A FUZZY DYNAMIC BANDWIDTH RE-ALLOCATOR

Dean Slonowsky, * Qingshan Jiang, ** Raj Srinivasan
Department of Statistics & Institute of Industrial Mathematical Sciences,
University of Manitoba, Winnipeg, Manitoba, Canada R3T 2N2
**Manifold Data Mining Incorporated,*
501 Alliance Avenue, Suite 205, Toronto, Ontario, Canada M6N 2J1
***Department of Mathematics & Statistics,*
University of Saskatchewan, Saskatoon, Saskatchewan, Canada S7N 5E6
dean_slonowsky@umanitoba.ca, jiang@manifolddatamining.com, raj@math.usask.ca

Abstract

A high-speed network will typically assign each admitted connection a fixed bandwidth, somewhere between its mean and peak transmission rate. As an alternative, we develop a novel scheme for re-assigning network resources periodically on the basis of current network conditions. Our algorithm, which employs the theory of fuzzy logic control, is computational efficient, making frequent bandwidth re-allocations over relatively small time intervals feasible. In situations where traffic intensities change drastically over short time intervals, our algorithm, tested against both static and non-fuzzy bandwidth re-allocation schemes, significantly lowers data loss while increasing network efficiency.

Keywords: *High-speed networks; dynamic bandwidth re-allocation; fuzzy logic control.*

1 Introduction

We begin by describing a fairly general network. In particular, consider the situation depicted in Fig. 1, where N individually buffered traffic sources are serviced by a common output link. This represents only a portion of the overall network in the sense of both space and time. In particular, we are only considering the time interval, I over which these N connections are active. "In space" refers to the fact that, once they are multiplexed, these N sources could constitute a single aggregate source in a larger hierarchical buffered network (see [4]).

Upon being admitted to the network, each of these N sources is allocated a portion of the available bandwidth, proportional to their service requirements. Depending on the application, there are various methods by which to accomplish this *bandwidth allocation*. The simplest is *peak rate allocation* whereby each source is allotted a bandwidth as great as its maximum rate of transmission.

While being the most conservative with respect to network reliability (i.e., with respect to data loss), peak rate allocation can be very inefficient with respect to bandwidth utilization if the traffic is "bursty", having pronounced on and off periods, for example. In these cases, a significant improvement in network efficiency can be made by allocating bandwidths somewhere between the mean and peak rates. Since our sources are buffered, this does not necessarily compromise the reliability of the network. This is precisely the idea behind the *effective bandwidth* approach (see [4]), where bandwidth is allocated so as to guarantee a pre-specified probability of data loss to each individual source.

To make further gains in overall network performance, we propose a novel and efficient *dynamic bandwidth re-allocation* (DBR) scheme which adaptively re-adjusts the bandwidth among the N sources on the basis of on-line measurements. The prefix "re" refers to the fact that we are neither admitting new connections nor are we adjusting the total bandwidth, C dedicated to the current N connections. Our scheme operates at the sub-connection admission level.

In Section 2, we develop a DBR scheme for a two buffer system, employing a fuzzy logic controller, the inputs of which are certain buffer statistics. The case of $N > 2$ buffers can be handled in a recursive manner (Section 4). In Section 3, our fuzzy DBR is tested against several non-fuzzy alternatives, including the static scheme, which corresponds to the standard case of fixed bandwidths. A marked reduction in data loss and improved link utilization over the static scheme is consistently observed in our simulations. Furthermore,

Proceedings of the 2002 IEEE Canadian Conference on Electrical & Computer Engineering
0-7803-7514-9/02/$17.00 © 2002 IEEE

our fuzzy DBR yields significant improvements (over all alternatives) in situations where individual traffic intensities change drastically over time intervals much shorter than inter-connection times.

2 The Two-Buffer Case

In this section, we analyze a special case of the situation depicted in Fig. 1. In particular, we consider the

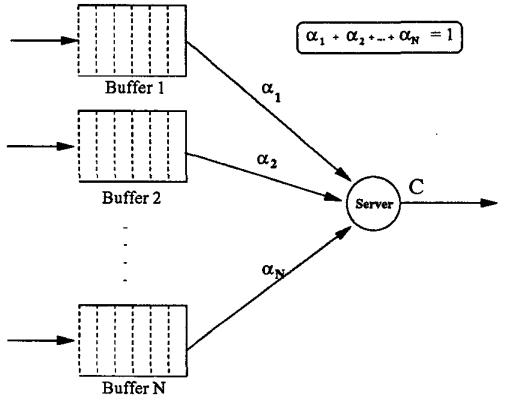

Figure 1: N buffered traffic sources sharing a single output link.

case where $N = 2$. Here, two sources, denoted TS1 and TS2, pass though buffers 1 and 2, respectively. At any point in time, $\alpha_1\, C$ bandwidth is allocated to TS1 and $\alpha_2\, C$ bandwidth is allocated to TS2. Since $\alpha_1 + \alpha_2 = 1$, we define $\alpha = \alpha_1$ so that $\alpha_2 = 1 - \alpha$. Our main goal is to devise a DBR mechanism using fuzzy logic which makes frequently updates of α based strictly on buffer statistics, in particular, buffer contents. A DBR for the case of $N > 2$ will be discussed in Section 4.

