 15], where we proposed a method based on TD learning for link allocation, a sub-problem of network routing. In simulations with Poisson call processes, this method outperformed standard heuristics, and achieved performance similar to that of Dziong's and Mason's dynamic programming solution [13]. Further, Marbach et al. have shown how to extend the use of TD learning to network routing [14], also assuming Poisson call arrival processes. Tong et al. have studied the case where the routing problem is subject to Quality of Service constraints [16].

The paper is organized as follows: Section 2 motivates the use of self-similar traffic in our study. Section 3 states the link admission control problem. Section 4 shows how this problem can be solved by predictive gain scheduling. Section 5 summarizes our previous TD learning solution, which is compared to predictive gain scheduling in the simulations presented in section 6. Finally, section 7 concludes the paper.

2 Self-Similar Call Arrival Processes

The limitations of the traditional Poisson model for network arrival processes have been demonstrated in a number of studies [4, 5, 6, 7]. Poor correspondence with the Poisson process has been shown for a range of traffic types, including arrivals of HTTP requests and TCP connections in wide-area networks. This difference is manifested by heavy-tailed inter-arrival time distributions and long-term correlations in the arrival processes. Various self-similar (fractal-like) models for such processes have been shown to correspond better with this traffic.

The reasons for self-similarity are not yet fully understood. Current explanations are based on the characteristics of popular applications as well as the dynamical behavior of networks, for examples the effects of TCP flow control.

This paper studies exploitation of temporal correlations in control of self-similar call arrival processes in multi-service networks. One example of where this situation occurs is an ATM network carrying TCP/IP traffic with self-similar connection arrivals. As another striking feature of network traffic is heavy-tailed holding times [6, 7], this should also be possible to exploit in traffic control. However, this is left as future work.

2.1 Properties of Self-Similar Traffic

A self-similar arrival process has no "natural" burst length. On the contrary, its arrival intensity varies considerably over many time scales. This makes the variance of its sample mean decay slowly with the sample size, and its auto-correlation function to decay slowly with time, compared to Poisson traffic [5].

The complexity of control and prediction of Poisson traffic is reduced by the memory-less property of the Poisson process: its expected future depends entirely on the (constant) arrival intensity, not on the history of the process. On the other hand, the long-range dependence of self-similar traffic makes it possible to improve the predictions of the process future by observing the process history. Norros [17] gives a rule-of-thumb for predicting one type of self-similar processes (fractional Brownian motion): "one should predict (with the appropriate nonuniform weights) the next second with the latest second, the next minute with the latest minute, etc."

A compact statistical measure of the degree of self-similarity of a stochastic process is the *Hurst parameter* [8]. For self-similar traffic this parameter takes values in the interval (0.5, 1], whereas Poisson processes have a Hurst parameter of 0.5.

3 The Link Admission Control Problem

In the link admission control problem, a link with capacity C [units/s] is offered calls from K different service classes. Calls belonging to such a class $j \in J = \{1, ..., K\}$ have the same bandwidth requirements b_j [units/s]. In the set-up procedure, the terminal that requests the new call declares what service class the new call belongs to.

We focus on arrival processes with memory, so we assume that calls arrive according to self-similar processes, and that their holding times are exponentially distributed with mean $1/\mu_j$ [s].

Access to the link is controlled by a *policy* π that maps *states* $x \in X$ to *actions* $a \in A$, $\pi: X \rightarrow A$. The set X contains all feasible link states, and the action set is

$$A = \left\{(a_1, ..., a_K) : a_j \in \{0, 1\}, j \in J\right\},$$

where a_j is 0 for rejecting a presumptive class-j call and 1 for accepting it. The set of link states is given by $X = N \times H$, where N is the set of feasi-

ble call number tuples, and H is the Cartesian product of some representations, h_j, of the history of the per-class call arrival processes (needed because of the memory of self-similar arrival processes). N is given by

$$N = \left\{ n : n_j \geq 0,\, j \in J;\, \sum_{j \in J} n_j b_j \leq C \right\},$$

where n_j is the number of type-j calls accepted on the link.

The objective of the link admission controller is to learn a policy that maximizes the long-term reward (revenue) on the link. By taking optimal actions, the policy controls the probabilities of state transitions so as to increase the probability of reaching states that yield high long-term rewards. We assume uniform call charging, which means that the reward rate $\rho(t)$ at time t is equal to the carried bandwidth:

$$\rho(x(t)) = \sum_{j \in J} n_j(t) b_j \tag{1}$$

Time evolves continuously, with discrete arrival and departure events $k = 0, 1, 2, \ldots$ The reward expected to be received from entering a state x_k at time t_k until entering the next state x_{k+1} at time t_{k+1}, is

$$R_{k+1} = E_\pi \left\{ \rho(x_k) \left[t_{k+1} - t_k \right] \right\} = \rho(x_k) \tau(x_k, \pi(x_k)) \tag{2}$$

where $\tau(x_k, \pi)$ is the expected sojourn time in state x_k under policy π.

In the presentation of the link admission control algorithms, we will use two different formulations of the control objective:

- average reward per stage, and
- infinite-horizon discounted reward.

The average-reward-per-stage objective is to find a policy π that maximizes

$$\overline{R}(\pi) = \lim_{N \to \infty} E_\pi \left\{ \frac{1}{N} \sum_{k=0}^{N} r_{k+1} \,\middle|\, x_0 = x \right\},\ x \in X. \tag{3}$$

Note that the average reward does not depend on the initial state x, as the contribution from this state to the average reward tends to zero as $N \to \infty$ (assuming non-zero probability of reaching any other state $y \in X$ from every state $x \in X$).

