Analysis of neural-network-based congestion control algorithms for ATM networks

Christos Douligeris\textsuperscript{a,}\textsuperscript{*}, Brajesh Kumar Singh\textsuperscript{b,}\textsuperscript{1}

\textsuperscript{a}University of Piraeus, Department of Informatics, Karaoli & Dimitriou 80, 18534, Piraeus, Greece
\textsuperscript{b}Department of Electrical and Computer Engineering, University of Miami, FL, USA

Received 1 March 1998; accepted 1 February 1999

Abstract

Congestion control provides a challenge in the design of Asynchronous Transfer Mode (ATM) networks. Several algorithms have been proposed in the literature for alleviating or reducing congestion. In this paper the Jumping Window (JW), Triggered Jumping Window (TJW) and the Exponentially Weighted Moving Average (EWMA) window algorithms are analyzed, based on a closed-loop predictive feedback mechanism using a neural network. Single- and multiple-source models with real-world and simulated data are used to test the performance of the proposed mechanisms. The consequences of delayed feedback messages is also considered. Results indicate that neural controllers are effective in reducing the cell loss rate, while introducing minimal additional delays. © 1999 Elsevier Science Ltd. All rights reserved.

Keywords: ATM networks; Neural networks; Congestion control; Feedback control

1. Introduction

Within the last decade, the demand for communication and electronic information has increased exponentially. Advances in computer technology and communication technology are driving communication traffic to limits that traditional voice and packet networks cannot accommodate.

The Asynchronous Transfer Mode (ATM) was adopted by the Comite’ Consultatif International Telegraphique et Telephonique (CCITT) as the transport method for Broadband Integrated Services Digital Networks (B-ISDN), a network that will integrate all traffic types and services into a single network. The diagram in Fig. 1 depicts a portion of an ATM network, where different sources of traffic are shown, including voice, video and data services. The users will typically access a local network so that they interact with each other locally. The ATM network backbone is then accessed when a user needs to ‘connect’ to another distant user. This connection is achieved at the User-Network Interface (UNI). Once on the ATM backbone, one ATM network communicates to another via the Network-Network Interface (NNI), until finally reaching another local network, where a UNI is used again.

Given the co-existence of several different types of connection with differing and often conflicting requirements and characteristics makes managing network resources a very hard problem that needs novel approaches (Hui, 1988). In traditional networks, once a channel was allocated, it was busy until released, whether it carried any traffic or not. In this integrated environment, statistical multiplexing is used, which allows sharing of the channels and better utilization of the available bandwidth.

ATM is able to support variable bandwidth for traffic by statistically multiplexing heterogeneous data onto the same transmission channel. This may lead to congestion, even if all the sources comply with their negotiated traffic parameters, in cases where all types
of traffic transmit at their peak levels. The effectiveness of traditional reactive feedback and windowing control mechanisms is poor because of the very high data rates and high bandwidth supported by the newer micro-chip technology and fiber-optic transmission facilities. Too much data would be lost by the time a source receives the message to reduce its output. Similarly, very large buffers would be needed in case data needed to be retransmitted.

To alleviate these congestion incidents, several researchers have tried to predict incoming traffic characteristics and control the rules of transmitted information before such congestion occurs. Neural networks have been used as such predictors in (Liu and Douligeris, 1995; Liu and Douligeris, 1997; Chen, 1994) with a leaky-bucket algorithm as the main controller. In this paper, a neural network is used as a preventive controller predictor that uses the current traffic patterns to predict the onset of congestion; the controller uses three congestion-control algorithms recommended in the literature, namely the Jumping Window (JW), the Triggered Jumping Window (TJW) and the Exponentially Weighted Moving Average Window (EWMA). These three different congestion control algorithms are analyzed with several scenarios using real-world and artificially generated data. For each scenario, the cell loss rate (CLR) and the delay with and without the neural network are determined. The controller takes into account the propagation delay that would be needed for the source to receive the message. Neural networks prove to provide effectiveness in terms of the cell loss rate and delays achieved, which greatly surpass the performance of the plain schemes. In addition, they offer robustness in cases of lost or delayed packets, and of training with a different dataset than the one used in operation.