For the sake of context, we will describe our network as a *packet switched network*. Conventionally, the server will serve one *packet* of data during one *time slot*. For example, for an *asynchronous transfer mode* (ATM) line with a capacity of 155.52 Mbits/second, since each packet ("cell") contains 53 bytes, the duration of a time slot is approximately 2.762 microseconds (see [5]). Therefore, it may be impractical to update α after every time slot. Instead, we will only make updates at certain *epoch times*, $T_1, T_2, \cdots$. In this paper, we consider the *periodic case*, where $T_0 = 0$ and

$$T_k - T_{k-1} \;=\; L \text{ time slots}$$

for each $k = 0, 1, 2, \cdots$, some constant L. A fuzzy DBR incorporating *adaptively determined epoch times* will be studied in a future publication.

Adopting the conventions in [5], in any given time slot, first the packets arrive into the buffer(s) then, according to some scheduling scheme (for example, weighted round-robin), the server serves exactly one packet from either buffer 1 or buffer 2. If L is a large number, the temporal separation between consecutive time slots is microscopic with respect to the $\{T_k : k = 1, 2, \ldots\}$ time scale. In this *fluid scale*, traffic essentially "flows" into the buffers and flows out the output link. See Fig. 2.

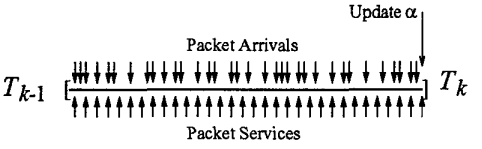

Figure 2: The major events in the time interval $[T_{k-1}, T_k]$.

Remark. A smaller value of L implies more updates of α which in turn implies improved performance of the network. An essential lower bound on L is the time (measured in time slots) necessary to compute the updated value of α. In this case, the α update calculated using buffer statistics measured at time T_{k-1} can be used no earlier than time $T_k = T_{k-1} + L$.

Define

$$B_i^{(k)} \;=\; \text{content of buffer } i \text{ measured at epoch time } T_k \quad (i = 1 \text{ and } 2)$$

and

$$\alpha^{(k)} \;=\; \text{proportion of bandwidth allocated to buffer 1 during } [T_k, T_{k+1}).$$

If we update α on the basis of buffer contents alone, then, in accordance with the previous remark, we have the relation

$$\alpha^{(k)} = g(B_1^{(k-1)}, B_2^{(k-1)}); \quad k = 1, 2, 3, \cdots$$

where g denotes the functional form of the DBR. We now focus on an explicit construction of g via fuzzy logic.

2.1 A fuzzy logic controller

In this subsection we construct a *Sugeno-type fuzzy logic controller* (SFLC) to serve as a DBR. Our choice

of the fuzzy logic methodology is motivated by earlier successful applications to different but related problems (see [1], [6]). What follows is a self-contained description of our particular SFLC. A more general discussion on the topic can be found in [7], for example.

Define the input variable,

$$\mathrm{P} = \frac{\text{contents of buffer 1}}{\text{capacity of buffer 1}} - \frac{\text{contents of buffer 2}}{\text{capacity of buffer 2}}.$$

The operation of our SFLC follows four steps: *measurement, fuzzification, inference* and *defuzzification.* We describe each in detail.

1. Measurement: Measure the value of P from the system. Note that P always lies in the interval $[-1, 1]$. For this reason, P is said to have $[-1, 1]$ as its *universe of discourse.*

2. Fuzzification: If our goal is to reduce overall data loss, then by simply observing the value of P, we get an idea of what change should be made to α. For example, if P is very negative, it is clear that buffer 2 requires more service than buffer 1 (an intuition which is validated by our simulations). As it stands, such a statement is subjective and inexact.