The infinite-horizon discounted reward objective is to maximize the *value function* for all $x \in X$:

$$V(x,\pi) = \lim_{T\to\infty} E_\pi\left\{\int_0^T e^{-\beta t}\rho(x(t))dt \;\middle|\; x_0 = x\right\},\ x \in X,\ \beta > 0\,. \qquad (4)$$

The exponential discounting of the future expected reward prevents the values from approaching infinity. With the appropriate normalization, this discounted reward objective approximates the average-reward-per-stage objective for small β [2]. One reason for using this approximation is that the convergence properties of TD learning algorithms based on the average-reward-per-stage objective remain to be ascertained. Further, [14] indicates stability problems with this type of TD learning. TD learning algorithms based on (4) can be proved to converge to optimal solutions under fairly general conditions [2].

Some of the states are especially interesting when it comes to finding the optimal policy. These are the states that are candidates for *intelligent blocking*. The set of such states $X_{ib} \subset X$ is identified by certain call number tuples: $X_{ib} = N_{ib} \times H$, where N_{ib} are the call number tuples where the available bandwidth is a multiple of the bandwidth of a wideband call. Assume for example that there are two call classes, $J = \{1, 2\}$, where class number 2 is a wideband class in comparison to class 1 ($b_2 > b_1$). Then N_{ib} is given by

$$N_{ib} = \left\{(n_1, n_2) : C - (n_1 b_1 + n_2 b_2) = k b_2,\ \ (n_1, n_2) \in N\right\}, \qquad (5)$$

where k is a positive integer. In the states of X_{ib}, the long-term reward may increase by rejecting a narrowband call to reserve bandwidth for future, expected wideband calls, especially for $k = 1$.

4 Solution by Predictive Gain Scheduling

Gain scheduling is a technique from control theory, where the parameters of a controller are changed as a function of varying operating conditions [18]. One example is aircraft control, where observations of speed and atmospheric pressure can be used too select controller parameters from a lookup table. The set of controller parameters is predetermined for the operating conditions the aircraft is presumed to encounter.

The approach taken here is to look up policies in a table from predictions of the near-future per-class call arrival rates. For Poisson arrival

processes, the optimal policy for the link admission control problem does not depend on the history, H, of the arrival processes. Due to the memoryless property, only the (constant) per-class arrival rates λ_j, $j \in J$, matter. In the gain scheduling control of self-similar arrival processes, near-future λ_j are predicted from h_j. The self-similar arrival processes are approximated by quasi-stationary Poisson processes, and precomputed polices are selected from a lookup table from the predicted λ_j. Radial-basis function neural networks are trained to predict the arrival rates.

4.1 Solving the Link Admission Control problem for Poisson Traffic

For Poisson arrival processes, there exist well-established techniques from dynamic programming for solving the LAC problem [19]. In this paper, policy iteration is used. It involves two steps:

1. value determination
2. policy improvement

The value determination step makes use of the objective function (3), and the concept of *relative values* [3]. The difference $v(x,\pi) - v(y,\pi)$ between two relative values under a policy π can be interpreted as the expected difference in accumulate reward over an infinite time interval, starting in state x instead of state y. In this paper, the relative values are computed by solving a system of linear equations, a method chosen for its fast convergence.

As stated above, the state descriptor for Poisson arrival processes needs not contain the history vectors, h_j. Accordingly, the state symbol x in the equations above can be replaced by the remaining part of the history descriptor: n. The equation system for the relative values under a policy π is

$$\begin{cases} v(n,\pi) = R(n,a) - \overline{R}(\pi)\tau(n,a) + \sum_{m \in X} p_{nm}(a)v(m,\pi); \;\; a = \pi(n), \; n \in N \\ v(n_r,\pi) = 0 \end{cases} \tag{6}$$

where n_r is an arbitrarily chosen state, for example the empty state. The expected sojourn time $\tau(n,a)$ in a state n after performing an action a and the probabilities p_{nm} for transition from state n to state m are given by

the following equations, where the λ_j are the the call arrival intensities and $1/\mu_j$ are the mean holding times:

$$\tau(n,a) = \frac{1}{\sum_{j\in J}\left(a_j\lambda_j + n_j\mu_j\right)} \tag{7}$$

and

$$p_{nm}(a) = \begin{cases} \lambda_j a_j \tau(n,a), & m = n + \delta_j \in N,\ j \in J \\ n_j\mu_j\,\tau(n,a), & m = n - \delta_j \in N,\ j \in J \\ 0, & \text{otherwise} \end{cases} \tag{8}$$

where δ_j is a vector of zeroes except for a 1 in the j-th position.

The policy improvement step consists of finding the action that maximizes the relative value in each state:

$$\pi_{k+1}(n) = \arg\max_{a\in A(n)}\left\{R(n,a) - \overline{R}(\pi_k)\tau(n,a) + \sum_{m\in N} p_{nm}(a)v(m,\pi_k)\right\} \tag{9}$$

After improving the policy, the value determination step is repeated for the new policy, and the policy is improved over again. This is iterated until $\pi_{k+1} = \pi_k$.

4.2 Determining The Prediction Horizon

Over what future time horizon should the rates used to select policies be predicted? In this work, the prediction horizon is set to an average of estimated mean first passage times from states back to themselves, henceforth referred to as the *mean return time* of state s. The arrival process is approximated by a quasi-stationary Poisson process within this time interval. Only states that are candidates for intelligent blocking are considered, and the average is weighted by the limit probabilities of the system being in these states.

