HBP: An Optimization Technique to Shorten the Control Cycle Time of the Neural Network Controller (NNC) that Provides Dynamic Buffer Tuning to Eliminate Overflow at the User level

Wilfred W.K. Lin[1], Allan K.Y. Wong[1] and Tharam S. Dillon[2]

[1]Department of Computing, Hong Kong Polytechnic University, Hunghom, Kowloon, Hong Kong SAR, Emails: cswklin@comp.polyu.edu.hk, csalwong@comp.polyu.edu.hk

[2]Faculty of Information Technology, University of Technology, Sydney Broadway, N.S.W. 2000, Email address: tharam@it.uts.edu.au

Abstract

The NNC (*Neural Network Controller*) automatically tunes the buffer size at the user/server level to eliminate any chance of overflow in the client/server interaction over a TCP logical channel. Together with the buffer tuning operations at the system/router level (e.g. the AQM (*Active Queue Management*) activities) they form a unified solution. The power and stability of the NNC was verified over the Internet, but the result shows that the drawback of the NNC is its long control cycle time. This drawback hinders the deployment of the NNC in the real-time applications. To overcome this we propose the novel HBP (*Hessian Based Pruning*) optimization technique. This technique operates as a renewal process, and within the service life of the *Optimized NNC* (O-NNC) the optimization operation repeats as renewal cycles. The feed-forward neural network configuration of the O-NNC is optimized in every cycle that involves two phases. In its original un-optimized form the NNC runs as a twin system of two modules:" *Chief* + *Learner*". The O-NNC always starts with the un-optimized configuration. In the first phase the weights for the *Learner's* neural network arcs are computed and sorted. Those arcs with weights insignificant to the control convergence speed and precision are marked. The marking is based on "*dynamic sensitivity analysis*" that utilizes the HBP technique. In the second phase the *Chief* optimizes the neural network by excluding/skipping the marked arcs. The aim is to shorten the computation for the control cycle. The "*HBP+NNC*" is the basis of the O-NNC model, which essentially uses virtual pruning because the marked arcs are excluded from the computation but not physically removed. While the *Chief* is carrying out actual dynamic buffer tuning the *Learner* undergoes training. The O-NNC model is verified by running the Java-based prototype on the Aglets mobile agent platform in the Internet environment. The results are positive and indicate that the HBP technique indeed yields a shorter O-NNC control cycle time than the original un-optimized NNC in a consistently manner.

Keywords: O-NNC, HBP, optimization, pruning, Taylor series, buffer overflow control

1. Introduction

Ideally a *Transmission Control Protocol* (TCP) channel could have two levels of buffer overflow control mechanisms to achieve a unified solution: a) throttling and AQM (*Active Queue Management* [Brad98]) at the router/system level, and b) dynamic buffer tuning at the server/user level [PRDC99, PID01, FLC03, GAC02]. Different methods exist for a router to throttle a client process that sends a lot of traffic in a short time because the sudden burst of traffic can easily overflow the router buffer, causing widespread retransmissions and network congestion.

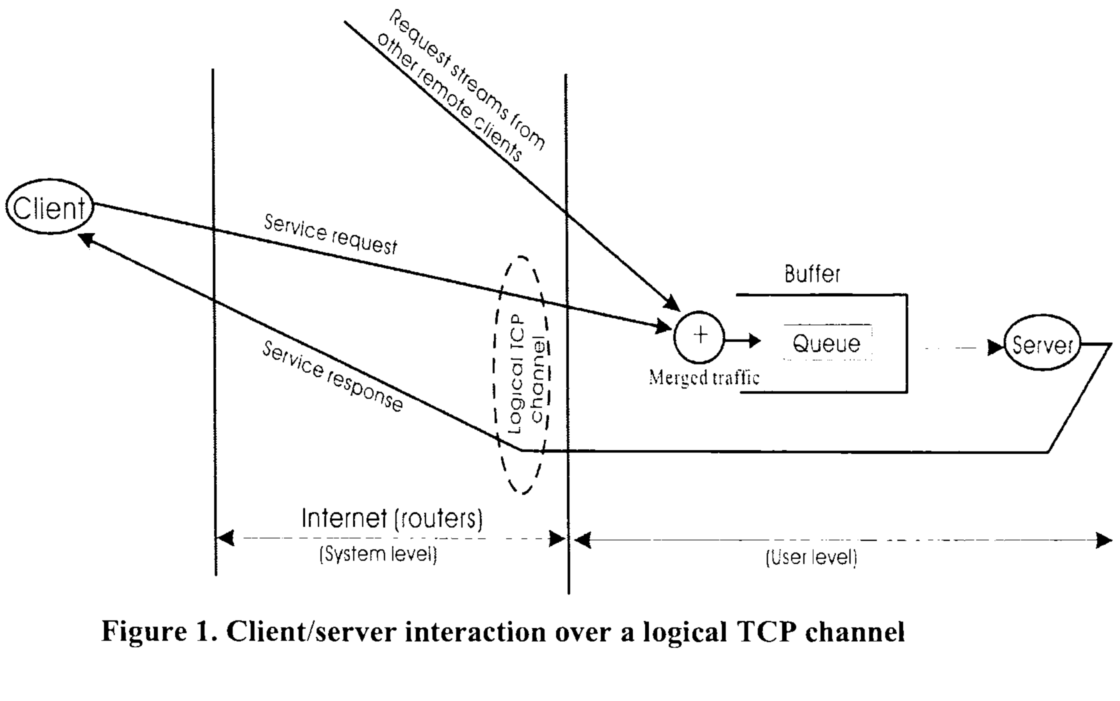

Figure 1. Client/server interaction over a logical TCP channel

Throttling can be achieved by two means: a) by using the user-provided ad hoc algorithms that work with the TCP/IP choke packets [Tan03], and b) with organized solutions designed for TCP end-to-end *active buffer management* at the system level. The IETF (*Internet Engineering Task Force*) proposes to use the AQM approach in the RCF2309 and the RED (*Random Early Discard*) algorithm [Floyd93] is the candidate [Brad98]. The RED operates in two stages: a) firstly, to throttle the sender to cut down transmission voluntarily, and b) if this does not work then new messages are dropped in favor of those already queued. Different experimental results, however, indicate that the RED algorithm is unstable [Ott99] and this led to the appearance of different algorithmic and intelligent AQM methods [Ren02, Fir00]. The RED experience has provided valuable information for how to design a better RED-based algorithm. The problem of the RED is caused by its "*un-configured*" approach in which new packets are randomly dropped. This approach makes RED unstable by being overly sensitive to network parameters [May00, Chris00], and to over the instability the enhanced RED versions are self-configuring and/or configurable. A self-configuring mechanism drops packets selectively, and the FRED (*Fair RED*) [Lin97] is an example that controls its selectivity by the per-flow information. The FRED, however, consumes more memory and processing time than the simpler WRED (*Weighted RED*) and RIO (*RED In and Out*) mechanisms [Bod00]. The WRED, which is for usage in the core router rather than the peripherals, can be configured to drop packets selectively based on the IP precedence. The higher priority traffic (with higher precedence) is then delivered with a higher probability. In the Cisco 7500 series router, the WRED implementation is called the DWRED (*Distributed WRED*) because it is VIP (*Versatile Interface Processor*) based [Cisco]. The other intelligent AQM algorithms that do not base on the RED include the expert *Fuzzy-PI* model proposed in [Ren02]. The extant AQM algorithms work exclusively with *fixed-length buffers* (FLB) and the ultimate action is to drop the incoming messages by the "*drop from front strategy* [Laks97]". The tradeoff of the FLB approach is increase in network congestion due to widespread subsequent retransmissions. Since the AQM-throttled client reacts to reduce its transmission rate in a voluntary manner, the benefit obtained by throttling is difficult to ascertain. However one can be certain that system-level throttling does not prevent buffer overflow at the (server) level because of the character of the merged traffic (indicated by the "+" symbol in Figure 1) [Mit98, GAC02]. Figure 1 abstracts the client/server interaction over a TCP channel, and the server provides service to different clients in the *asymmetric rendezvous* (*one-to-many relationship*). The merged traffic for the different streams of client requests may be LRD (*long-range dependence*), SRD (*short-range depen*dence) or multi-fractal [Pax95, Med00]. If all the clients make their requests simultaneously, the merged traffic easily increases the queue length to overflow the buffer. The subsequent retransmissions due to loss of requests in the overflow prolong the service *roundtrip time* (RTT). Therefore, eliminating the overflow at the user level by dynamic buffer tuning [GAC02] is important for achieving a reasonable and shorter RTT. The elimination also prevents waste of the throttling resources already dished out at the system level.