In Section 2, the ATM protocol is introduced along with a detailed description of ATM traffic control characteristics. In Section 3, the basics of neural networks are introduced. The neural-network model used in this paper, the generalized delta rule, or back-propagation is presented. The methods used in this paper are also presented in this section. This includes the simulation model used for the three congestion controllers, as well as the feedback controller model used with the neural network. The characteristics of the various traffic data used are shown. A detailed explanation of the buffering and timing methods is also presented here. Finally, a training and testing example of a neural net is provided for completeness. In Section 4, the results of a large number of experiments with the proposed schemes are presented. Section 5 presents the conclusions, as well as suggestions for future research topics.

2. ATM details

ATM is asynchronous, and therefore provides only end-to-end flow control. Node-to-node flow control would be much more complex and incur greater delay, given the higher data rates that ATM operates. Before entering the network, users agree on a set of traffic parameters and the network promises a certain quality of service (QOS) in terms of achieving of throughput, delays etc. ATM guarantees the QOS and the delivery of cells in the order transmitted. A new connection must be first screened to determine if the network has enough resources to support it. This is referred to as Call Admission Control (CAC). Once admitted, a connection or source's traffic is monitored using Congestion Control algorithms or Usage Parameter Control (UPC) functions to check for compliance with
the negotiated parameters, and to make sure that congestion does not occur.

Typical characteristics of traffic that can be used for CAC are average rate, peak rate, burstiness and average burst length. Call admission control algorithms must use information to determine if a call can be accepted by the network or not, for all the nodes involved in the connection. These algorithms will need to take into account the statistical multiplexing characteristics of a variable-bit-rate multiplexed stream. Typically, a connection is allocated bandwidth between its average and its peak rates. This balance is difficult to determine, even when all the traffic parameters are given.

Neural networks have been proposed in the ATM literature in different stages of the CAC operations. A neural-network approach has been suggested in Neves et al. (1993), where the neural network is trained using previous traffic patterns in previous traffic situations to predict the quality of operations caused by each new call. In Hiramatsu (1990) a neural network is used to model the relationship between the offered traffic and service quality. Saito (1991) presents a dynamic CAC algorithm where the distribution of the number of cells arriving in a fixed period of time is used to determine the CAC result.

As described above, congestion may occur even if all the sources comply with their negotiated parameters (Kwon et al., 1993). To prevent this from happening, the network must monitor each source at the network entry point by utilizing Usage Parameter Control (UPC) functions. The ATM literature has seen several algorithms designed to alleviate or make congestion smoother. In Hartanto and Sirisena (1992) a user-network policer is considered for congestion control, where the user can help the network determine the information content and importance of the cells. In Fratta and Musumeci (1992), additional congestion-control strategies are considered for ATM networks. In Ramamurthy and Dighe (1992) a rate-based network access control is implemented for an end-to-end ATM connection.

UPC functions are typically reactive. They are configured to police a certain rate of traffic. If the source exceeds this rate, then the extra cells can be dropped or marked using the CLP (cell loss priority) bit, available in the header of the cell. If the cells are marked, then later, at a point of congestion, the marked cells will be the first ones to be dropped. UPC functions can also send feedback signals to the source (close-loop control) to reduce the transmission rates. One disadvantage with this approach is the previously mentioned long propagation delays and high transmission rates. By the time a feedback signal arrives at the source, too many cells may have been lost.

Received control can, however, be combined with

Fig. 2. (a) The leaky bucket; (b) The Jumping Window mechanism.
some proactive methods which can try to predict the traffic flow. Since multi-layer neural networks are capable of learning any continuous mapping to an arbitrary accuracy (Fausett, 1994), it is possible to train a neural network on a set of sample traffic patterns and then use it to predict the future cell rate one propagation time based on the past and current traffic, as demonstrated by Yu and Chen (1993), Liu and Douligeris (1995) and Durand (1995). Thus, a congestion controller can in effect predict the onset of congestion and warn the source ahead of time to reduce its rates. In this paper, a neural network is trained to learn traffic patterns and predict the cell loss at a future time frame. This prediction takes into account the propagation delay back to the source.

In this paper, NN controllers are integrated with traditional congestion controllers as described by Rathgeb (1991). Single- and multiple-source models with various traffic types are considered. The Jumping Window (JW), Triggered Jumping Window (TJW), Moving Window (MW), Exponentially Weighted Moving Average window (EWMA), and the Leaky Bucket (LB) algorithms were described and investigated by Rathgeb. The most promising algorithms considered by Rathgeb were the LB, EWMA and the JW algorithms. The MW and the TJW were not very highly rated due to their complexity and reduction in speed. A basic description of the algorithms is presented in the next section.