To model the inexactness of the statements, "P is very negative", "P is close to zero" and "P is very positive", we employ the notion of *fuzzy sets*, each uniquely determined by a *fuzzy membership function*, f. Rather than assigning a "black-or-white", "0-or-1"-type response, fuzzy membership functions permit a continuous range of *membership grades.* In our application, we define three fuzzy sets, $f_{\mathrm{V.Neg.}}$, f_{Zero} and $f_{\mathrm{V.Pos.}}$, intended to model three *linguistic values* of P: "P is very negative", "P is close to zero" and "P is very positive", respectively. Explicitly, we define

$$f_{\mathrm{V.Neg.}}(\mathrm{P}) = \begin{cases} 1 & , \text{if } \mathrm{P} < -0.5 \\ -2\mathrm{P} & , \text{if } -0.5 \le \mathrm{P} \le 0 \\ 0 & , \text{if } \mathrm{P} > 0 \end{cases}$$

$$f_{\mathrm{Zero}}(\mathrm{P}) = \begin{cases} 1 - 2|\mathrm{P}| & , \text{if } -0.5 \le \mathrm{P} \le 0.5 \\ 0 & , \text{otherwise} \end{cases}$$

$$f_{\mathrm{V.Pos.}}(\mathrm{P}) = \begin{cases} 0 & , \text{if } \mathrm{P} < 0 \\ 2\mathrm{P} & , \text{if } 0 \le \mathrm{P} \le 0.5 \\ 1 & , \text{if } \mathrm{P} > 0.5\,. \end{cases}$$

They are depicted in Fig. 3. (Note that each has a maximum value of 1 and a minimum value of 0.)

For obvious reasons, $f_{\mathrm{V.Neg.}}$ and $f_{\mathrm{V.Pos.}}$ are examples of *trapezoidal membership functions* whereas f_{Zero} is an example of a *triangular membership function.* Their simple forms lead to the quick computation of membership grades.

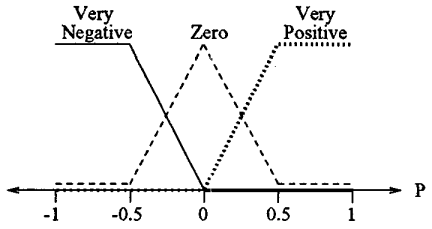

Figure 3: The fuzzy sets $f_{\mathrm{V.Neg.}}$, f_{Zero} and $f_{\mathrm{V.Pos.}}$ corresponding to the linguistic values of P.

Now, the act of *fuzzifying* the crisp input P involves determining the membership grades, $f_{\mathrm{V.Neg.}}(\mathrm{P})$, $f_{\mathrm{Zero}}(\mathrm{P})$ and $f_{\mathrm{V.Pos.}}(\mathrm{P})$. Note from Fig. 3 that for any input, at most two of the three fuzzy sets will return a non-zero membership grade. Furthermore, note that

$$f_{\mathrm{V.Neg.}}(\mathrm{P}) + f_{\mathrm{Zero}}(\mathrm{P}) + f_{\mathrm{V.Pos.}}(\mathrm{P}) = 1 \qquad (1)$$

for any input P in $[-1, 1]$.

3. Inference: So far, we have made a crisp observation P and have determined its membership grades with respect to the three linguistic values of P. Now, we must decide how to adjust α on the basis of these values. This is a two-step process, the first of which is termed *inference.* The mechanism for this inference is the *inference engine* which makes use of a *fuzzy rule base.* In our particular application, the rule base has three fuzzy rules:

R_1 : IF P is very negative, THEN α is small
R_2 : IF P is close to zero, THEN α is medium
R_3 : IF P is very positive, THEN α is large.

Before discussing the evaluation of these fuzzy rules, note that their forms are consistent with the control objective. For example, if P is negative, then the relative contents of buffer 2 exceeds the relative contents of buffer 1, implying a need to improve service to buffer 2, i.e., the value of α needs to be lowered. R_2 and R_3 have similar interpretations.

Given the crisp input P, the inference engine evaluates each of the rules, R_1, R_2 and R_3. We illustrate this evaluation for R_1. First, define the *antecedent part* of R_1 to be

$$\mathrm{An}(\mathrm{R}_1) = \text{"P is very negative"}.$$

Next, determine the *truth value*, $\mathrm{Tr}(\mathrm{An}(\mathrm{R}_1))$ of $\mathrm{An}(\mathrm{R}_1)$. In our particular case,

$$\mathrm{Tr}(\mathrm{An}(\mathrm{R}_1)) = f_{\mathrm{V.Neg.}}(\mathrm{P}).$$

If $\mathrm{Tr}(\mathrm{An}(\mathrm{R}_1)) > 0$, R_1 is said to have *fired*.