The motivation for this choice of prediction horizon is that the effects of a decision (action) in a state x_d influence the future probabilities of reaching other states and receiving the associated rewards, until the state x_d is reached the next time. When this happens, a new decision can be made, where the previous decision does no longer influence the future expected reward.

Due to the continuous-valued history descriptor, the probability of returning to exactly the same state again is zero. But according to the assumption of quasi-stationarity, the history does not affect the expected future within the mean return time. Thus, the mean return time can be estimated for the call tuples n instead of the full state descriptor x.

In case of Poisson arrival processes, the mean first passage times $E_\pi\{T_{in}\}$ from other states to a state n are the unique solution of the linear system of equations

$$E_\pi\{T_{mn}\} = \tau(m,a) + \sum_{l \in N \setminus \{n\}} E_\pi\{T_{ln}\}, \quad m \in N \setminus \{n\}, \; a = \pi(m) \tag{10}$$

Such equation systems are solved for all states $n \in N_{ib}$, and the mean return times $E_\pi\{T_{nn}\}$ are extracted from the solutions.

The limiting probability q_n of occupying state n is determined for all states that are candidates for intelligent blocking, by solving

$$qB = 0 \tag{11}$$

under the constraint that the sum of limit probabilities $qe = 1$, where e is a vector of unity elements. B is a matrix containing the state transition intensities and q is the vector of limit probabilities. The elements of B have the following form:

$$b_{nm} = \begin{cases} \lambda_j a_j, & m = n + \delta_j \in N, \; j \in J \\ n_j \mu_j, & m = n - \delta_j \in N, \; j \in J \\ d, & m = n \\ 0, & \text{otherwise} \end{cases} \tag{12}$$

where the diagonal elements d are determined such that the sum of elements in each row of B equals zero. The actions a_j are the actions of the optimal policy, computed as described in the previous section.

The mean return time for the link, T_l, is the average of the individual mean return times weighted by their limiting probabilities and normalized:

$$T_l = \frac{\sum_{n \in N_{ib}} q_n T_{nn}}{\sum_{n \in N_{ib}} q_n} \tag{13}$$

For ease of implementation, the window of future returns may be measured in terms of a number of arrivals instead. This length L_j of this win-

dow for class j is computed by multiplying the mean return time by the class-j arrival rate and rounding off to the nearest integer:

$$L_j = \lambda_j T_l \tag{14}$$

and rounding off to the closest integer.

It can be observed that although this window size varies with λ_j, this variation is partly cancelled by T_l decreasing with increasing λ_j.

4.3 Neural Network-Based Prediction of Future Arrival Rates

The prediction of future arrival rates is naturally based on measures of recent arrival rates. In this work, the following representation of the history of the arrival process is used: for all classes $j \in J$, exponentially weighted running averages $h_j = (h_{j1}, ..., h_{jM})$ of the inter-arrival times are computed on different time scales. These *history vectors* are computed recursively using forgetting factors $\{\alpha_1, ..., \alpha_M\}$ taking values in the interval (0, 1):

$$h_{ji}(k) = \alpha_i\Big[t_j(k) - t_j(k-1)\Big] + (1-\alpha_i)h_{ji}(k-1)\,, \tag{15}$$

where $t_j(k)$ is the arrival time of the k-th call from class j.

In studies of time-series prediction, non-linear feed-forward neural networks outperform linear predictors on time series with long memory [20]. The two most common neural models for time-series prediction are the *multilayer perceptron* and *radial-basis function* (RBF) neural networks. Here, we employ RBF networks. In a RBF network having S hidden nodes, the hidden nodes compute their outputs $\phi_k(h_j), k = 1, ..., S$, in response to a history vector input:

$$\phi_k(h_j) = \exp\left[-\frac{1}{{\sigma_{kj}}^2}\sum_{i=1}^{M}\Big(h_{ji} - v_{kji}\Big)^2\right], \tag{16}$$

where v_{kji} are *location* parameters and σ_{kj} determine the *widths* of the Gaussian activation functions. These are manually distributed after inspection of the data sets to cover the history vector space.

The main reason for choosing RBF networks is this local activation of hidden layer basis functions. As the inputs to the network are weighted sums of call inter-arrival times, the input range will be large due to the heavy-tailed nature of the arrival processes. The major part of the input

values will be close to their mean value, and a few values will be very large. With locally activated basis functions in the network, the hidden nodes that respond to the rare large inputs are solely activated by these inputs, and the corresponding weights are trained. If, instead, multilayer perceptrons would be used, their (smooth) threshold-type functions would be activated in an unbounded region of the input space. This global activation imposes a risk of the neural network "forgetting" what it has learnt from the rare inputs, when trained on more frequent inputs. Further, RBF networks are linear in their free parameters (but not in their inputs), which speeds up learning compared to multilayer perceptrons.

The sum of activations of the hidden nodes in the RBF network are normalized to unity, to make the function produced by the network smoother. By putting a lower limit on the normalization coefficient Φ, the region where a the nodes are activated is limited:

$$\Phi = \max\left(\Phi_{act}, \Phi_{min}\right), \tag{17}$$

where

$$\Phi_{act} = \sum_{k=1}^{S} \phi_k . \tag{18}$$

The normalized activations of the hidden nodes are given by

$$\psi_k = \begin{cases} \dfrac{\phi_k}{\Phi}, & k \in \{1, ..., S\} \\ 0, & k = S + 1,\ \Phi_{act} \geq \Phi_{min} \\ 1 - \left(\Phi_{act}/\Phi_{min}\right), & k = S + 1,\ \Phi_{act} < \Phi_{min} \end{cases}, \tag{19}$$

where ψ_{S+1} is a default activation, that is non-zero outside the region of input space where the Gaussian hidden nodes are activated.