The control mechanisms have in common a cell counter \( N \), also called a pseudo-buffer because it does not actually buffer the cells but counts up to \( N \) cells before starting to drop or mark the cells. The mechanisms are (and should be) transparent or passive to the complying sources of the network. The value of \( N \) is typically related to the average rate that must be policed. After a brief description of the algorithms, more detail is given about the effects of the pseudo-buffer.

2.1. Leaky Bucket (LB) mechanism

Assume that a bucket as in Fig. 2(a) is being filled by a water tap. The bucket has a hole in the bottom, from where the water can leak out. The bucket has an inherent capacity, beyond which water will overflow from the sides. The incoming tap water can be modeled as the cells arriving at the leaky bucket congestion controller. The leak in the bucket is the service rate of the congestion controller, or, in other words, it represents the negotiated rates between the user and the network. If a sudden burst of cells arrives, the bucket just fills up (while continuously leaking). If the bucket gets full and more cells arrive at a rate faster than the leak, then the bucket will overflow, causing the cells to be dropped.

2.2. Jumping Window mechanism (JW)

The Jumping Window mechanism is shown in Fig. 2(b). This mechanism only allows a maximum of \( N \) cells within a fixed time interval or window of time \( T \). The next time window starts immediately after the previous window (hence jumping window), and the count of the number of cells is reset to zero. For example, if \( N = 10 \) cells and the length of the window, \( T = 1 \) s, then, if more than 10 cells arrive within 1 s of the start of the window, the additional cells will be discarded or marked. Again, this will allow short bursts of data through, depending on the value of \( N \) (pseudo-buffer). The Jumping Window is also simple to implement.

2.3. The Triggered Jumping Window (TJW) mechanism

This mechanism is very similar to the Jumping Window. However, a window only starts with the arrival of a cell (i.e., it is triggered by the arrival of the first cell after an idle period). In the case of the Jumping Window, the next window starts immediately after the first, but in this case the next window will only start if triggered by the arrival of a cell.

2.4. The Exponentially Weighted Moving Average (EWMA) mechanism

Like the Jumping Window, this mechanism has the same parameters \( N \) and \( T \). However, in the Jumping Window, \( N \) was fixed for every window. In this case, \( N_i \) (for the \( i \)-th window) is a function of the allowed mean number of cells per interval \( N \) and an exponentially weighted sum of the number of accepted cells \( X_i \) in the preceding interval.

The equations used to calculate the current value of \( N_i \) using the specified value \( N \) are shown in Eqs. 1 and 2.

\[
N_i = \frac{N - \gamma S_{i-1}}{1 - \gamma} \quad \text{where} \quad 0 \leq \gamma \leq 1
\]

\[
S_{i-1} = (1 - \gamma)X_{i-1} + \gamma S_{i-2}
\]

Note that the range of \( \gamma \) is between 0 and 1. If \( \gamma = 0 \) then this algorithm is reduced to the Jumping Window. The parameter \( X_i \) represents the number of cells that were accepted in the \( i \)-th window of the algorithm. The algorithm is simple to implement, and needs a few variables to store the moving average values.

2.5. The Moving Window (MW) mechanism

This mechanism also has the same parameters \( N \) and \( T \) as the Jumping Window. For every cell that
comes in, a common global cell counter is incremented. Each cell also has an associated timer: $T$ s after the arrival of that particular cell, the global cell counter is decremented. In essence, the arrival time of every cell is noted. If a cell arrives and the cell counter is already at $N$, then the cell is discarded or marked. This algorithm gets increasingly more complex with increasing $N$ because a timer must be set for each of up to $N$ cells. It is analogous to a windowing scheme where the window is slowly moving up the time axis.

2.6. Effect of the value of the pseudo-buffer $N$

The response time of an algorithm is the time it takes for that mechanism from the onset of congestion until the violating source has been curbed to the prescribed rate. It can also be interpreted as the time that it takes for a mechanism to detect that there is a violation of the traffic contract. The response times are not considered in this paper, as they clearly discussed in Rathgeb (1991).

Rathgeb used a dimensioning factor $C$, where $C$ was the ratio of the rate the mechanism was trying to control, divided by the mean rate. For all the mechanisms described, $C = 1$ (controlling the exact mean rate) yields a large violation probability or cell loss rate (CLR). However, this probability decreases as the pseudo-buffer $N$ is increased (still holding $C$ at 1). This is because a larger $N$ is able to better control a burst than a smaller $N$. The response time improves with the reduction of the pseudo-buffer ($N$) associated with the algorithms. With neural-network feedback, it is assumed that the response time will be improved if the controller is able to reduce the pseudo-buffer size successfully.