In view of the role α plays in the switching node, it is natural to consider the value $\alpha_{\mathrm{Low}} = 0.1$ to be small, $\alpha_{\mathrm{Med.}} = 0.5$ to be medium and $\alpha_{\mathrm{High}} = 0.9$ to be large. Hence, it is natural to assign the value

$$\mathrm{Val}(\mathrm{R}_1) \;=\; f_{\mathrm{V.Neg.}}(\mathrm{P}) \cdot \alpha_{\mathrm{Low}}$$

to the rule R_1. Similarly, we set $\mathrm{Val}(\mathrm{R}_2) = f_{\mathrm{Zero}}(\mathrm{P}) \cdot \alpha_{\mathrm{Med.}}$ and $\mathrm{Val}(\mathrm{R}_3) = f_{\mathrm{V.Pos.}}(\mathrm{P}) \cdot \alpha_{\mathrm{High}}$. In total, the output of the inference engine is the vector, $\langle \mathrm{Val}(\mathrm{R}_1), \mathrm{Val}(\mathrm{R}_2), \mathrm{Val}(\mathrm{R}_3) \rangle$.

4. Defuzzification: In our SFLC, the defuzzification step transforms the output of the inference engine into a crisp control output, α. Under the *weighted average scheme*, this value is given by

$$\alpha = \frac{f_{\mathrm{V.Neg.}}(\mathrm{P})\cdot\alpha_{\mathrm{Low}} + f_{\mathrm{Zero}}(\mathrm{P})\cdot\alpha_{\mathrm{Med.}} + f_{\mathrm{V.Pos.}}(\mathrm{P})\cdot\alpha_{\mathrm{High}}}{f_{\mathrm{V.Neg.}}(\mathrm{P}) + f_{\mathrm{Zero}}(\mathrm{P}) + f_{\mathrm{V.Pos.}}(\mathrm{P})}$$

$$= \mathrm{Val}(\mathrm{R}_1) + \mathrm{Val}(\mathrm{R}_2) + \mathrm{Val}(\mathrm{R}_3) \quad \text{[by (1)]}.$$

The SFLC determined by Steps 1 through 4 is a special case of a *fuzzy P-type controller* (see [7]). A general P-type controller takes, as input, the deviation or "proportion" that the system output differs from the target. In our case, the target output is a zero difference in relative buffer contents.

A *fuzzy PD-type controller*, as defined in [7], has also been constructed and tested by the authors. No significant gain in performance over the above P-type controller was observed. What is more, the rule base for a PD-controller has three times as many rules as does the related P-type controller, making the inference step more computationally intensive.

2.2 The fuzzy DBR algorithm

With the SFLC now constructed, the periodic updates of α go as follows:

(A): At time $T = 0$, fix an initial value $\alpha^{(0)}$ (for example, the originally allocated bandwidth).

(B): At time T_{k-1}, record the buffer contents, $B_1^{(k-1)}$ and $B_2^{(k-1)}$ and set

$$\mathrm{P}_{k-1} \;=\; \frac{B_1^{(k-1)}}{B_1^{\mathrm{Cap.}}} - \frac{B_2^{(k-1)}}{B_2^{\mathrm{Cap.}}}$$

where $B_i^{\mathrm{Cap.}}$ represents the storage capacity of buffer i ($i = 1, 2$).

(C): In the time interval $[T_{k-1}, T_k)$, while serving buffers 1 and 2 at the rates $\alpha^{(k-1)}C$ and $(1 - \alpha^{(k-1)})C$, respectively, calculate the updated value, $\alpha^{(k)}$ by applying the SFLC in Section 2.1 to the crisp measurement P_{k-1}. By assumption, this calculation takes no longer than $T_k - T_{k-1}$ time slots to perform.

(D): At time T_k, replace the parameter $\alpha^{(k-1)}$ by the update, $\alpha^{(k)}$.

(E): Return to (B), replacing k by $k + 1$.

Remarks. (a) As revealed through numerous simulations, the above controller adapts quickly to changes. Hence, even a poor choice for the initial value $\alpha^{(0)}$ will not have a negative effect on performance.

(b) In general, α is adjusted so as to keep the relative contents of both buffers equal, the goal being an improvement in overall system reliability. To achieve a more specialized control objective, one can weight the two terms in the definition of P_{k-1} accordingly.

(c) Recall that the two sources (N sources in general) are only assigned a fixed portion, C of the total bandwidth, C_{Total}. With minor changes, our fuzzy DBR algorithm can also be applied to the distribution of the *excess bandwidth*, $C_{\mathrm{Total}} - C$ (see [2]).

2.3 Non-fuzzy alternatives

To test the effectiveness of our fuzzy dynamic bandwidth re-allocator (F-DBR), we will compare its performance in simulations against alternate DBR schemes. The operation of the two-buffer system under any of these schemes follows Steps (A) through (E) with Step (C) changed accordingly.