Since Internet computing is naturally distributed, inter-object collaboration over the Internet involves routing messages that traverse through different physical nodes and links of varying qualities and capacities. For the sheer size of the Internet, routing is frequently plagued by faults and errors of various kinds due to the network dynamics (e.g. transmission errors, partial network hardware failures, and buffer overflow). If the collective error probability ρ encompasses all the possible faults, then the *average number of trials* (ANT) to get a transmission success over a logical channel is $ANT = \sum_{j=1}^{k \to \infty} jP_j$, which can be simplified to $ANT = \sum_{j=1}^{k \to \infty} j[\rho^{j-1}(1-\rho)] \approx 1/(1-\rho)$. Since ρ includes the error probability component ρ_O for buffer overflow, the elimination of ρ_O definitely reduces ANT and shortens the RTT. In fact, all the buffer overflow control methods seek to reduce or eliminate ρ_O . Overflow prevention methods that have previously appeared in the literature can be separated into two classes: FLB (*Fixed Length Buffer*) or VLB (*Variable Length Buffer*). The older approaches (e.g. Blue [Feng99] and the intelligent AQM model in [Awey02]) are all FLB-based techniques. The FLB principle is to ensure the delivery of those messages already queued by dropping the new comers to avoid overflow, with respect to some predetermined criteria. This approach easily creates *random loss* over the Internet and widespread retransmissions [Laks97]. Hence the FLB methods are intrinsically deleterious because they introduce congestion while preventing overflow. The VLB methods, which are more recent, eliminate buffer overflow with much less or no random loss at all. They usually work with dynamic buffer tuning, which ensures the buffer length always cover the queue length. The first VLB algorithm includes the *"P+D"*(*Proportional + Derivative*) controller, designed for user-level Internet applications [PRDC99]. The *"P+D"* controller and the system-level AQM mechanism(s) together form a unified solution that minimizes channel buffer overflow. The "P+D" controller uses the QOB (*queue length over buffer length*) ratio for P (*proportional*) control and the current rate of change of the queue length, namely, dQ/dt for the anticipative D (*derivative*) control. The experience with actual "$P + D$" deployments, however, indicates that it cannot eliminate overflow completely, due to insufficient anticipation. This led to the proposal and development of the more powerful *algorithmic PID* (A-PID) controller, which is an enhancement of the *"P +D"* model by re-enforcing it with *integral* (I) control [PID01]. It is algorithmic because the control parameters do not change at run-time. The integral control element feeds the past performance back to moderate and tune the current control process. The feedback loop enables the A-PID to predict the advent of buffer overflow more accurately and to take proactive corrective actions. The inclusion of the *Convergence Algorithm* (CA) [CA01] in the integral control element gives speed and precision to the latter. The CA is an IEPM (*Internet End-to-End Performance Measurement* [IEPM99]) technique. In the A-PID controller the CA is implemented at the *micro level* because it exists as an independent logical object to be invoked for service by message passing. In its *micro* Java form the CA is known as the M^3RT tool [M^3RT02, M^2RT01]. Although the A-PID experience shows that it always eliminates buffer overflow at the user level, it has two distinctive shortcomings: a) it locks unused buffer space after every corrective action and thus has negative impact on the system throughput, and b) the queue length can get dangerously close to the buffer length and this threatens overflow under serious perturbations. The desire to eliminate these shortcomings prompted the development of the intelligent A-PID versions that use soft computing techniques to tune the A-PID control process. These versions include: a) the GAC (*Genetic Algorithm Controller* [GAC02]), and b) the FLC (*Fuzzy Logic Controller* [FLC03]). The NNC (*Neural Network Controller* [NNC03]) is a spin-off from the GAC and FLC experience. Both the GAC and the FLC use soft computing techniques to support the objective function: $\{0,\Delta\}^2$, which maintains the given safety margin Δ. The maintenance augments the anticipative power of the A-PID component, which by itself has no concept of a safety margin. The NNC also works with the $\{0,\Delta\}^2$ objective function but unlike the FLC and GAC it involves no A-PID model.

2. Intelligent Buffer Controllers and Motivation for HBP

The NNC, which tunes the buffer size in the adaptive manner at the user/server level, has been stable and effective [NNC03]. The problem is its long control cycle time and therefore we propose the *Hessian Based Pruning* (HBP) technique to shorten it. So far, the FLC, GAC and NNC are the only intelligent "*dynamic buffer tuners*" that can be identified from literature. They all eliminate overflow at the user-level by maintaining the given safety margin Δ about the reference, which is represented by "0" in the $\{0,\Delta\}^2$ objective function. This reference can take many forms, and for the FLC, GAC and NNC it is the QOB_R, namely, the chosen "*queue length over buffer length* " ratio. For example, one can have the combination of $\Delta = 0.2$ (i.e. 20%) and $QOB_R = 0.8$ (i.e. 80%).

In the GAC the control parameters are tuned in a timely fashion to eliminate overflow. In this way it provides a more accurate dynamic buffer tuning mechanism than the A-PID. Yet, buffer overflow still occurs under the GAC control for serious perturbations. Our analysis shows that this is caused by the very nature of the genetic algorithms (GA) not to ensure the global-optimal solution in the solution *hyperplane* [Mit99]. Any successful finding of the global-optimal solution hinges upon whether it is present in the subset of solutions provided for the GA operation. The FLC, which is basically "*A-PID + Fuzzy Logic*", has smoother and more precise control than the A-PID. Our experience with the FLC shows that it consistently eliminates overflow completely. Although the FLC is structurally more complex compared to the A-PID, they have comparable execution times. The FLC control, however, can oscillate at the system steady state and yield poor system performance. The desire to eliminate these oscillations motivates the proposal of the NNC. The *control cycle times* (CCT) of the FLC, GAC, and NNC have been measured and compared, with the help of the *Intel's VTune Performance Analyzer* [VTune]. This tool records the CCT in terms of the number of neutral clock pulses/cycles. A measurement can be converted into the corresponding *physical control* *cycle time* (P-CCT) for any platform by: $P-CCT = CCT * \frac{1}{Hz}$, where *Hz* is the platform's operating speed in mega hertz (MHz). The VTune shows that the Java -based, un-optimized NNC prototype always has the longest CCT, compared to the A-PID, FLC, and GAC prototypes (also Java-based), as presented in Table 1.

VLB tuners (Java-based prototypes)
 	
P = proportional control
D = derivative control
I = integral control
	Technique involved	Number of Java lines	Average number of clock cycles to converge to the optimal reference after a perturbation $QOB_R = 0.8$ or $\Delta = 0.2$	Average number of clock cycles per control cycle (CC)

$P+D$ (*frequent overflow*)	Algorithmic and ***no*** IEPM support	48	6450 ($\approx$74 CC)	87
A-PID (***no*** *overflow*)	Algorithmic & **IEPM** $\Rightarrow$ M^3RT	82	5000 ($\approx$22.7 CC)	220
GAC (*occasional but rare overflow*; A-PID& $\{0,\Delta\}^2$)	Genetic Algorithm (GA)	111	4995 ($\approx$10.5 CC)	475
FLC (***no*** *overflow*; A-PID& $\{0,\Delta\}^2$)	Fuzzy logic	116	5220 ($\approx$20.5 CC)	255
NNC by backpropagation ([*Input-Hidden-Output*] neurons: [*10-20-1*]; $\{0,\Delta\}^2$ and ***no*** *overflow*)	Neural Network & **IEPM** $\Rightarrow$ M^3RT	240	10800 (1 CC)	10800

Table 1. The execution times of four intelligent VLB controllers/tuners by VTune [Lin02]

The development of the NNC model had gained from the previous positive experience of using neural networks to support FLB-based AQM algorithms (e.g. [Aweya98]). The experience with the Java-based NNC prototype confirms that the backpropagation approach can indeed get rid of oscillations in the control process effectively. The NNC's feed-forward neural network configuration is shown in Figure 4. Repeated analyses and experiments suggest that there is no performance advantage of having a more complex NNC configuration than Figure 4 (e.g. more neurons for the hidden layer). The long CCT for the NNC prototype nevertheless indicates that the NNC structural complexity should be optimized for the sake of better response timeliness. It is otherwise difficult to deploy the NNC in real-life time-critical applications. For this reason the HBP optimization technique is proposed. The principle is to prune the insignificant neural network connections or arcs in the NNC in a virtual, dynamic fashion [Oh98, Moody96, Goh94].

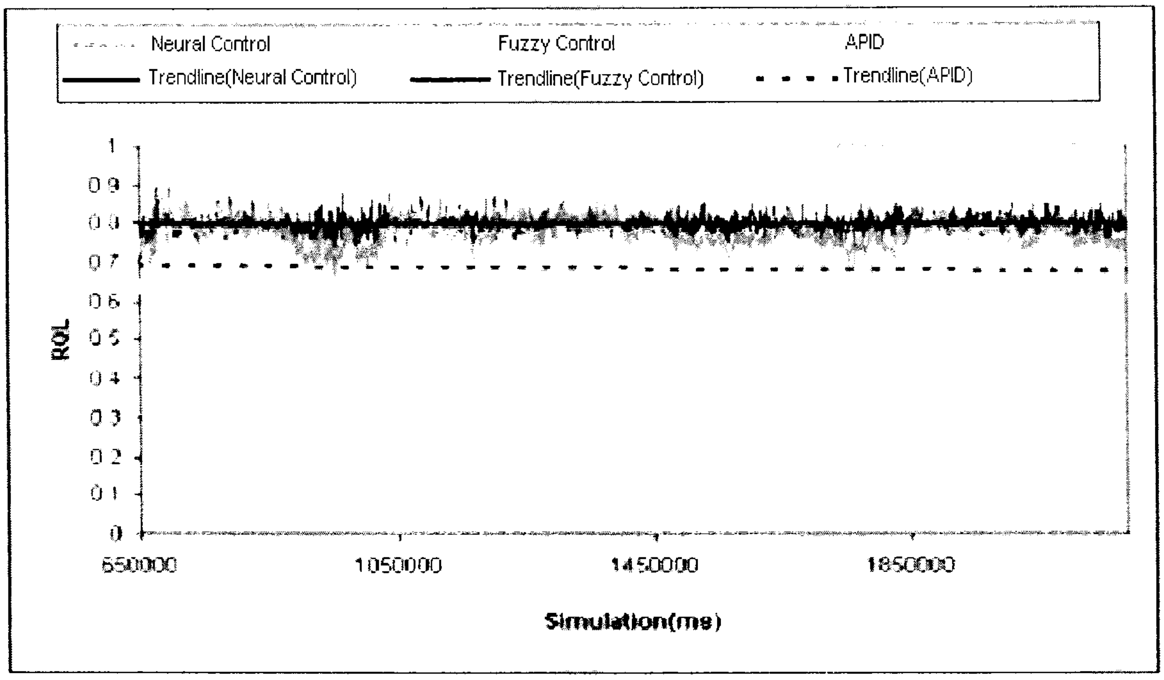

Figure 2. A comparison case of NNC, A-PID and FLC convergence with optimal reference $QOB_R = 0.8$