3. Neural-network-based control

‘An artificial neural network is an information-processing system that has certain performance characteristics in common with biological neural networks’ (Fausett, 1994). A neural network is characterized by:

1. Its pattern of connections between neurons (called its architecture);
2. Its method of determining the weights on connections (called its training or learning algorithm);
3. Its activation function.

Although neural networks were designed as far back as 1943 (McCulloch and Pitts), they were limited in application due to the small range of problems they could solve. The discovery of back-propagation techniques for training neural networks was the major reason for the revival of neural networks because back-propagation gave a general method for training a multi-layer net. This method, also known as the generalized delta rule, is simply a descent method to minimize the total squared error of the output computed by the net. Using the generalized delta rule (or back-propagation), the choice of a good activation function can simplify the network computation. An activation function should be ‘continuous, differentiable and monotonically non-decreasing’ (Fausett, 1994). The differentiability is important during the feedback phase when weights are updated to reduce the error in the best direction of the local minimum. A commercial back-propagation neural-network software called PROPAGATOR (Propagator, 1993) was used to train the neural network in this paper. Propagator was chosen because of its simplicity to use and the generation...
of ‘C’ code for the final network. The program is geared primarily for training neural networks using the back-propagation algorithm. It was very flexible in adjusting network parameters including learning rules, learning rates, momentum, number of nodes and layers, and various other parameters.

3.1. Window mechanism using a neural-network predictor

3.1.1. Single-source feedback network controller

The diagram in Fig. 3 shows the network controller model that was used to determine the effectiveness of the neural-network controller. The source of the cell traffic is a data file that contains the number of cells per frame. The cell traffic is passed through a pre-shaper. Initially, because the pre-shaper has no future information, it allows all the traffic to go through (it applies a multiplier or throttle of $T_i = 1$). Based on the incoming traffic rate and the previous two rates (initialized to 1), the neural network generates a predictive throttle value that is passed back to the pre-shaper. The incoming rate is shown in Fig. 3 as $(T_i \cdot \text{Rate}_i + \text{NBuf})/T_i$. Note that the rate is divided by the throttle to get back the original rate of the source. Also note that the additional buffer cells are included in this rate. Additional buffer cells would only have been added to the traffic stream if there was additional space for them. Thus, by adding the buffered cells, the rate is not being increased beyond the average, and the prediction of the neural network will still be correct. The propagation delay is assumed to be such that the traffic for two frames later is predicted. This is a sufficient time ahead for the throttle value to propagate back to the pre-shaper, just when the appropriate sample is arriving.

To emulate the delay caused by propagating back the feedback signal, the throttle was buffered for one frame time in the throttle array. Thus, the correct throttle would be applied to the correct frame. The discarded cells are added to a buffer.

3.1.2. Pre-shaper unit, buffer and delay calculations

The throttle is applied in the pre-shaper unit and if there is a rate reduction, the additional cells are stored in a buffer. The buffer can be emptied during periods of low traffic by stuffing back additional cells into the traffic stream. Because the $N$ and $T$ parameters of the window controller are known to the pre-shaper, the shaper can determine, on the basis of the current rate, if additional cells can be added to the traffic stream as shown in Fig. 3. These additional cells are not used by the neural network as an input rate. However, even if they were, it would not significantly affect the NN pre-

![Fig. 4. Multiple-source feedback network model.](image-url)
diction. This is because buffer cells would only be added to the traffic stream at below average rates and the NN would not indicate any future if the traffic is at the average rates.

In this paper, end-to-end delay for the traffic is given. This is the additional amount of time required to transfer the required number of cells using a neural-network controller. In a sense, by buffering, the cells are being delayed, and even if there is a reduction in CLR, there should be very little additional delay; otherwise the CLR reduction is pointless. By taking advantage of emptying the buffer during low rates, it will be shown in the results that the end-to-end delay is very small.