The output of the RBF network is the weighted sum of activations:

$$y_j = \sum_{k=1}^{S+1} w_k \psi_k(h_j) , \tag{20}$$

where w_k are the trainable weights of the network.

The network is trained with the average inter-arrival time as target. After every new arrival, the prediction error $\epsilon_j(k)$ is computed:

$$\epsilon_j(k) = \frac{\sum_{i=1}^{L_j} \left[t(k+i) - t(k+i-1) \right]}{L_j} - y_j(k). \quad (21)$$

In case of on-line learning, the updating must be delayed L_j time steps. The network weights are updated using the standard least mean squares (LMS) rule:

$$w_i(k+1) = w_i(k) - \eta\epsilon(k)\psi_i(h_k), \quad i \in \{1, ..., S+1\}, \ \eta \in (0,1), \quad (22)$$

where η is a step size parameter.

The predicted per-class inter-arrival rates $\hat{\lambda}_j(k) = y(k)^{-1}$ are then used to look up the control policy.

Given the prediction horizon and the arrival rate predictor, the forgetting factors $\{\alpha_1, ..., \alpha_M\}$ can be tuned by linear search to minimize the prediction error on sample traffic traces.

5 Temporal-Difference Learning Solution

In reinforcement learning, the value function and its associated policy are iteratively refined from sample rewards, in interaction with the controlled system. The reinforcement learning solution described here was previously presented in [11]. It uses Sutton's temporal-difference (TD(0)) learning rule [9] to learn the value function for the current policy, in parallel with improving the policy from the current value function. The value function approximates the infinite-horizon discounted reward of equation (4).

The design of the TD learning controller benefits from the tuning of forgetting factors in the previous section. A good history descriptor is useful both for prediction of future arrivals and for control. Consequently, the same forgetting factors are used for TD learning as for gain scheduling.

When the link admission controller takes an action a_j in a state $x = (n, h)$, the new system configuration x' that will occur *immediately after* the action is performed is

$$x' = \begin{cases} (n + a_j\delta_j, h), & a_j = 1, (n + \delta_j) \in N \\ (n, h) & \text{otherwise} \end{cases} \quad (23)$$

By learning values of such afterstates [9], the goodness of the feasible actions in a state can be compared. A "greedy" policy selects the action

that results in the afterstate that has the highest value. Henceforth, the afterstates are denoted with a prime (x'), to separate the from the standard definition of states (x).

Due to the continuous-valued history vectors, a parametric value function is needed. As for gain scheduling, RBF networks are employed. For each $n \in N$, a RBF network with MK inputs approximates the value function for this n: $V_n(h_{11}, ..., h_{MK})$.

The center and width parameters of the hidden nodes in this neural network are the same as in section 4.3, and the activations are computed in a similar way. A difference is that the neural network now has MK inputs, where the prediction network has only M inputs. Activations $\phi_{jk}(h_j)$, $k \in \{1, ..., S\}$are computed and normalized per class as in equation (19). Then the total activations are computed by taking the outer product of the normalized activations, resulting in S^K product activations. This is possible while maintaining a RBF network thanks to the the Gaussian being factorizable. Factorization of RBF networks is further discussed by Haykin [20]. Last, a hidden default activation is added to to the hidden layer analogously to equation (19), in case one or more class-wise activation vectors do not sum to unity.

The policy $\pi(x)$ is implicitly defined by the value function. During learning, actions are selected stochastically, with the highest probability for the action that results in the afterstate associated with the highest value, using epsilon-greedy action selection [9]. With a small probability, a random action is selected. As learning proceeds, the degree of randomness is successively lowered, until finally the policy deterministically selects the action that results in the highest expected after-state value. This trial-and-error search allows the controller to evaluate all actions and improve its policy.

After every decision, the value $V(x'_k)$ of the previous after-state is adjusted by updating the weights of the RBF network on the temporal-difference error, $\epsilon(k)$:

$$\epsilon(k) = \left[1 - e^{-\beta \Delta_k}\right] r_k + e^{-\beta \Delta_k} V(x'_{k+1}) - V(x'_k), \tag{24}$$

where $\Delta_k = t_{k+1} - t_k$. The LMS update rule for the weights of the RBF network that corresponds to the call number tuple n_k in x'_k gives:

$$w_i(k + 1) = w_i(k) + \eta \epsilon(k) \psi_i h(k), \quad i \in \left\{1, ..., S^2 + 1\right\}, \ \eta \in (0, 1) \tag{25}$$

The value function is updated on transitions caused by call arrivals as well as call departures. For obvious service reasons, call departures must always be accepted.

6 Simulation

The performance of the proposed link admission controllers was evaluated on a simulated communication link with capacity C = 24 [units/s], that was offered calls from the self-similar arrival processes. For comparison, we repeated the simulations with two other link admission controllers: one using a lookup table version of TD learning, and one using complete sharing, i.e. to accept a call if the free capacity on the link is sufficient. The OPNET Modeler simulation software [21] was used to model and simulate the link and its controllers.