Table 1 shows that the structurally simplest "$P+D$" tuner (only 48 Java lines) needs the most number of control cycles (i.e. 74 CC) to converge to the optimal QOB_R reference. The CCT time, however, is the shortest (87 clock pulses) among all the listed tuners. The FLC execution time (i.e. 255 clock cycles) is comparable to the much simpler A-PID (i.e. 220 clock cycles) because the presence of "*don't care*" states in the FLC fuzzy control [Lin02]. These states are inert and require no computation at all. Although the NNC always provides the smoothest and the most precise control (Figure 2), it has the longest convergence time (i.e. 10800 clock pulses) towards QOB_R=0.8. This doubles that required by A-PID and FLC. Although physically 10800 clock pulses is not a very long time, its impact on control timeliness can be significant. In light of the Intel-Pentium III 9333 MHz platform the NNC convergence time is $P-CCT = [10800/(933*10^6)] \approx 11.6\ \mu\text{sec}$.

Even in this short period deleterious results are possible if the perturbations are shorter than 11.6μsec. This makes it worthwhile to cut down the CCT of the NNC, and this is precisely what the O-NNC is aiming at.

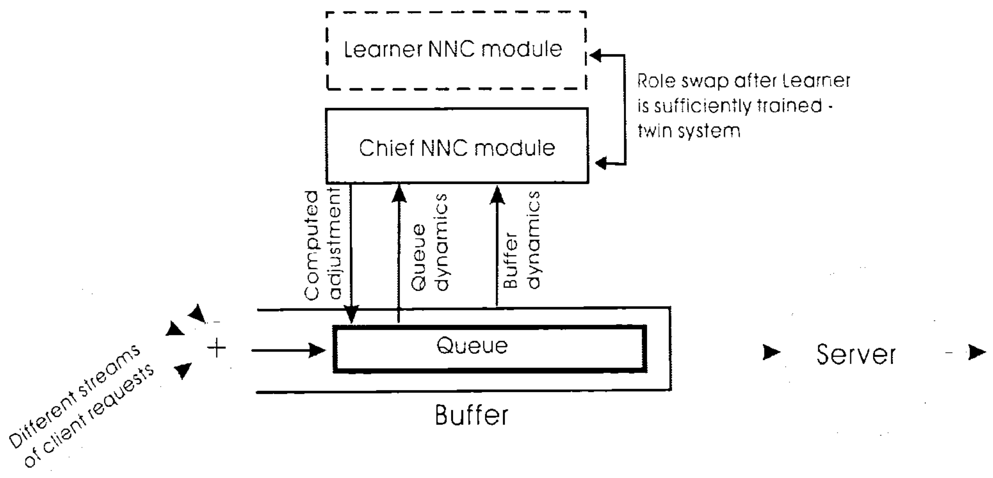

Figure 3a. The O-NNC – a twin system of two NNC clones

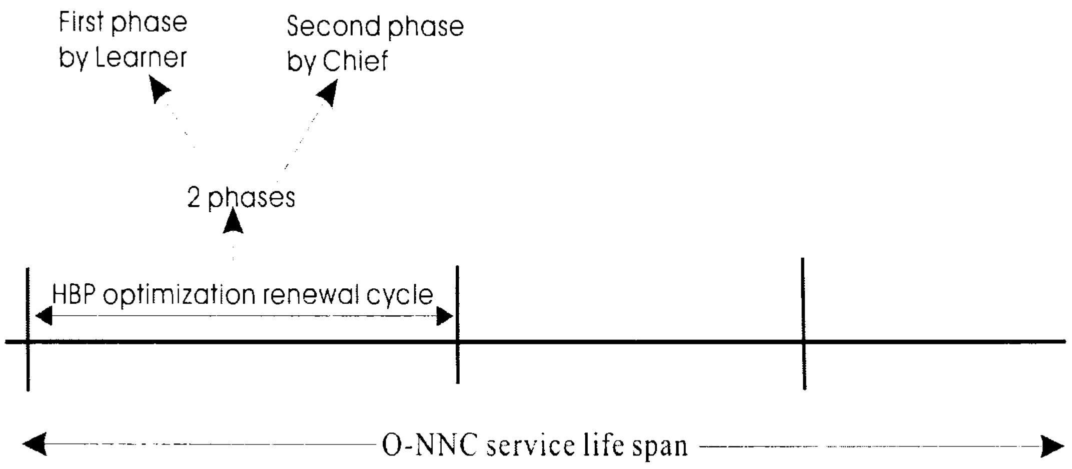

Figure 3b. The HBP is as a renewal process

3. The Optimized NNC

When the O-NNC controller is not running it is structurally a system of two NNC clones: *Chief* and *Learner* that cooperate asynchronously (Figure 3a). While the *Chief* is performing actual dynamic buffer tuning the *Learner* undergoes training/relearning. After finishing training the *Learner* performs the first phase of the HBP optimization process and thereafter the *Learner* swaps position with the current *Chief.* In the first phase the *Learner* computes the weights for all the arcs in the neural network and marks those that have insignificant impact on the control speed and precision. In the second phase of the optimization process the *Chief* excludes (virtually prunes) the marked arcs from the actual computation so that the control cycle time is shortened. The O-NNC is a dynamic model and carries out virtual pruning on the fly. The HBP technique in action is a renewal process, and therein every optimization renewal cycle has two phases. These phases are repeated cyclically throughout the service life span of the O-NNC (Figure 3b). After the *Learner* has trained sufficiently it assumes the role of the *Chief.* The previous NNC development experience, however, shows that any increase in neural network complexity (e.g. more neurons in the hidden layer) amplifies the training time disproportionately [Lin02]. The quest for a simpler configuration led to the choice of the backpropagation model for the NNC. Further inquiry revealed that there is no convergence and performance advantage to have a more complex configuration for the NNC than Figure 4.

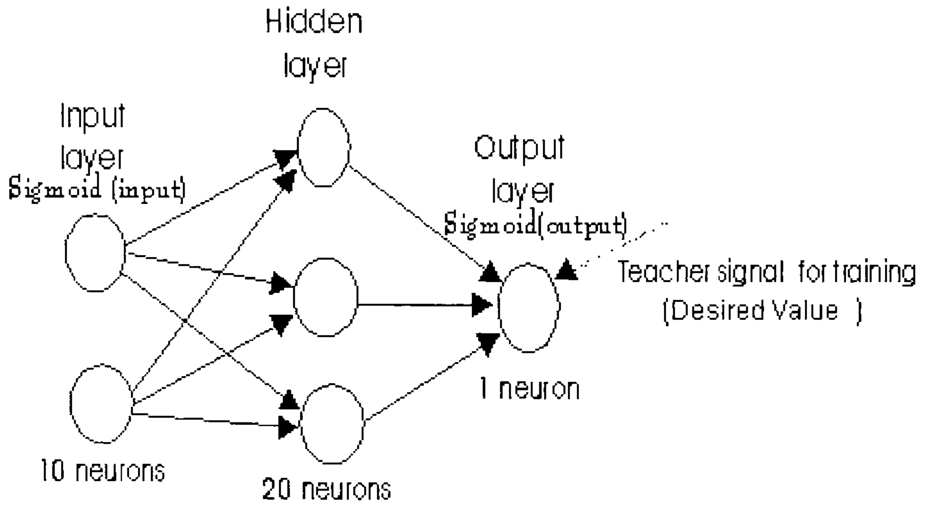

Figure 4. The NNC backpropagation configuration

The skeletal O-NNC neural network configuration is the same as the original NNC: 10 input neurons, 20 hidden neurons, and 1 output neuron. The NNC, which works by backpropagation, is trained with Δ as the teacher signal and the *Sigmoid* function:

$f(x)=1.0/(1.0+e^{-x})$. The activation energy (value) for the neurons in the hidden and output layers are computed by the following:

a) Sigmoid ($\sum$ InputActivation * weight (input-hidden))

b) Sigmoid ($\sum$ OutputActivation * weight (hidden-output))

The input neurons in the NNC are fed by the ten entries in the vector Q_{vector} with the following composition:

a) *Ten queue length samples*: They are sampled at equal time distances within the chosen renewal window of size W, divided equally into ten portions. The data sample for each portion is denoted by Q_{tX} ; $X=1,2,...,10$.