3.1.3. Multiple-source feedback network controller

The design model used for the multiple source feedback network controller is shown in Fig. 4. The multiple-source controller is very similar to the single-source controller. In this case, each of the sources has a pre-shaper and a buffer unit. A single neural network is used to predict the traffic for all three sources. This is handled by the box labeled ‘Neural Network’. The multi-controller interfaces to the common neural network to determine predictions for future traffic. Within this multi-controller module are the previous two rates for each source. For each source, the current rate and the previous two rates (normalized in the same way as for the single source) are fed into the neural network as inputs 1, 2 and 3, as shown in the diagram. The neural network generates an output for the appropriate source, and the controller forwards this output (or throttle) to the appropriate source’s pre-shaper. The propagation delay of DT is shown for each of the feedback signals. The frame and feedback timings are detailed in the next section.

3.1.4. Frame throttle prediction timings

Time frames of data are used to generate inputs for the neural network. Once a frame has passed, the neural network has enough information from the congestion controller (as well as from the previous two cycles) to make a prediction on how to regulate the rate of the frame after the next frame. This prediction can take a certain amount of time, $T_{prediction}$. Then this prediction must propagate back to the source before the frame it is predicting emerges (the frame after the next frame). In the case of the multiple controller, a single neural network is being used to predict all three sources. Thus the prediction time for each source will be the time per prediction, plus the time it had to wait for other sources to be predicted. For example, the 3rd source will have to wait for the 1st and 2nd to be predicted.

Once the neural network has made a prediction (just after the current frame has passed), the feedback signal is sent to the source. At this point, the next frame is being monitored by the controller (call it the 2nd frame). In other words, the 2nd frame has already arrived at the controller. For the feedback signal to make it back to predict the frame after the next frame, call it the 3rd frame, the 2nd frame must not have been transmitted more than half-way from the source; otherwise, the feedback signal will not make it back in time.

Thus the propagation delay must be less than half the time of a frame, to allow time for the prediction to occur. The time for prediction, $T_{prediction}$ assumes waiting for any previous sources to be predicted, and can be represented:

$$T_{propagation} < 1/2(frame\ time) - T_{prediction}.$$

4. Performance evaluation

For every experiment, except from the experiments for the cross-testing of the neural networks, a separate neural network was trained by a small sample of the representative data and then used to test a large sample of the data. Overall, 8 separate neural networks were used.

4.1. Generating a training data set for the neural network

When the status of the window is sampled after time $T$, the current rate and the throttle that should have been used to reduce the cell loss are written to a train-
Throttle is computed as follows:

\[ \text{Throttle} = 1 - \left( \frac{\text{cells dropped}_{\text{interval} t}}{\text{total cells}_{\text{interval} t}} \right) \]

Note that if the throttle is 0 all the cells are dropped, and if the throttle is 1, no cells are dropped. The training patterns are used as inputs to the neural network during training, as shown in Fig. 5. The normalization of the input rates is based on the maximum (or peak) rate of the input file.

4.2. Normalization of input data

Normalization is very important because of the activation function used in a neural network. In this paper, the sigmoid function was used as the activation function.

Note that because the inputs to the neural network are normalized on the basis of peak rates, during operation of the neural network, the peak rate will have to be known a priori. This peak rate can be obtained during the CAC procedure. It can also be dynamically monitored while the neural network is running, and updated in case it was not correctly specified during the CAC phase. In cases of cross-testing the network, the burstiness of the data must also be known. Recall that burstiness is the peak rate divided by the mean rate.

4.3. Characteristics of data used for simulation

Several different sources of data were used to verify the congestion-control algorithms in the simulations. A summary of these is presented in Table 1. For all the simulations, \( N \) and \( T \) were chosen such that the congestion controller would police the average rate. This is the case with the over-dimensioning factor \( C \), set to 1. The average rate is assumed to be known a priori. This peak rate can be obtained during the CAC procedure. It can also be dynamically monitored while the neural network is running, and updated in case it was not correctly specified during the CAC phase.

4.4. Jumping Window

A summary of the JW algorithm with the three types of data, MPEG, AR and JPEG is presented in Table 2. The comparison is made at a CLR of 10^{-4}. The MPEG data yields a 19% buffer reduction and a factor of 12 in reduction of cell loss. This suggests a better response time for the neural-network-based algorithm, as well as a better quality of service. The end-to-end delay is very small, about 10^{-4} percent. The CLR and delay graphs for the JW with MPEG data and an MPEG-trained NN experiment are shown in Fig. 6(a, b).

The AR data shows a much more dramatic performance improvement, about 43% for the pseudo-buffer reduction and a 100 times improvement in cell loss. The additional delay incurred is also very small. This is because the neural net is able to predict this data much more easily. The CLR and delay graphs for the JW with AR data and an AR-trained NN experiment are shown in Fig. 6(c, d).