6.1 Traffic Synthesis by The Fractional ARIMA Process

There exist many methods for synthesis of self-similar processes. For an overview, see for instance [5, 8]. In this paper, synthetic traffic traces are generated from a Gaussian fractional ARIMA (0, d, 0) model [8].

First, N_0 zero-expectance continuous values from a fractional ARIMA (auto-regressive integrated moving average) process Z_i are generated by a convolution formula:

$$Z_i = \sum_{j=0}^{N_0} c_{i-j} e_j, \quad i \in \{0, ..., N_0\}, \tag{26}$$

where e_j are the normally distributed innovations of the process, having mean 0 and variance 1. The coefficients c_j are given by

$$c_j = \frac{\Gamma(j + d)}{\Gamma(d)\Gamma(j + 1)}, \quad j \in \{0, ..., N_0\}, \tag{27}$$

where Γ is the standard gamma function. The c_j's can be computed recursively:

$$c_{j+1} = \frac{j + d}{j + 1} c_j, \quad c_0 = 1, \quad j \in \{0, ..., N_0\}. \tag{28}$$

This process is asymptotically self-similar, with Hurst parameter $H = d + 0.5$, $0 < d < 0.5$. The increments Z_i are then generated from

Z_i by adding a constant d, truncating negative values to zero, and rounding off to integers.

As the resulting time series tells the number of increments per time slot, it must be transformed to individual arrival times. This is done by distributing the number of arrivals per time slot uniformly within the same slot. Thus, the resulting traffic trace contains $(Z_0 + Z_1 + ... + Z_{N_0})$ arrival times. The choice of the duration of the time slot determines the intensity of the arrival process.

We generated synthetic traffic traces containing arrival/departure pairs from two call classes, characterized by bandwidth requirements $b_1 = 1$ (narrow-band) and $b_2 = 6$ (wide-band) [units/s] and call holding times with mean $1/\mu_1 = 1/\mu_2 = 1$ [s].

The per-class self-similar call arrival processes were generated from a fractional ARIMA process with Hurst parameter 0.85. The offset δ was set to 4.0, and the arrival rates were linearly scaled to make the expected long-term arrival rates λ_1 and λ_2 for the two classes fulfil

$$\frac{b_1\lambda_1}{\mu_1} + \frac{b_2\lambda_2}{\mu_2} = \gamma C, \tag{29}$$

where C is the capacity of the communication link and σ is the mean relative load. Traces with $\gamma = 1.0$, 1.2 and 1.5 were generated. The ratio λ_1/λ_2 was varied from 0.4 to 2.0.

6.2 Gain Scheduling

For simplicity, a constant prediction horizon was used throughout the simulations. This was computed according to section 4.2, with one difference: the average return time was averaged only for a subset of N_{ib}: the states where intelligent blocking was observed for any of the policies used in the gain scheduling controller. By averaging the resulting prediction windows for $\lambda_1/\lambda_2 = 0.4$, 1.0 and 2.0, $\gamma = 1$, a window size $L_1 = L_2 = 6$ was achieved.

The table of policies to be used for gain scheduling was computed for predicted $\hat{\lambda}_1$ and $\hat{\lambda}_2$ ranging from 0.5 to 15 with step size 0.5; in total 900 policies. The solution to the system of equations for relative values (6) as well as the other linear systems of equations (10, 11), were obtained by the LASPack package for solving sparse systems of linear equations [22], written in the C programming language.

The two rate-prediction neural networks both had 9 hidden nodes, including the default activation node. The networks' weights were initialized to 0.

6.3 Temporal-Difference Learning

The tabular TD learning controller assumes Poisson arrival processes. From this assumption, it follows that the call number tuples $n \in N$ constitute Markovian states. Consequently, the value function table stores only one value per n.

The discount constant β of both the TD learning controllers was set to 2.0. This value was chosen because it yields an approximate discounting of the value of the next state in the TD(0) update rule equation (24) of 0.9 — a typical value of the discount parameter in discrete-time reinforcement learning. It also worked well in simulations. The step size η was linearly lowered from 0.09 to 0.01 during training.

The TD learning controller employing neural networks used RBF networks (one per $n \in N$) with 65 hidden nodes, factorized to 8 nodes per call class, plus a default activation node. Its weights were initialized to favor acceptance of all calls in all states.

6.4 Numerical results

Both the TD learning controllers and the gain scheduling controller were allowed to adapt to the first 400 000 simulated arrivals of the traffic traces. The performance of all four methods was measured on the subsequent 400 000 arrivals.

Figure 1 shows graphs of throughput versus arrival rate ratio for γ = 1.0, 1.25 and 1.5. Each data point is the averaged throughput for 10 traffic traces. Gain scheduling (GS/RBF) achieves an insignificant throughput improvement (0.3% at maximum) compared to TD learning with RBF (RL/RBF) networks, up to 1.6% compared to tabular TD learning (RL/TBL), and up to 5.8% better than complete sharing (CS).

The difference in throughput between TD learning and complete sharing is highest for low arrival rate ratios. The reason for this is that the throughput increase by reserving bandwidth for high-rate calls is considerably higher than the loss of throughput from the blocked low-rate narrowband traffic.

Figure 2 shows the evolution of selected weights in the RBF networks of the gain scheduling controller and the TD learning controller. From the gain scheduling controller, the weights of the arrival rate predictor

for service class 2 are plotted. From the the TD learning controller, nine weights of the RBF network corresponding to the call number tuple $(n_1, n_2) = (6, 2)$ are plotted. These weights correspond to eight different class-2 center vectors, plus the default activation.