c) The *10th input replacement*: If the NNC is supported by the *micro* M^3RT IEPM technique to predict the trend of change in the queue length, then the 10th entry in the Q_{vector}, namely, Q_{t10_W} is replaced by the $Q_{CA-estimate}$ value. This value enables the NNC to maintain Δ more accurately by using the past control performance as the feedback to moderate the current control cycle for integral effect. The NNC model is summarized in the equations: (3.1) and (3.2). The new buffer length: $L(W+1)$ to maintain the safety margin Δ in the next or $(W+1)^{th}$ control cycle is a function (i.e. $f_{NNC}(.)$) of Q_{vector}(W) and $Q_{CA-estimate}$ $(t10_w)$ if M^3RT is used (Q_{t10_W} otherwise).

$$L(W+1) = f_{NNC}\,[Q_{vector}(W), Queue_{CA\_estimate}(t10_w) \text{ or } Q_{t10_W}\,] \ldots\ldots (3.1)$$

$$Q_{vector}(W) = \{Q_{t1_W}, Q_{t2_W}, Q_{t3_W}, Q_{t4_W}, Q_{t5_W}, Q_{t6_W}, Q_{t7_W}, Q_{t8_W}, Q_{t9_W}, Q_{t10_W}\} \ldots\ldots (3.2)$$

Controller ↓ **Mean deviation** →	**Case 1**	**Case 2**	**Case 3**
NNC (no M^3RT support)	0.02386	0.01853	0.02245
NNC&M^3RT	0.02260	0.01655	0.02115
Performance improvement: $\frac{NNC-"NNC\,\&\,M^3RT"}{NNC}*100\%$	5.28%	10.7%	5.8%

Table 2. Comparison of three cases of mean deviations (with $\Delta = 0.2$ and $k = 7000$)

The evaluation of the smoothness of the O-NNC convergence is obtained by measuring the *mean deviation* (MD) of the control output from the QOB_R reference. That is, $MD = \left[\sum_{i=1}^{k} |\Delta - QOB_i|\right] \Big/ k$, with k as the sample size. For demonstration purposes, Table 2 shows three MD cases, with $\Delta = 0.2$, $QOB_R = 0.8$, and $k = 7000$. The largest MD recorded for the un-optimized NNC with no M^3RT support is 25%. The M^3RT presence lessens the MD to a maximum of only 19%, a 6% improvement. By default the O-NNC always has M^3RT support for better convergence precision and shorter CCT, which is further reduced by the inclusion of HBP.

3.1 The Hessian-based Pruning Technique

The *Hessian Based Pruning* (HBP) technique is proposed in this paper for optimizing the NNC neural network configuration in the adaptive, dynamic, and cyclical manner. In operation the HBP is a renewal process, and the optimization in every HBP renewal cycle has two phases of operations, as shown in Figure 3b:

a) *First phase*: The *Learner* computes the weights of all the arcs in its neural network and marks those insignificant ones with the principle of *dynamic sensitivity analysis*. They are insignificant because they have relatively lower impact on the NNC convergence speed and precision. The phase ends when the *Learner* swaps position with the current *Chief*.
b) *Second phase*: After the *Learner* becomes the *Chief* all the marked arcs are virtually pruned (excluded) from its computation to shorten the control cycle time.

Our literature search indicates that the HBP technique has not been used before for optimizing neural network configurations. This original technique is proposed for the NNC only after a thorough search and investigation of different optimization approaches in literature [Oh98, Goh94, Horn00, Gall92]. The HBP technique is unique because it is for real-time applications and its simplicity is the key to success. Other extant techniques from the literature usually require complex mathematical manipulations. The published previous experience for pruning feed-forward neural network configurations is exclusively off-line in nature. This makes them unsuitable for the on-line NNC application envisaged here. For the sake of supporting NNC pruning on the fly, we could either adapt an extant technique or propose a brand new one. We chose the latter approach because the extant algorithms are mathematically complex and therefore potentially require a long execution time.

The HBP operation is based on concept of *dynamic sensitivity analysis*, and the rationale is to mark and skip a neural network connection if the error/tolerance of the neural computation is insensitive to its presence. For the NNC the error/tolerance is the $\pm\Delta$ band about the QOB_R reference in light of the $\{0,\Delta)^2$ objective function. The core argument of the HBP technique is: "*if a neural network converges toward a target function so will its derivatives* [Gall92]". In fact, the main difference among all the identified *performance-learning* laws from literature [Hagan96] is how they leverage the different parameters (e.g. weights and biases).

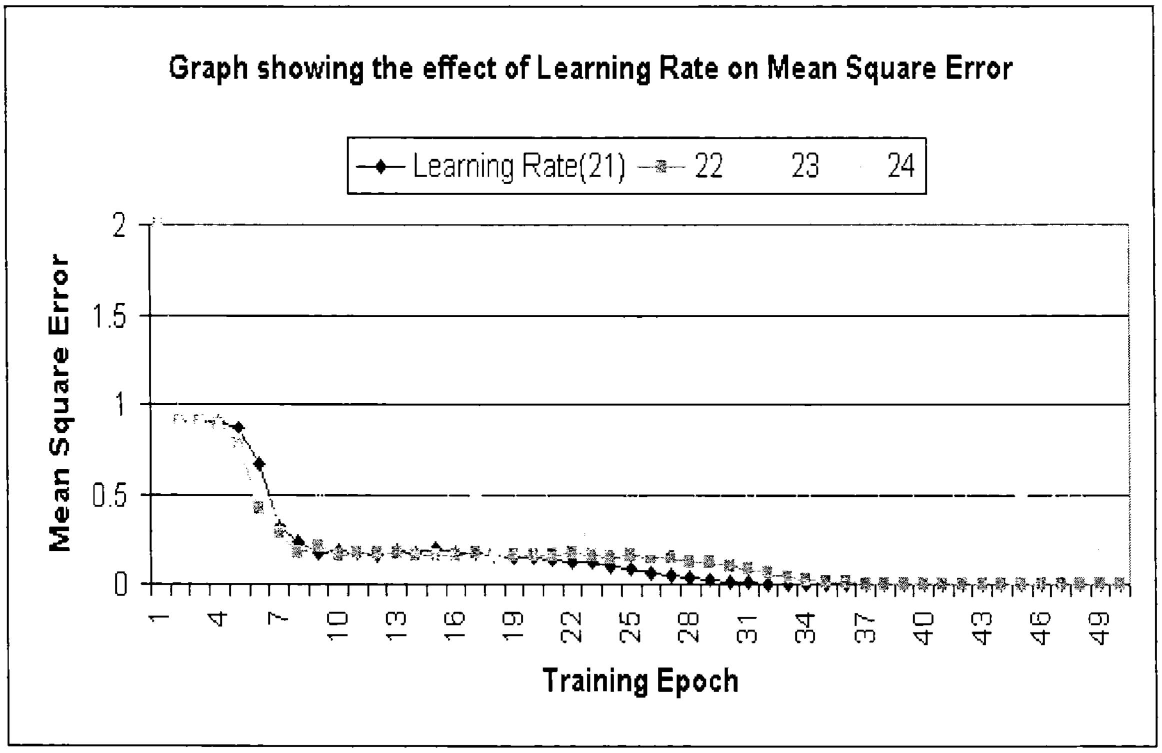

Figure 5. The graph showing the effect of learning rate on mean square error

The HBP adopts the *Taylor Series* [Finn94] (equation (3.3)) as the vehicle to differentiate the relative importance of the different neural network (NN) parameters. The meanings of the parameters in equation (3.3) are: *F()* - the function, *w* - the NN connection weight, *Δw* – the change in *w*, *∇F(w)* - the gradient matrix (i.e. expression (3.4)), and ∇^2F(w) - the *Hessian* matrix (i.e. expression (3.5)). The symbols in the equations mean the following: T for transpose, O for higher order term, n for the n^{th} term, and $\partial/\partial w_1$ for partial differentiation. The expansion about w: $F(w+\Delta w)$ is described by equation (3.3).

$$F(w+\Delta w) = F(w) + \nabla F(w)^T \Delta w + \frac{1}{2}\Delta w^T \nabla^2 F(w) \Delta w + O(\|\Delta w\|^3) + \ldots\ldots (3.3)$$

$$\nabla F(w): \begin{bmatrix} \frac{\partial}{\partial w_1}F(w) & \frac{\partial}{\partial w_2}F(w) \ldots \frac{\partial}{\partial w_n}F(w) \end{bmatrix}^T \ldots\ldots\ldots\ (3.4)$$

$$\nabla^2 F(w): \begin{bmatrix} \frac{\partial^2}{\partial w_1^2}F(w) & \frac{\partial^2}{\partial w_1 w_2}F(w) & \ldots & \frac{\partial^2}{\partial w_1 \partial w_n}F(w) \\ \frac{\partial^2}{\partial w_2 \partial w_1}F(w) & \frac{\partial^2}{\partial w_2^2}F(w) & \ldots & \frac{\partial^2}{\partial w_2 \partial w_n}F(w) \\ \ldots & \ldots & \ldots & \ldots \\ \frac{\partial^2}{\partial w_n \partial w_1}F(w) & \frac{\partial^2}{\partial w_n \partial w_2}F(w) & \ldots & \frac{\partial^2}{\partial w_n^2}F(w) \end{bmatrix} \ldots\ldots\ldots (3.5)$$