The improvements in the JPEG are not as marked, but do reduce CLR by a factor of 10. Note that the delay for the JPEG data is, albeit small, significantly higher than the other two data types. This is because its rate is more constant, and the pre-shaper buffer does not have many opportunities to empty itself at lower rates. Overall, it may not be a good idea to police single source images for file transfer using a neural network. The CLR and delay graphs for the JW with JPEG data and a JPEG-trained NN experiment are shown in Fig. 6(e, f).

4.5. Exponentially Weighted Moving Average Window

A summary for the EWMA controller is presented in Table 2 for the CLR of 10^{-4}. Compared with the JW algorithm, it is immediately apparent that the EWMA algorithm presents similar CLR values, with smaller pseudo-buffer values. The performance gains using the neural-network controller are very similar to

---

Table 1: Data characteristics

<table>
<thead>
<tr>
<th>Data source and type</th>
<th>Mean (bits/fr)</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Burstiness</th>
<th>Mean (s/cell)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real JPEG data converted to 24 frames/s</td>
<td>113,649, 2.73 Mb/s</td>
<td>33,920, 0.79 Mb/s</td>
<td>312,912, 7.5 Mb/s</td>
<td>2.75</td>
<td>1.55E-4</td>
</tr>
<tr>
<td>Artificial AR Markov Video Stream, 30 frames/s</td>
<td>119,767, 3.59 Mb/s</td>
<td>400, 0.012 Mb/s</td>
<td>328,579, 9.86 Mb/s</td>
<td>2.74</td>
<td>1.18E-4</td>
</tr>
<tr>
<td>Real MPEG data from 110 + 20 K, 24 frames/s</td>
<td>17,215, 0.41 Mb/s</td>
<td>1009, 0.024 Mb/s</td>
<td>114,744, 2.75 Mb/s</td>
<td>6.67</td>
<td>1.026E-3</td>
</tr>
</tbody>
</table>
Table 2
Performance summary for $N$ at CLR of $10^{-4}$

<table>
<thead>
<tr>
<th></th>
<th>JW buffer without NN</th>
<th>JW buffer with NN</th>
<th>% buffer gain with NN</th>
<th>Factor of CLR improvement with NN at $10^{-4}$</th>
<th>% total delay incurred by pre-shaper buffer of NN</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPEG</td>
<td>&gt; 220</td>
<td>170</td>
<td>&gt; 22%</td>
<td>12</td>
<td>$10^{-4}$</td>
</tr>
<tr>
<td>AR Video</td>
<td>&gt; 120</td>
<td>88</td>
<td>&gt; 26%</td>
<td>100</td>
<td>$10^{-4}$</td>
</tr>
<tr>
<td>JPEG</td>
<td>63</td>
<td>59</td>
<td>6%</td>
<td>10</td>
<td>$10^{-2}$</td>
</tr>
<tr>
<td>EWMA buffer without NN</td>
<td>&gt; 110</td>
<td>95</td>
<td>&gt; 14%</td>
<td>15</td>
<td>$10^{-3}$</td>
</tr>
<tr>
<td>AR Video</td>
<td>&gt; 105</td>
<td>75</td>
<td>&gt; 29%</td>
<td>&gt; 100</td>
<td>$10^{-4}$</td>
</tr>
<tr>
<td>JPEG</td>
<td>63</td>
<td>53</td>
<td>16%</td>
<td>70</td>
<td>$4 \times 10^{-3}$</td>
</tr>
<tr>
<td>TJW buffer without NN</td>
<td>&gt; 220</td>
<td>190</td>
<td>14%</td>
<td>11</td>
<td>$10^{-4}$</td>
</tr>
<tr>
<td>Single source without NN</td>
<td>&gt; 120</td>
<td>94</td>
<td>22%</td>
<td>60–80</td>
<td>$10^{-4}$</td>
</tr>
<tr>
<td>MPEG</td>
<td>&gt; 220</td>
<td>168</td>
<td>170</td>
<td>123</td>
<td></td>
</tr>
<tr>
<td>AR</td>
<td>&gt; 120</td>
<td>91</td>
<td>88</td>
<td>75</td>
<td></td>
</tr>
</tbody>
</table>