The majority of the weights of the gain scheduling RBF network seems to converge in a few thousand call arrivals, whereas the TD learning controller needs about 100 000 arrivals to converge, despite its higher learning rate. This is not surprising, as the 65 different RBF networks of the TD learning controllers split up the set of training data, so that a single neural network is updated much less frequently than a rate-predicting neural network in the gain scheduling controller. Secondly, the TD learning networks have 65 trainable weights each, whereas the gain scheduling neural networks have only 9 weights each. Smaller neural networks learn faster than large ones. Thirdly, the TD learning neural networks are trained on moving targets, due to the temporal-difference learning rule, the stochastic action selection and the changing policy. Although the learning speed of the TD learning controller may be increased by parametrizing the value function in a way which allows for generalization between neighboring call number tuples, the arguments above make it unlikely that this will make it as fast as predictive gain scheduling.

A few of the weights of the gain scheduling neural network change considerably even after long training. These are weights on the outputs of nodes that are activated by rare, large inter-arrival times.

7 Conclusion

We have presented predictive gain scheduling, a novel method for admission control of self-similar call traffic in multi-service networks, and compared it to a temporal-difference learning method designed for the same problem. The temporal-difference learning controller employs neural networks for value function approximation, whereas the predictive gain scheduling controller uses neural networks to predict future call arrival rates. In simulations under a range of traffic loads, both methods improve the link utilization compared to complete sharing and temporal-difference learning with lookup table storage of the value function.

The improvement of the link utilization is up to 1.6% compared to what is achieved by ignoring that the call traffic is self-similar, instead treating it as Poisson traffic. It remains to be investigated if the difference becomes larger for the network routing problem. This may be the case, as

routing decisions make use of the continuous-valued link gains, while CAC only uses the link policy (given by the sign of the gain).

The main difference between the neural network-based methods is that the predictive gain scheduling controller learns approximately 100 times faster than the temporal-difference learning controller. The reason for this is the decomposition of the control problem. The gain scheduling controller only needs to learn to predict future arrival rates of the call arrival processes. The value functions (and thereby policies) needed are computed in advance. This makes gain scheduling better suited for online learning.

The computational complexity of the two methods is similar and not very high during learning and decision making. The gain scheduling controller will however reach a computational bottleneck if the size of the state space is increased: the determination of optimal policies for Poisson traffic by dynamic programming (policy iteration). This can be overcome by state aggregation, as in [23], or by parametrizing the value function, as in [14], and using temporal-difference learning or approximate dynamic programming [10] to solve the problem for Poisson traffic.

In conclusion, predictive gain scheduling is a promising technique for admission control of self-similar call traffic.

References

[1] Z. Dziong, *ATM Network Resource Management*, McGraw-Hill, 1997.

[2] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, vol. 1, Athena Scientific, Belmont, Mass., 1995.

[3] H.C. Tijms, *Stochastic Modelling And Analysis - A Computational Approach*, John Wiley & Sons Ltd., 1986.

[4] V. Paxson and S. Floyd, "Wide-Area Traffic: The Failure of Poisson Modeling", in Proc. of SIGCOMM 94 –8/94, London, UK, pp. 257–268, 1994.

[5] W.E. Leland, M.S. Taqqu, W. Willinger and D.V. Wilson, "On the Self-Similar Nature of Ethernet Traffic (Extended Version)", *IEEE/ACM Transactions on Networking*, vol. 2, no. 1, pp. 1–15, Feb. 1994.

[6] A. Feldman, A.C. Gilbert, W. Willinger and T.G. Kurtz, "The Changing Nature of Network Traffic: Scaling Phenomena", *Computer Communication Review*, vol. 28, no. 2, pp. 5–29, April 1998.

[7] M.E. Crovella and A. Bestavros, "Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes", *IEEE/ACM Transactions on Networking*, vol. 5, no. 6, pp. 835-845, Dec. 1997.

[8] J. Beran, *Statistics for Long-Memory Processes*, Monographs on Statistics and Applied Probability 61, Chapman & Hall, 1994.

[9] R.S. Sutton and A.G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, Mass., 1998.

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

[11] J. Carlström and E. Nordström, "Reinforcement Learning for Control of Self-Similar Call Traffic in Broadband Networks", *Teletraffic Engineering in a Competitive World - Proceedings of The 16th International Teletraffic Congress* (ITC 16), pp. 571–580, Elsevier Science B.V., 1999.

[12] E. Nordström and J. Carlström, "A Reinforcement Learning Scheme for Adaptive Link Allocation in ATM Networks", in *Proceedings of the International Workshop on Applications of Neural Networks to Telecommunications 2* (IWANNT*95), Alspector, Goodman, Brown (eds.), Lawrence Erlbaum, pp. 88–95, 1995.

[13] Z. Dziong and L. Mason,"Call Admission Control and Routing in Multi-service Loss Networks", *IEEE Transactions on Communications*, vol. 42, no. 2. pp. 2011–2022, Feb. 1994.

[14] P. Marbach, O. Mihatsch and J.N. Tsitsiklis, "Call Admission Control and Routing in Integrated Service Networks Using Neuro-Dynamic Programming", *IEEE Journal on Selected Areas in Communications*, Feb. 2000.

[15] E. Nordström, J. Carlström. O. Gällmo, L. Asplund, "Neural Networks for Adaptive Traffic Control in ATM Networks", *IEEE Communications Magazine*, vol. 33 no. 10, pp. 43–49, Oct. 1995.