The preliminary O-NNC results confirm that the HBP performs effectively as expected. These results and findings concur with similar experience published previously [Oh98]. That is, the weighing factors (*synaptic weights* or *learning rates*) affect the convergence speed. Many different experiments were carried out to study the effect of different learning rates, and one set of these results is chosen and presented in Figure 5 for demonstration purposes. It shows how the correlation between the learning rate and the *mean square error* (MSR) varies. A learning rate is the magnitude of change when a connection weight is adjusted in training. For example, the desired output is $w_{i,j}^m(k+1) = w_{i,j}^m(k) - \alpha\, \partial F(x)/\partial w_{i,j}^m$ with $w_{i,j}^m(k)$ as the current weight and α the learning rate. The MSR, which is defined as: $MSR = \boldsymbol{E}[(\textit{target output} - \textit{actual output})^2]$, measures the control accuracy, where $\boldsymbol{E}$ is the averaging operator. The MSR should decrease when the convergence gets closer to the QOB_R reference. The experimental results, however, indicate that the bigger learning rates may yield oscillatory convergence. This is clearly shown by the rates: 23 and 24 in Figure 5, and in contrast, the smaller rates: 21 and 22 produce much smoother control. Under equation (3.3), the learning/training process should converge to the QOB_R reference, which is mathematically known as the *target global minimum surface*. The convergence makes the gradient vector $\nabla F(w)$ insignificant and eliminates the "$\nabla F(w)^T \Delta w$" term from equation (3.3). This implies not only that the larger ordinal terms in equation (3.3) can be ignored but also a possible simplified form (equation (3.6)) for the equation. Further simplification of equation (3.6), based on: $\Delta F=F(w+\Delta w)-F(w)$, yields equation (3.7).

$$F(w+\Delta w) = F(w) + \frac{1}{2}\Delta w^T \nabla^2 F(w) \Delta w \ldots(3.6), \qquad \Delta F = \frac{1}{2}\Delta w^T \nabla^2 F(w) \Delta w \ldots(3.7)$$

To recap, the HBP optimization cycle has two consecutive phases. The first one is applied only to the *Learner* and the second only to the *Chief*. The details involved in these phases are as follows (first three points belong to the first phase and the fourth point is the second phase):

a) Use Taylor series (equation (3.3)) to identify the significant neural network parameters.

b) Choose appropriate learning rates for the significant parameters to avoid convergence oscillations, as illustrated in Figure 5.

c) Mark the synaptic weights that have insignificant impact on the Taylor series (1st phase ends here).

d) After the *Learner* has become the *Chief*, it excludes all the marked connections in its neural computation. The exclusion, which is represented by equation (3.8), is, in effect, virtual pruning of the insignificant connections. It is only a logical process because the physical skeletal NNC neural network configuration remains intact. The net effect is the exclusion of the marked connections in the subsequent O-NNC control computation.

The pruning decision is based on the *Lagrangian* index S (to be explained later). Since the optimization starts anew every time the *Learner* has completed training and acquired new weights for its neural network connections, the optimized outcome from every cycle is always unique. This characterizes the dynamic and adaptive nature of the HBP optimization process.

$w_i+\Delta_{wi}=0$...(3.8), $S=\frac{1}{2}\Delta w^T\nabla^2F(w)\Delta w-\lambda(U_i^T\Delta w+w_i)$...(3.9a)

If Δw in equation (3.7) is replaced by equation (3.8), then the *Lagrangian* equation (3.9a) is formed. Now equation (3.3) has become a typical *constrained optimization* problem [Bert82]. The symbols: U_i^T and λ in equation (3.9b), are the unit vector and the *Lagrange* multiplier respectively. The optimum change in the weight vector w_i (equation (3.8)) is shown in equation (3.9b). For every entry in w_i there is a unique *Lagrangian* index S_i (equation (3.9c)). In the first phase of the HBP optimization process the S_i values are sorted so that the corresponding less significant w_i (neural network connection) can be excluded from the *Chief'* neural computation, starting from the lowest S_i. The pruning process stops if the exclusion of the current S_i affects the accuracy and speed of convergence process. Only after the virtual pruning process has completed that the *Learner* would become the *Chief.*

$\Delta_{wi}=-\frac{w_i}{[\nabla^2F(w)^{-1}]_{i,i}}\nabla^2F(w)^{-1}U_i$...(3.9b) $S_i=\frac{w_i^2}{2[\nabla^2F(w)^{-1}]_{i,i}}$...(3.9c)

4. Experimental Results

Different experiments were conducted to verify the efficacy and correctness of the HBP technique and the O-NNC efficacy. The preliminary results confirm that the HBP technique shortens the O-NNC control cycle time consistently. In the experiments the skeletal O-NNC configuration is the same as the NNC prototype (Figure 3a), with 10 input neurons, 20 neurons for the hidden layer, and one output neuron. This configuration is fully connected, with 200 connections between the input layer and the hidden layer, as well as 20 connections between the hidden layer and the output layer. The O-NNC result in Figure 6 is produced by the configuration that has a hidden layer of 187 arcs instead of the 220 full connections. This is so because 33 of original 220 arcs are pruned by the HBP on the fly. The different experimental results confirm that the O-NNC indeed has the capability to yield the same level of dynamic buffer tuning and overflow efficacy as for the un-optimized NNC, but with a shorter convergence time to reach QOB_R. Figure 7 shows how the O-NNC always ensures the current QOB value to be within the given tolerance band of $|2\Delta|$ ($QOB_R=0.8$). It also compares the QOB deviation profiles of the three controllers: NNC, O-NNC and A-PID. As shown by Table 3, the O-NNC, however, has a larger mean deviation (MD) than the un-optimized NNC, $MD = \left[\sum_{i=1}^{k} |\Delta - QOB_i|\right] / k$.

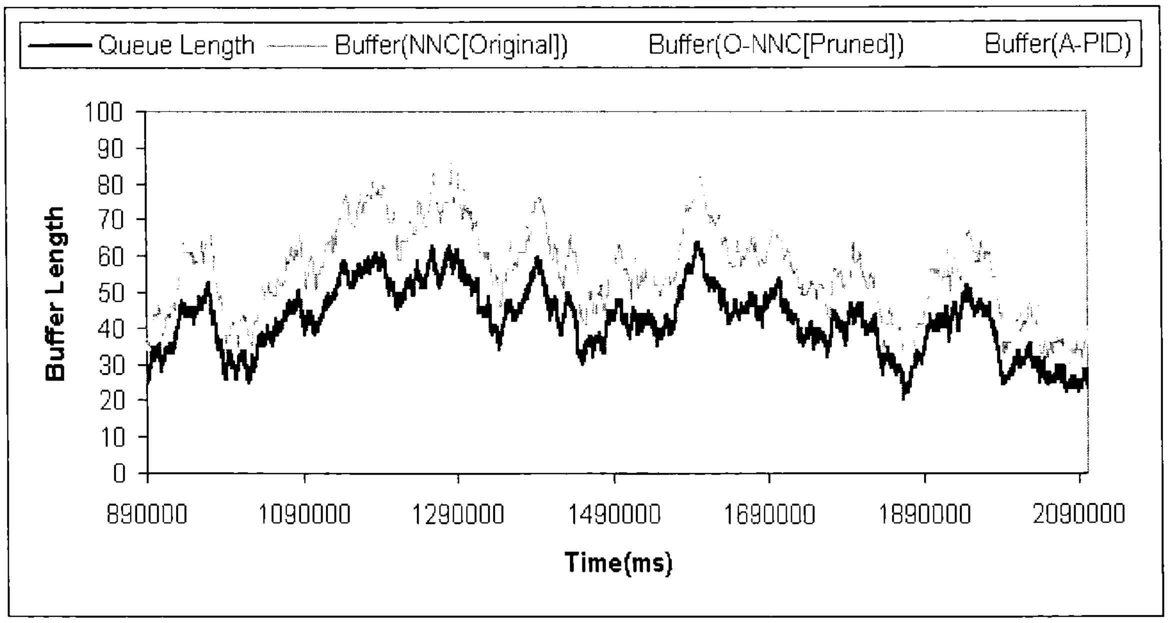

Figure 6. A set of experimental results to compare NNC, O-NNC and A-PID

Controller/tuner	Mean Deviation
NNC (Original)	0.0536
O-NNC (Pruned)	0.0916
PID	0.1279

Table 3. Mean deviations for Figure 7.