AR Model          MPEG model

<table>
<thead>
<tr>
<th>Feedback Signal Loss</th>
<th>88</th>
<th>109</th>
<th>170</th>
<th>196</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random 10% signal loss</td>
<td>109</td>
<td>196</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Random 30% signal loss</td>
<td>117</td>
<td>210</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Random 50% signal loss</td>
<td>122</td>
<td>230</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Fig. 6. (a) CLR for JW with MPEG data and NN trained by MPEG; (b) Delay for JW with MPEG data and NN trained by MPEG; (c) CLR for JW with AR video stream and NN trained with AR data; (d) Delay for JW with AR video stream and NN trained with AR data; (e) CLR for JW with JPEG data and JPEG-trained NN; (f) Delay for JW with JPEG data and JPEG-trained NN.
Fig. 6 (continued)
Fig. 7. (a) CLR for EWMA with MPEG data and MPEG-trained NN; (b) Delay for EWMA with MPEG data and MPEG-trained NN; (c) CLR for EWMA with AR data and AR-trained NN; (d) Delay for EWMA with AR data and AR-trained NN; (e) CLR for EWMA with JPEG data and JPEG-trained NN; (f) Delay for EWMA with JPEG data and JPEG-trained NN.
Fig. 7 (continued)
Fig. 8. (a) CLR for TJW with MPEG data and MPEG-trained NN; (b) Delay for TJW with MPEG data and MPEG-trained NN; (c) CLR for TJW with AR data and AR-trained NN; (d) Delay for TJW with AR data and AR-trained NN.
Fig. 9. (a) CLR for JW NN trained with MPEG data controlling an AR video stream; (b) CLR for JW NN trained with MPEG data controlling an AR video stream; (c) CLR for JW NN trained with AR controlling an MPEG video stream; (d) CLR for JW NN trained with AR data controlling an MPEG video stream.
the performance gains obtained with the neural net on the JW. For the MPEG data, there is a 14% reduction in buffer size, and a 15 times improvement in CLR. The CLR and delay graphs for the EWMA Window with MPEG data and a MPEG-trained NN experiment are shown in Fig. 7(a, b), respectively. With the AR model, the improvement is more pronounced, with a 29% reduction in pseudo-buffer size and over 100 times improvement in CLR (see Fig. 7(c, d)). The JPEG data shows a much better improvement over the JW JPEG model with a 16% reduction in pseudo-buffer size and a 70 times improvement in CLR. Note that for the JPEG, this improvement does not last for increasing values of $N$ because the algorithm ends up policing the peak rate. The CLR and delay graphs for the EWMA Window with JPEG data and a JPEG-trained NN experiment are shown in Fig. 7(e, f), respectively.

4.6. Triggered Jumping Window

A summary of the TJW controller is given in Table 2. Note that no JPEG model was analyzed for the TJW. The pseudo-buffer values are slightly higher than the JW model to yield the same CLR. The CLR and delay graphs for the TJW with MPEG data and an MPEG-trained NN experiment are shown in Fig. 8(a, b), respectively. The MPEG model yields a 14% pseudo-buffer reduction and an 11 times improvement in CLR. The CLR and delay graphs for the TJW with AR data and an AR-trained NN experiment are shown in Fig. 8(c, d), respectively. The AR model yields a 22% pseudo-buffer reduction and in the region of 60–80 times improvement in CLR. Recall that the response time of the TJW mechanism is poorer than that of the JW mechanism.

4.7. Cross-testing of the neural network using the JW

The cross-testing was only done with the JW algorithm. Cross-testing was performed against the MPEG and AR data sets. The CLR and delay graphs for a JW NN trained with MPEG data controlling an AR video stream are shown in Fig. 9(a, b). The CLR graph shows that after $N = 64$, the performance of the native controller was better than the MPEG controller. The CLR and delay graphs for a JW NN trained with AR data controlling an MPEG video stream are shown in Fig. 9(c, d). The AR controller is not able to control the MPEG data as well as the native MPEG controller.

4.8. Congestion control with multiple sources using the JW

Three sources were multiplexed and a window con-

![Fig. 10.](a) CLR for JW with 3 AR video stream sources; (b) Delay for JW with 3 AR video sources; (c) CLR for JW with 3 MPEG streams; (d) Delay for JW with 3 MPEG streams.)
The CLR and delay graphs for the JW controller with three AR model video streams are shown in Fig. 10(a, b). The same graphs for the JW controller with three MPEG data streams are shown in Fig. 10(c, d). A summary of the results is presented in Table 2. It is clear that there is a reduction of the pseudo-buffer with multiple sources, whether or not the neural network is used. It should be noted that the greater percentage of reduction is when the neural network controller is not used. Table 2 shows that the single source without the NN had a pseudo-buffer value of 220, but with multiple sources this was reduced to 168. Similarly, the AR case without the NN was reduced from 120 to 91. A similar result is apparent when the NN was used.