The Static Scheme: This is the standard situation where the value of α remains fixed for the duration of the connection. It is represented by $\alpha^{(k)} = \alpha^{(0)}$, $k = 1, 2, 3, \ldots$

The Proportional DBR (P-DBR): This DBR is determined by the formula

$$\alpha^{(k)} \;=\; \frac{B_1^{(k-1)}}{B_1^{\mathrm{Cap.}}} \cdot \left[\frac{B_1^{(k-1)}}{B_1^{\mathrm{Cap.}}} + \frac{B_2^{(k-1)}}{B_2^{\mathrm{Cap.}}} \right]^{-1}.$$

Here, the amount of bandwidth allocated to a buffer is proportional to its relative contents. Note that

$$1 - \alpha^{(k)} \;=\; \frac{B_2^{(k-1)}}{B_2^{\mathrm{Cap.}}} \cdot \left[\frac{B_1^{(k-1)}}{B_1^{\mathrm{Cap.}}} + \frac{B_2^{(k-1)}}{B_2^{\mathrm{Cap.}}} \right]^{-1}.$$

The Linear DBR (L-DBR):

$$\alpha^{(k)} \;=\; 1/2 \cdot \mathrm{P}_{k-1} + 1/2;$$

a linear approximation to the F-DBR.

3 Simulation Results

To compare the effectiveness of the above DBRs, simulations were run with diverse traffic sources using MATLAB Version 5.3. Borrowing from the language of ATM networks, the adopted units for the data in these traces is taken to be *cells*. Just the same, our simulations are not intended to model any particular data network; their primary purpose is to illustrate the advantages that our fuzzy DBR offers over static and, in some cases, other non-fuzzy schemes. The traces we used fall into three categories:

On/Off: These sources have *on-phases*, during which data is sent at a constant cell rate, and *off-phases* during which no data is sent. The lengths of all phases, both on and off, are independent and geometrically distributed with rates λ_1 and λ_2, respectively. This leads to the parameterization,

$$\text{OnOff}\,(\mu_1, \mu_2 \,|\, \gamma)$$

where $\mu_1 = \lambda_1^{-1}$ is the mean on-time, $\mu_2 = \lambda_2^{-1}$ is the mean off-time and γ is the (constant) peak on-rate. Such *Markov modulated* traffic is frequently encountered in telephony (see [5]).

On/Off with a Trend: These sources also have on- and off-phases, but do not have the same cell rate in all on-phases. Instead, for each subsequent on-phase, the transmission rate is increased by adding a constant factor. These sources can be parameterized as

$$\text{OnOff}_{\text{Trend}}\,(\mu_1, \mu_2 \,|\, \gamma, c)$$

where γ denotes the cell rate in the initial on-phase, c denotes the amount of increase per on-phase and λ_1 and λ_2 are as defined above. Note that the cell rate in the n^{th} on-phase is $\gamma + (n-1)\,c$.

Flattened Star Wars: The Star Wars data set, produced by Garrett and Vetterli at Bellcore (see [3]) contains 171 000 frames of MPEG-1 data encoding the movie Star Wars. Each frame represents 1/24 of a second of actual video. To generate realistic (although somewhat artificial) data sets for the purpose of testing our DBRs, we consider sub-traces starting at an initial frame f_0 and ending at a terminal frame f_1 where the units of data in each frame are scaled down by dividing by a constant factor ρ. The resulting data sets can be parameterized as

$$\text{FSW}(f_0, f_1, \rho)\,.$$

These traces do not have off-phases.

We now describe two traffic scenarios used to compare the above DBR schemes. In both, we take the overall *cell loss ratio*,

$$\text{CLR} = \frac{\#\text{ of cells lost}}{\#\text{ of cells sent}}$$

and *link utilization*—the proportion of time the server is busy—as performance measures. (The lower the CLR and the higher the link utilization, the better.) A cell is lost if it arrives to a full buffer.

3.1 Traffic scenario I

Here, TS1 is OnOff(20, 40 | 50) and TS2 is OnOff(40, 20 | 50). Thus, while both sources have a peak rate of 50 cells per time unit, the lengths of the on-phases for TS2 are twice as long as those for TS1 whereas TS1 has off-phases which are, on average, twice as long as those for TS2. Both traces are 20 000 time units long and are independent of each other.