Controller/tuner	The measured average number of clock cycles per tuner control cycle
NNC (Original and un-optimized)]	10800
O-NNC (Pruned/optimized)	9250 ($9250/10800 \approx 0.857$; 85.7%)

Table 4. Comparing the average number of clock cycles per tuner cycle

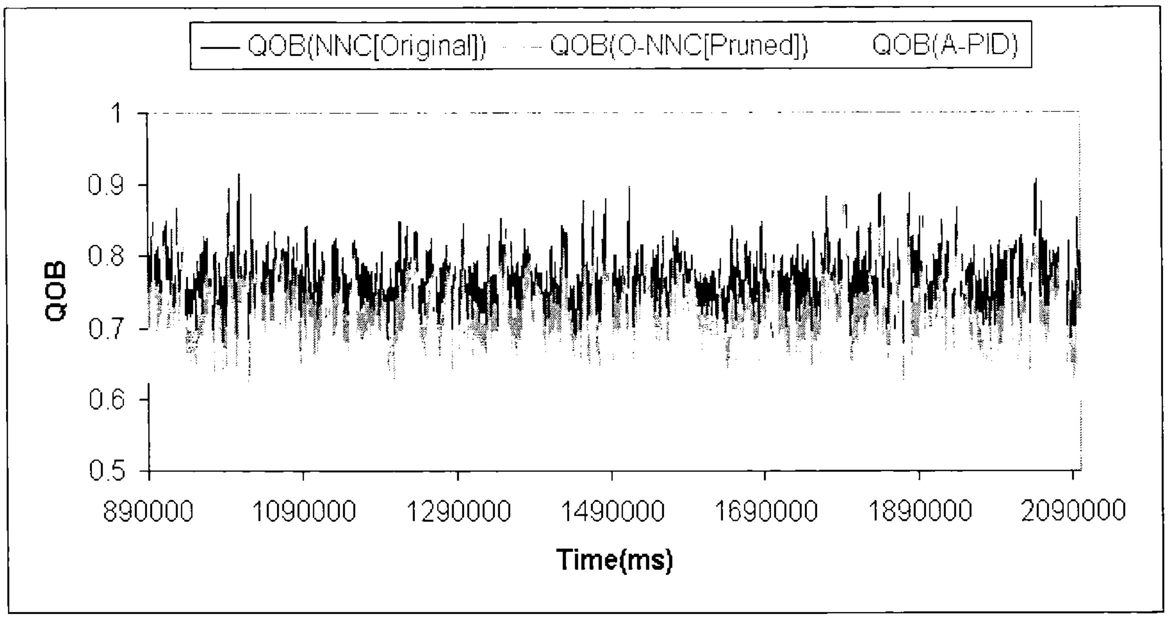

Figure 7. Indication of the HBP convergence stability

The average control cycle time or CCT for the O-NNC is only 9250 clock pulses compared to the 10800 for the NNC (Table 4). The average control cycle time or CCT for the O-NNC is only 9250 clock pulses compared to the 10800 for the NNC (Table 4). The CCT in clock pulses are measured with the *Intel's VTune Performance Analyzer* [VTune], and they can be converted easily into the *physical control cycle time* (*P-CCT*) for any platform by $P - CCT = CCT * \frac{1}{Hz}$, where *Hz* is the platform's speed in hertz.

Figure 7 also compares the three controllers O-NNC, NNC and A-PID in terms of the convergence smoothness. Figure 8 provides more convergence stability details for Figure 7, in terms of the individual deviations over time from the QOB_R reference chosen for the $\{0,\Delta\}^2$ objective function. The performance of the NNC (Original) and the O-NNC (Pruned) performs better than A-PID with respect to the deviation error. Figure 9 is another comparison of the three controllers. Figure 9a compares their efficacy in the dynamic buffer adjustment/tuning process. Figure 9b compares the QOB profiles of the three controllers over time, and Figure 9c to Figure 9e show the deviations of the individual controllers. From the preliminary experimental results we conclude that both the NNC and the O-NNC performs equally well as the A-PID, but without the latter's shortcomings. Despite its consistency of converging accurately to the QOB_R, the O-NNC dynamic buffer tuning process is more oscillatory than its un-optimized NNC predecessor. The oscillation is an undesirable side effect from the dynamic HBP optimization cycles. In the future work this problem will be investigated so that appropriate solutions could be proposed.