[16] H. Tong and T. Brown, "Adaptive Call Admission Control Under Quality of Service Constraints: A Reinforcement Learning Solution", *IEEE Journal on Selected Areas in Communications*, Feb. 2000.

[17] I. Norros, "On the Use of Fractional Brownian Motion in the Theory of Connectionless Networks", *IEEE Journal on Selected Areas in Communication*, vol. 13, no. 6, pp. 953–962, Aug 1995.

[18] K.J. Åström and B. Wittenmark, *Adaptive Control*, 2nd ed., Addison-Wesley, 1995.

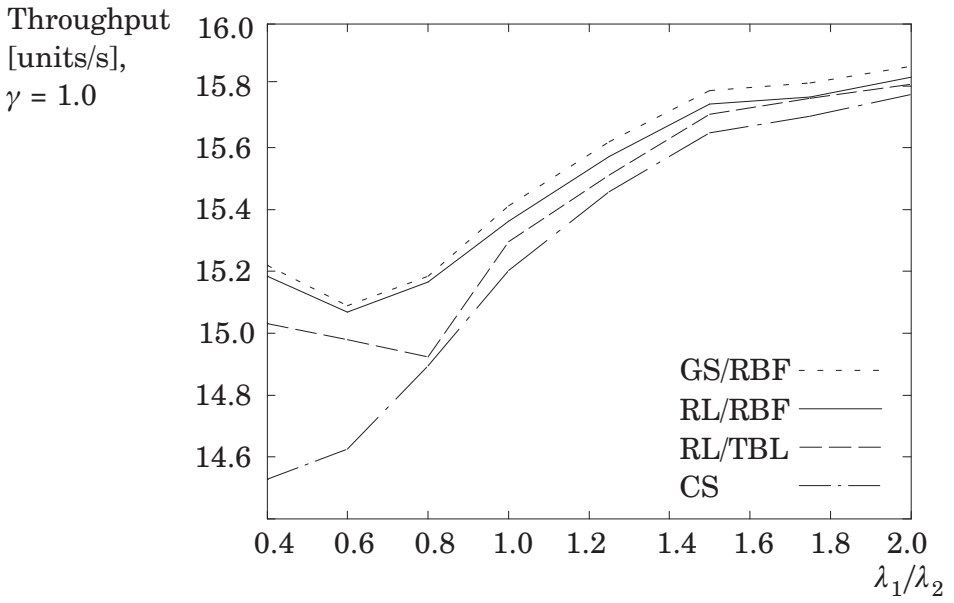
[19] E. Nordström, *Markov Decision Problems in ATM Traffic Control*, PhD. Thesis, Technical Report DoCS 98/100, Department of Computer Systems, Uppsala University, Uppsala, Sweden, 1998.

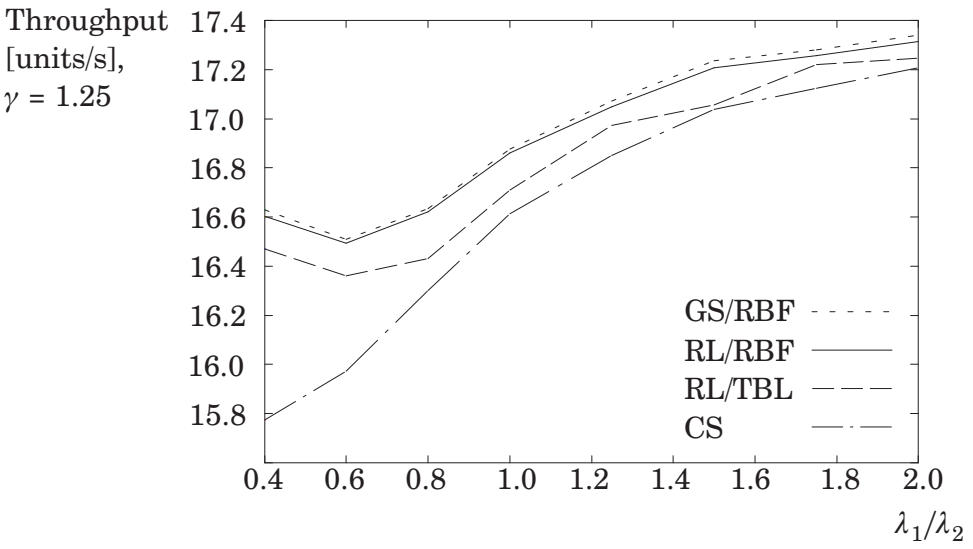
[20] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed., Macmillan College Publishing Co., Englewood Cliffs, NJ, 1999.

[21] *OPNET Modeler documentation*, OPNET, Inc., Washington, D.C., 1999.

[22] T. Skalicky, *LASPack Reference Manual*, `http://www.math.tu-dresden.de/~skalicky/laspack/`

[23] K.R. Krishnan and H. Huebner-Sabzo de Bucs, "Admission Control and State-Dependent Routing for Multi-Rate Circuit-Switched Traffic", *Teletraffic Contributions for the Information Age - Proceedings of the 15th International Teletraffic Congress* (ITC 16), pp. 1043–1054, Elsevier Science B.V.,1999.