In Fig. 4 (a), by fixing the link capacity at 80 cells per time unit (implies "time unit" = 80 time slots) and varying $B_1^{\text{Cap.}} = B_2^{\text{Cap.}}$ from 250 to 500 cells, we obtain a graph of common buffer capacity versus $\log_{10}(\text{CLR})$. Here, we update α every $L = 80$ time slots. On the other hand, Fig. 4 (b) contains a plot of link capacity versus link utilization where $B_1^{\text{Cap.}} = B_2^{\text{Cap.}} = 500$ cells. The value of $\alpha^{(0)} = 0.4$ is used in the static scheme since this value delivers the lowest CLR for the given traces (determined via repeated simulations). The following conclusions can be drawn: In terms of both CLR and link utilization,

- The performance of the P-DBR and the F-DBR are comparable. Indeed, with respect to CLR (Fig. 4 (a)), F-DBR is just slightly better than P-DBR for medium buffer capacities whereas P-DBR is slightly better than F-DBR for large buffer capacities (> 300 cells).
- The performance of L-DBR is worse than both P-DBR and F-DBR.
- The performance of the static scheme is, by far, worse than any of the DBRs.

3.2 Traffic scenario II

Here, TS1 is $\text{OnOff}_{\text{Trend}}(50, 100 \,|\, 10, 1)$ and TS2 is FSW(40 000, 60 000, 500). These are two drastically different traffic sources, with TS2 being strongly non-homogeneous over time. The cell rate of TS1 increases steadily from 10 to 142 cells per time unit whereas TS2

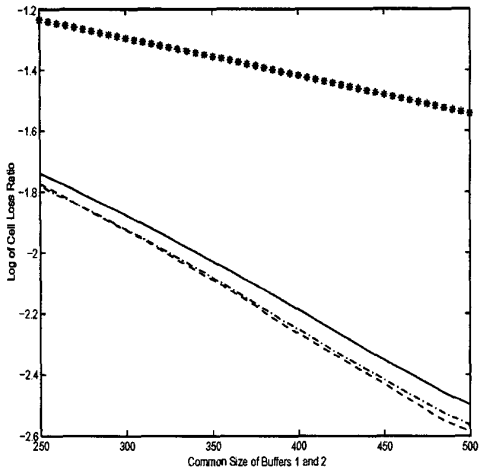

Figure 4: Results for scenario I: (a) buffer size versus $\log_{10}$(CLR); (b) link capacity versus utilization. KEY: $*$ $\equiv$ static scheme; — $\equiv$ L-DBR; -- $\equiv$ P-DBR; -· $\equiv$ F-DBR.

fluctuates from 46 to 116 with a mean rate of 74.3 cells per time unit.

In Fig. 5 (a), the link capacity is fixed at 160 cells per time unit (implies "time unit" = 160 time slots) and we update α every $L = 160$ time slots. In Fig. 5 (b) the maximum capacity of buffers 1 and 2 is fixed at 500 cells each. In this scenario, the optimal value of $\alpha^{(0)}$ for the static scheme is 0.6. Whereas the static scheme continues to perform poorly in comparison to the three DBRs, the conclusions we draw here are quite different from those of scenario I. In particular, from Fig. 5:

- Whereas the linear, proportional and fuzzy DBRs perform equally well with respect to link utilization, our fuzzy DBR performs significantly better than the remaining two DBRs in terms of CLR reduction. In fact, the $\log_{10}$(CLR) graph has the steepest slope under the F-DBR scheme. Therefore, under the F-DBR scheme, the CLR decreases to zero (as a function of buffer size) exponentially faster than under any of the other schemes.

- Unlike scenario I, the L-DBR slightly out-performs the P-DBR with respect to CLR reduction.

4 The N-Buffer Case

We now consider the general situation depicted in Fig. 1. First note that the P-DBR defined in Section 2.3 extends automatically to the N-buffer case. In particular, for each $i = 1, 2, \cdots, N$, we define

$$\alpha_i^{(k)} = \frac{B_i^{(k-1)}}{B_i^{\text{Cap.}}} \cdot \left[\sum_{j=1}^{N} \frac{B_j^{(k-1)}}{B_j^{\text{Cap.}}} \right]^{-1} \qquad (2)$$

where $B_i^{(k-1)}$ represents the contents of buffer i measured at time T_{k-1} and $\alpha_i^{(k)}$ denotes the proportion of C assigned to buffer i during time interval $[T_k, T_{k+1})$. On the other hand, the construction of the fuzzy DBR in Section 2.2 relied heavily on the special relation between α_1 and α_2, namely, $\alpha_2 = 1 - \alpha_1$; a relation only present in the $N = 2$ case. In this section, we propose two fuzzy DBR schemes for the case of $N > 2$ buffers.