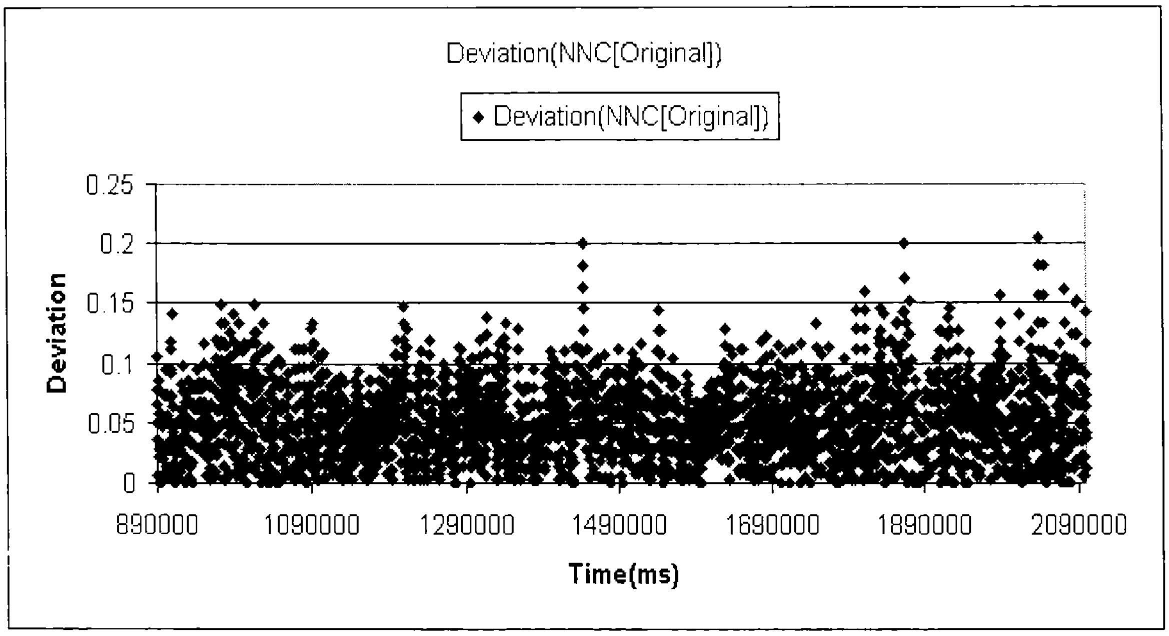

Figure 8a. Deviation profile of the original NNC

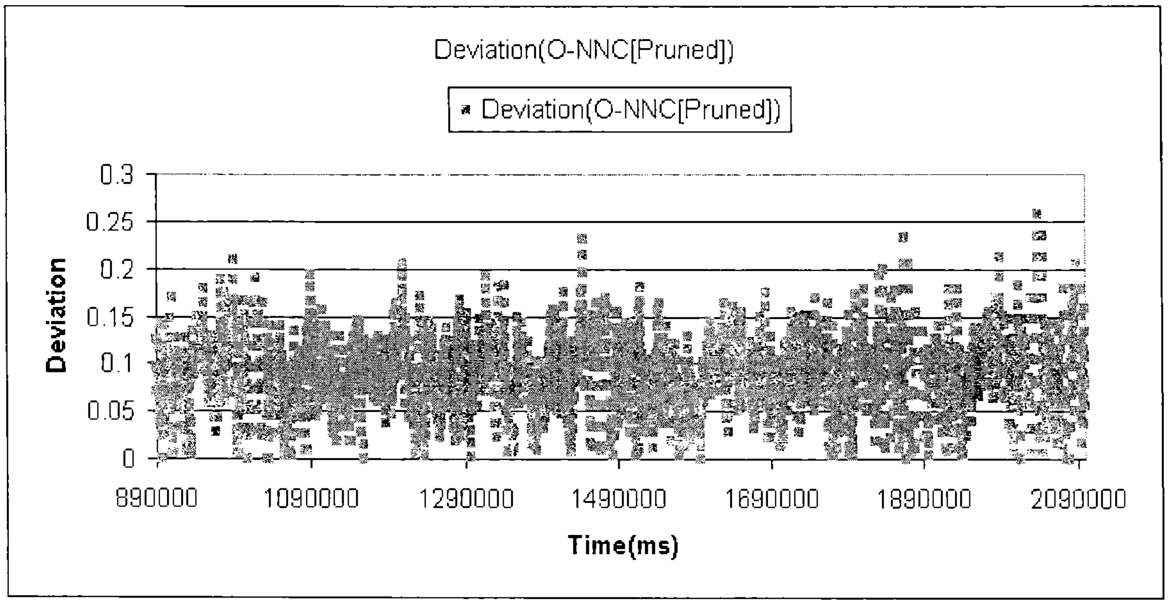

Figure 8b. Deviation profile of the O-NNC

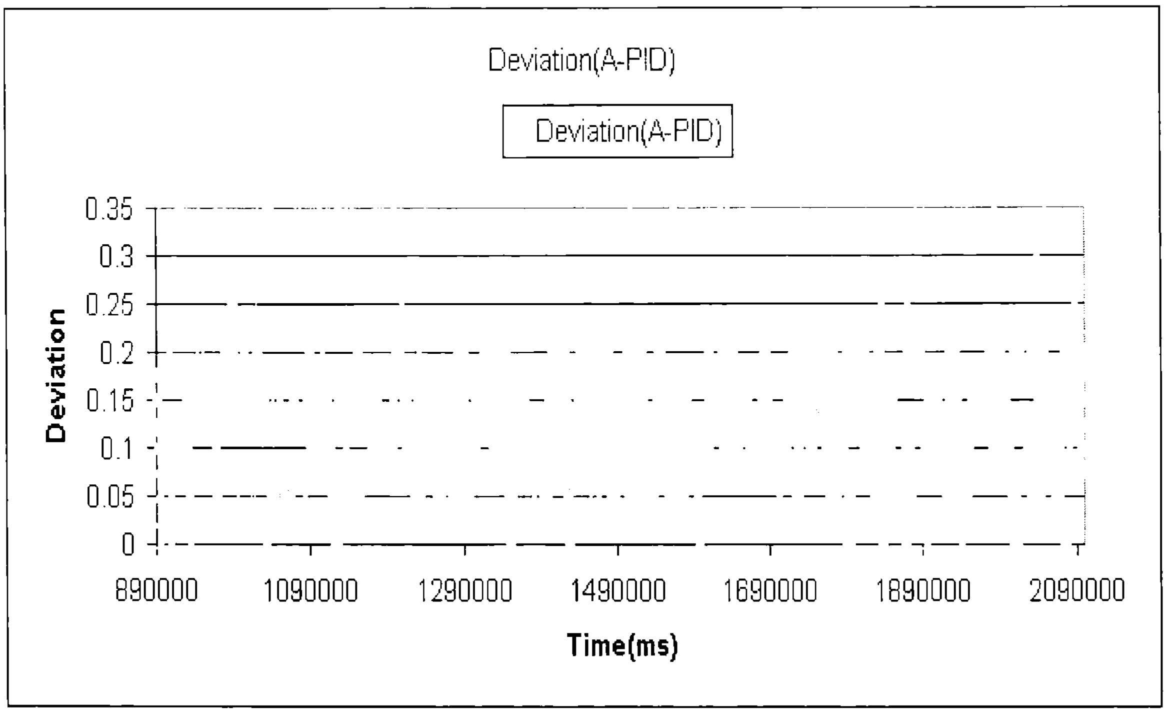

Figure 8c. Deviation by the A-PID controller

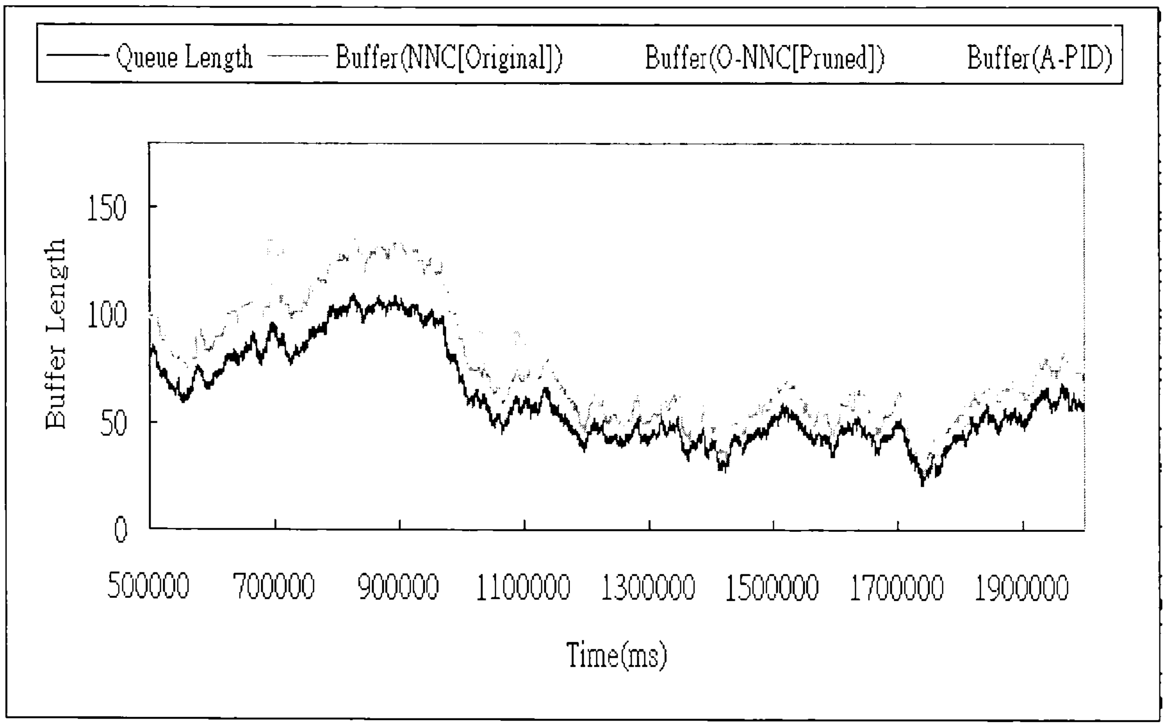

Figure 9a. Another comparison of three controllers

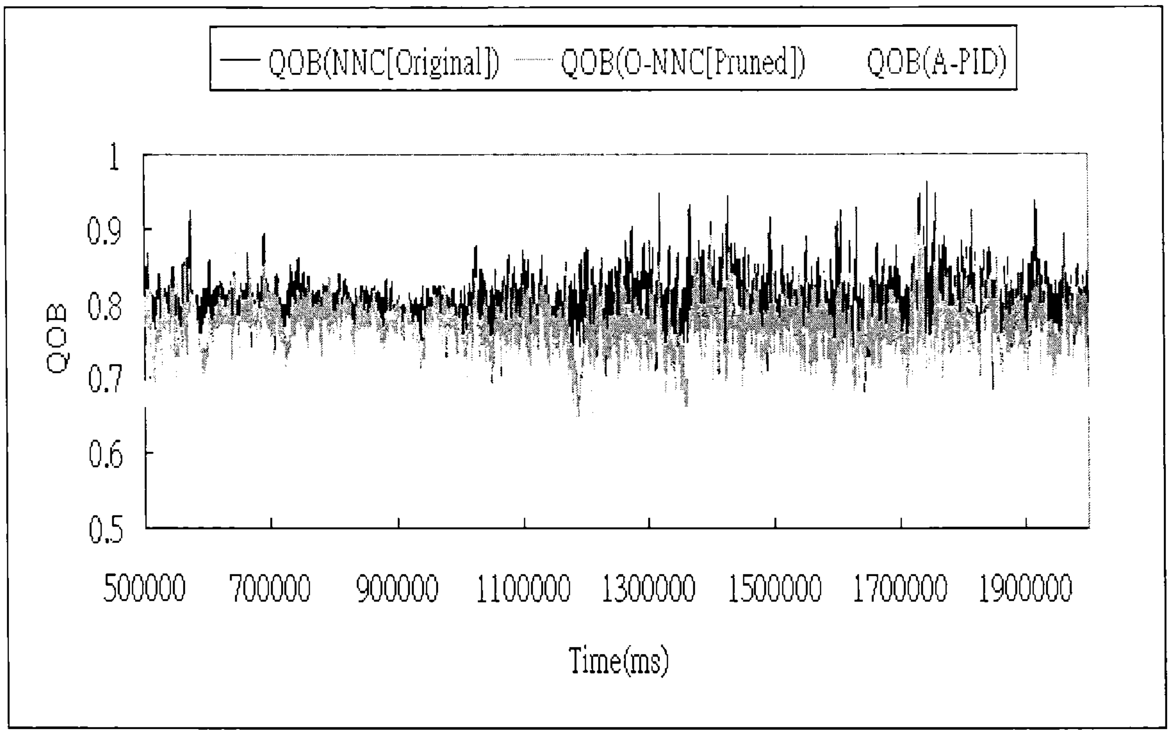

Figure 9b. The QOB profiles of the three controllers in Figure 9a

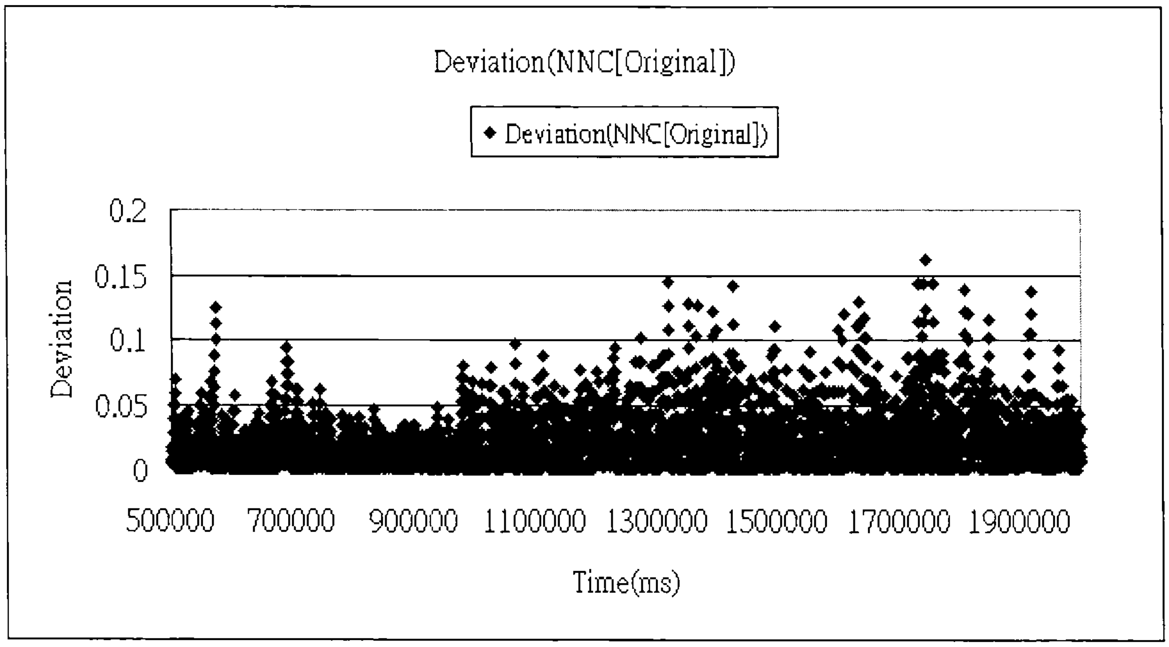

Figure 9c. The deviation profile by the original NNC

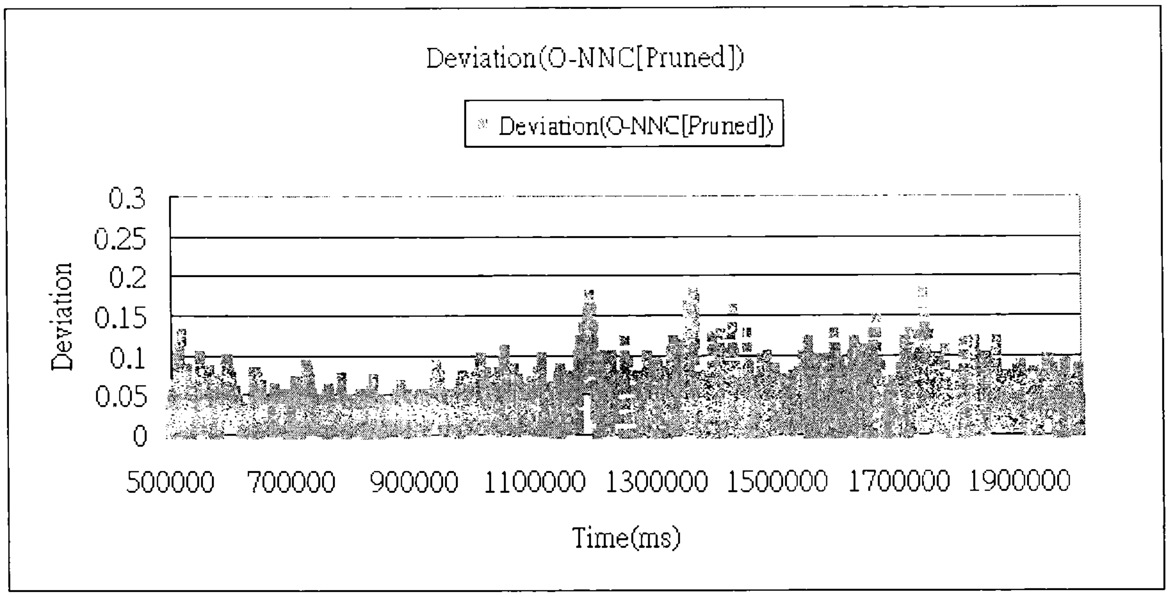

Figure 9d. The deviation profile by the O-NNC

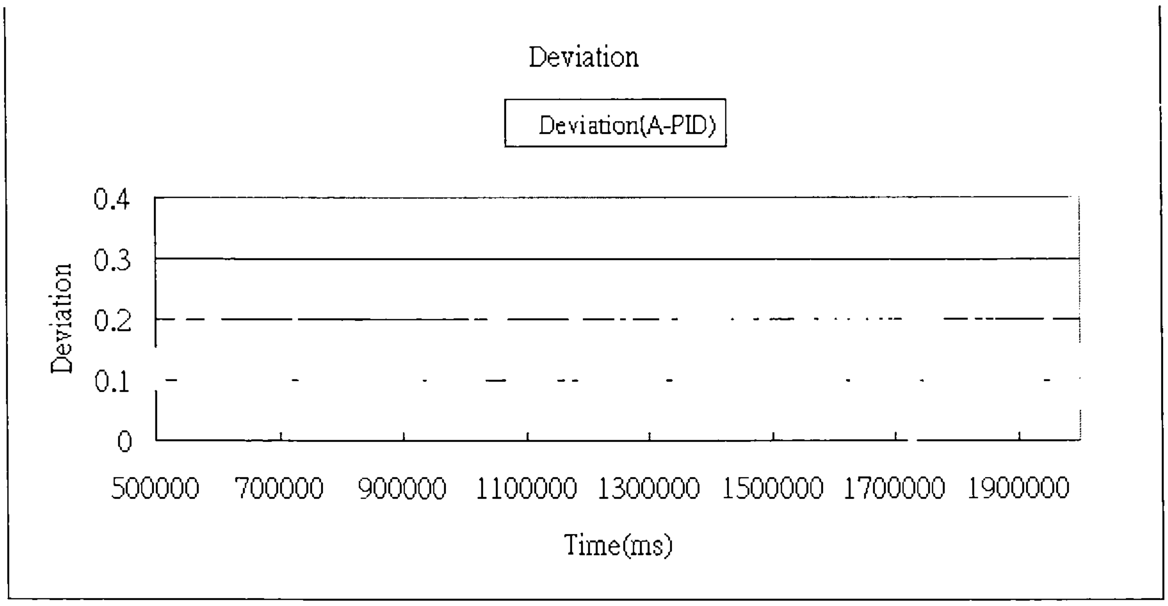

Figure 9e. The deviation profile by the A-PID

5. Conclusion

The HBP technique is proposed for optimizing the NNC neural network configuration in a dynamic manner. The aim is to shorten the NNC control cycle time, and the "*HBP+NNC*" combination is called the *Optimized NNC* or O-NNC model. This model is dynamic because when the O-NNC is not running it a system of two NNC clones. The preliminary experimental results confirm that the aim is successfully achieved because the O-NNC always yields a shorter CCT than its original un-optimized NNC predecessor (about 14.3 percent better). The HBP technique in action is a renewal process and every optimization cycle has two consecutive phases: "*Learner + Chief*". In the first phase, the *Learner* computes the weights for the neural network arcs anew and then marks the insignificant ones in a relative manner. In the second phase the *Chief* excludes the marked arcs from its computation to shorten the control cycle time. The first phase always starts with the original skeletal NNC configuration. The marking and exclusion of the insignificant arcs underlie the HBP concept of virtual pruning. The pruning process is only logical because it removes no physical connection from the skeletal NNC configuration at all. The present HBP optimization technique is based on the Taylor series. In the next stage of the research the following issues will be explored and addressed: a) replacing the Taylor series with other techniques to support the virtual pruning process, b) damping the possible oscillations in the HBP optimization process, and c) the impact of different Internet traffic patterns on the stability and efficacy of the HBP technique and the O-NNC. This third issue is important for successful O-NNC deployment over the Internet because in the network traffic pattern changes without warning. For example, it may switch suddenly from being SRD (*short-range dependence*) to LRD (*long-range dependence*).