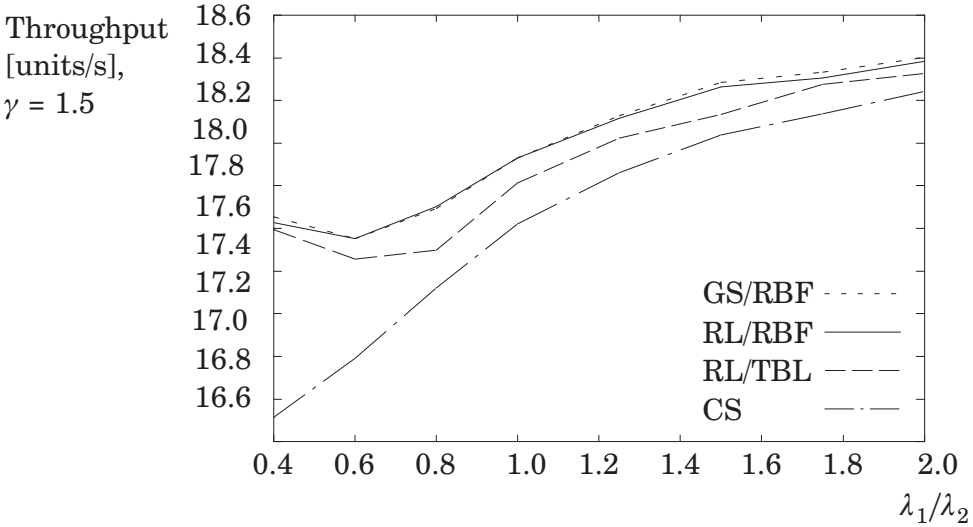


Figure 1: Throughput versus arrival rate ratio

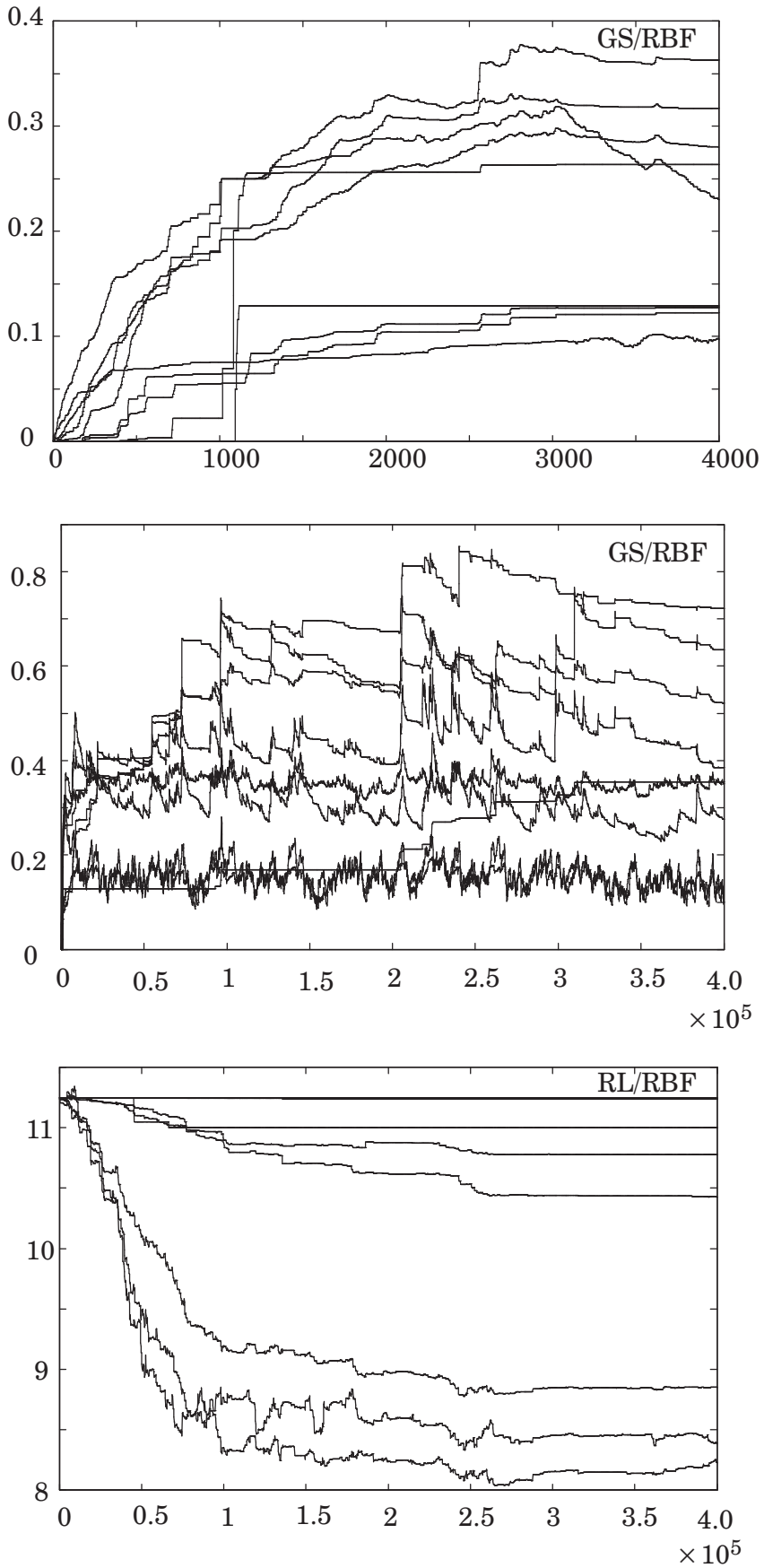


Figure 2: RBF neural network weights versus number of call arrivals