4.1 The binary tree F-DBR

This fuzzy scheme employs a recursively defined fuzzy DBR whose generated proportions, $\alpha_i^{(k)}$ can be represented by branches of a binary tree. Here, we replace Steps (B) and (C) in the two-buffer F-DBR algorithm (Section 2.2), by the following steps:

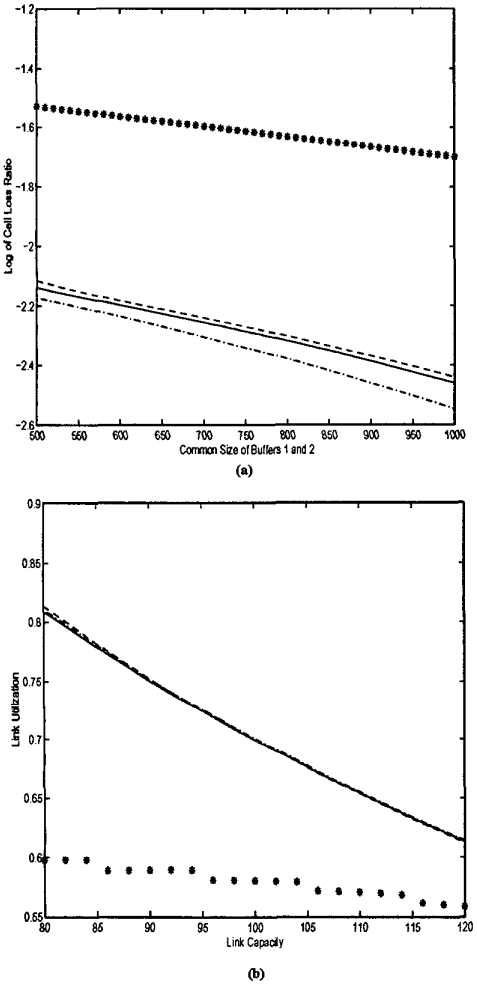

Figure 5: Results for scenario II: (a) buffer size versus $\log_{10}$(CLR); (b) link capacity versus utilization. KEY: $*$ $\equiv$ static scheme; — $\equiv$ L-DBR; -- $\equiv$ P-DBR; -· $\equiv$ F-DBR.

1) Split the N buffers into two groups, G_1 and G_2, preferably, but not necessarily of the same size. For example, if $N = 7$, we can take $G_1 = \{1, 2, 3\}$ and $G_2 = \{4, 5, 6, 7\}$. This determines an initial *rooted binary tree*, with the server as the root, and two leaves, G_1 and G_2.

2) Treating each group as a single buffered source, measure the difference,

$$\mathrm{P}_{k-1} = \frac{\sum_{j\in G_1} B_j^{(k-1)}}{\sum_{j\in G_1} B_j^{\mathrm{Cap.}}} - \frac{\sum_{j\in G_2} B_j^{(k-1)}}{\sum_{j\in G_2} B_j^{\mathrm{Cap.}}}$$

and, using P_{k-1} as the input, calculate $\alpha^{(k)}$ using the SFLC in Section 2.1. (Hence, we will allocate $\alpha^{(k)}C$ bandwidth to group G_1 and $(1-\alpha^{(k)})C$ bandwidth to group G_2.) In the binary tree, label the edge corresponding to G_1 with this value of $\alpha^{(k)}$ and the edge corresponding to G_2 with the value $1-\alpha^{(k)}$.

3) For each group, G (in the previous step) containing more than one buffer, we split G into two groups of buffers, which we generically denote G_1 and G_2. For example, if $N = 7$ and $G\,(= G_1) = \{1, 2, 3\}$, we can take $G_1 = \{1\}$ and $G_2 = \{2, 3\}$. For each such pair, (G_1, G_2), calculate the corresponding P_{k-1} and $\alpha^{(k)}$ as done in the previous step. Then, update the binary tree by adding two leaves (hence, two edges), G_1 and G_2 to each such leaf G and label these edges as done in the previous step.

4) Repeat this "splitting of groups" process until there are no groups left with more than one buffer.

This process of splitting groups generates a labeled binary tree, the leaves of which represent the N buffers themselves. Each proportion, $\alpha_i^{(k)}$ is then defined to be the product of the values labeling the edges on the unique path from buffer i to the server node ("root"). Clearly, $\sum_{i=1}^{N} \alpha_i^{(k)} = 1$.

For these updated proportions to be of any use, the above calculations must be completed by time T_k. Thus, as the number of connections increases, the value of L must increase accordingly. Indeed, we need to apply the SFLC in Section 2.1 $O(N \log_2 N)$ times.