4.9. Unexpected delay analysis

In the model proposed in this paper, there is no acknowledgement from the source as to whether the feedback signal from the neural network was received. Even if this acknowledgement were sent (possibly piggybacked on a cell), it would do little good on arriving at the controller, because it would be too late for the controller to take any action, although several negative acknowledgements could indicate a problem in the network.

In this set of simulations, the effect of a feedback signal not making it back to the source transmitter was analyzed. It was assumed that if the source did not receive a feedback signal, then the exact unthrottled rate of cells would be transmitted. This is equivalent to a throttle value of unity being transmitted back to the source, and this was what was done in the simulation to achieve the same effect.

In the results following, 10, 30 and 50% of the feedback signals not making it back to the source were considered for the JW model, with both MPEG and AR data.

It is possible that a delayed signal may make it to the source after half the frame has been transmitted. In this case, it would help to reduce the CLR for the remaining half of the frame. The cases shown therefore give the worst-case scenarios, and real scenarios may give better results.

Fig. 11(a, b and c) shows the CLR for MPEG data with 10, 30 and 50% feedback signals lost. As shown in Table 2, as a larger percentage of the feedback signal is lost, a larger pseudo-buffer is needed to achieve a CLR of $10^{-4}$, for both the MPEG and AR models.

5. Conclusions

In this paper, neural-network-enhanced JW, TJW and EWMA algorithms were investigated for ATM congestion control. The neural-network controller was employed to do pro-active prediction of congestion by sending feedback messages to the source. The propagation delay back to the source was taken into account. Real-world MPEG data, JPEG data and artificially generated Auto-Regressive Markov data were used for the simulations.

Performance was measured by noting the point at which the neural-network controller gave a CLR of $10^{-4}$. Then this value was compared with what the CLR would have been without the use of the NN controller. The AR data was quite easy to predict with the neural network since it represents a well-defined theor-
ethical model. The neural network improved the CLR by an order of 100, with very little additional delay. The real-world MPEG data gave promising results, with more realistic CLR improvements, in the order of ten times the standard controller at a CLR of $10^{-4}$. The JPEG data showed that with a source not fluctuating very frequently, increasing the pseudo-buffer counter very quickly approaches policing the peak rate. Although the JPEG data was bursty, the burst length was very short. In cases with very short and infrequent bursts, a NN controller may not be appropriate, and may cause additional delay.

It was shown that cross-usage of the neural networks is possible. Therefore, a neural network trained with a very wide set of data characteristics may be used in several different situations. The MPEG-trained NN was able to control the AR model very well, but not vice versa.

An example was presented with multiple multiplexed sources, and the results clearly showed better performance than the single sources due to the statistical multiplexing gain. In real use, this is going to be the most likely scenario, and the single-source controllers may be used as a baseline to gauge the effectiveness of adding a controller. Also, the effects of a JPEG source (small and infrequent burst lengths) can be minimized when a controller monitors multiple sources.

The response time of the controlling mechanisms has always been a key issue. The use of the neural-network controller helps to reduce the pseudo-buffer size by up to 25% in several cases. Thus a lower value of the over-dimensioning factor can be used to obtain a tighter control of the traffic.

The delayed feedback signal analysis shows that the NN controller is quite robust, giving a performance with over 10 times reduction in CLR at $10^{-4}$, with 10% of the feedback signals being randomly delayed. This performance improvement was demonstrated by both the MPEG and AR models. The AR model, however, was more sensitive to the lost feedback signals. Although performance declines, with 30 and 50% of the feedback signals being delayed, these situations should arise very rarely (including the 10% delay).

The results provided in this paper complement the results in previous papers (Liu and Douligeris, 1995, 1997; Neves et al., 1993), where the controllers investigated were of the Leaky Bucket type. In addition, this paper has evaluated the effects of cross-training, lost feedback signals and multiple sources on the performance of the algorithms for the first time in the ATM literature. Further work is needed to identify the possibility of hardware implementation of the controller, as well as the use of other types of neural networks.

References


Propagator, 1993. ARD Corporation, 9115 Whiskey Bottom Road, Suite I, Laurel, MD 20723.