4.2 The pairing F-DBR

In terms of both description and implementation, a simpler F-DBR for an N-buffer system goes as follows. If N is even, split the buffers into pairs, $\{1, 2\}, \{3, 4\}, \cdots, \{N-1, N\}$. Treating each pair as single buffered source (as done in Section 4.1), apply

the P-DBR in Equation (2), denoting the respective $\frac{N}{2}$ proportions by $\beta_1^{(k)}, \beta_2^{(k)}, \cdots, \beta_{\frac{N}{2}}^{(k)}$. For each pair j ($j = 1, 2, \cdots, \frac{N}{2}$), apply the SFLC in Section 2.1 to the corresponding two-buffer system, denoting the output by $\gamma_j^{(k)}$. We then define the bandwidth proportions by

$$\begin{aligned}
\alpha_1^{(k)} &= \gamma_1^{(k)} \cdot \beta_1^{(k)}, \\
\alpha_2^{(k)} &= (1 - \gamma_1^{(k)}) \cdot \beta_1^{(k)}, \\
\alpha_3^{(k)} &= \gamma_2^{(k)} \cdot \beta_2^{(k)}, \\
\alpha_4^{(k)} &= (1 - \gamma_2^{(k)}) \cdot \beta_2^{(k)}, \\
&\vdots \quad\quad \vdots \\
\alpha_{N-1}^{(k)} &= \gamma_{\frac{N}{2}}^{(k)} \cdot \beta_{\frac{N}{2}}^{(k)}, \\
\alpha_N^{(k)} &= (1 - \gamma_{\frac{N}{2}}^{(k)}) \cdot \beta_{\frac{N}{2}}^{(k)}.
\end{aligned}$$

If N is odd, we simply treat the N^{th} buffer as a "pair" when applying the P-DBR. Hence, in this case

$$\alpha_N^{(k)} = \beta_{\lfloor \frac{N}{2} \rfloor + 1}^{(k)}$$

where $\lfloor x \rfloor$ denotes the integer part of x. In any case, we only require $\lfloor \frac{N}{2} \rfloor$ applications of the SFLC.

Remark. Both schemes decompose the N-buffer system into many simpler two-buffer systems, each of which utilizes the original two-buffer fuzzy DBR. Therefore, in situations similar to scenario II, where traffic sources have rapidly varying transmission rates and all buffers have equal storage capacity, we infer from the results in Section 3 that the binary tree F-DBR will outperform the pairing F-DBR, which will outperform the N-buffer version of the P-DBR, which, itself, will greatly outperform the static scheme. This is assuming we use the same value of L in each scheme.

5 Conclusions

In this paper, we developed a fuzzy DBR scheme which was robust and non-parametric, relying only on on-line measurements. Two principal observations can be made. First, employing a DBR mechanism greatly enhances the performance of a buffered network (over the static scheme) with respect to key performance measures. Secondly, our fuzzy DBR significantly out-performs all alternatives (including the intuitive proportional DBR) when the individual traffic sources have intensities which change drastically over short time intervals.

Acknowledgment

This work was partially funded by a grant from MITACS, a Network of Centres of Excellence, and by Nortel Networks. We would like to thank Don Dawson (Carleton University, Canada), Tadeusz Drwiega (Nortel Networks) and James Yan (Nortel Networks) for their many helpful insights throughout this project.

References

[1] P. Chemouil, J. Khalfet, M. Lebourges, "A fuzzy control approach for adaptive traffic routing," *IEEE Communications Magazine*, vol. 33, no. 7, pp. 70–76, 1995.

[2] N. G. Duffield, T. V. Lakshman, D. Stiliadis, "On adaptive bandwidth sharing with rate guarantees," *Proceedings of IEEE INFOCOM' 98*, San Francisco; April 1998, pp. 1122–1130.

[3] M. W. Garrett, W. Willinger, "Analysis, modeling and generation of self-similar VBR video traffic," *Proceedings of ACM SIGCOMM'94*, London, UK; August 1994, pp. 269–280.

[4] N. Giroux, S. Ganti, *Quality of Service in ATM Networks*. Prentice-Hall 1998.

[5] J. M. Pitts, J. A. Schormans, *Introduction to ATM Design and Performance*. Wiley 1996.

[6] B. Qiu, "The application of fuzzy prediction for the improvement of QoS performance," *Proceedings of ICC'98*, pp. 1769–1773, 1998.

[7] L. Reznik, *Fuzzy Controllers*. Newness 1997.

[8] G. Trajkovski, B. Cukic, M. Bogatinovski, "A comparison of two buffer occupancy control algorithms in ATM networks," *1999 IEEE Symposium on Application-Specific Systems and Software Engineering & Technology (ASSET 99)*, Dallas, TX; March 1999, pp. 18–25.