6. Acknowledgement

The authors thank the Hong Kong PolyU for the grant G-T426.

7. References

[CA01] A.K.Y. Wong and J.H.C. Wong, A Convergence Algorithm for Enhancing the Performance of Distributed Applications Running on Sizeable Networks, International Journal of Computer Systems Science & Engineering, 16(4), July 2001

[Feng99] W. Feng, D. Kandlur, D. Saha and K. Shin, Blue: A New Class of Active Queue Management Algorithms, CSE-TR-387-99, University of Michigan, USA, 1999

[Laks97] T. Lakshman U. Madlow, The Performance of TCP/IP for Networks with High Bandwidth-Delay Products and Random Loss, IEEE/ACM Transactions on Networking, 5(3), June 1997

[PRDC99] Allan K.Y. Wong and Tharam S. Dillon, A Fault-Tolerant Data Communication Setup to Improve Reliability and Performance for Internet-Based Distributed Applications, the Proceedings of the 1999 Pacific Rim International Symposium on Dependable Computing
(PRDC'99), Hong Kong (SAR), P.R. China, December 1999

[PID01] May T.W. Ip, Wilfred W.K. Lin, Allan K.Y. Wong, Tharam S. Dillon and Dian Hui Wang, An Adaptive Buffer Management Algorithm for Enhancing Dependability and Performance in Mobile-Object-Based Real-time Computing, Proc. of the IEEE ISORC'2001, Magdenburg, Germany, May 2001

[NNC03] Wilfred W.K. Lin, Allan K.Y. Wong and Tharam S. Dillon, HBM: A Suitable Neural Network Pruning Technique to Optimize the Execution Time of the Novel Neural Network Controller (NNC) that Eliminates Buffer Overflow, Proc. of Parallel and Distributed Processing Techniques and Applications (PDPTA), vol. 2, Las Vegas USA, June 2003

[Awey02] J. Aweya, M. Ouellette and D.Y. Montuno, Multi-level Active Queue Management with Dynamic Thresholds, Computer Communications, 25(8), May 2002

[VTune] Intel Vtune, http://developer.intel.com/software/products/vtune/

[Bert82] D.P. Bertsekas, Constrained Optimization and Lagrange Multiplier Methods, New York, Academic Press, 1982

[Horn00] K. Hornik, M. Stinchocombe, and H. White, Universal Approximation of an Unknown Mapping and Its Derivatives Using Multiplayer Feedforward Networks, Neural Networks, vol. 3, 1990

[Gall92] A. R. Gallant and H. White, On Learning the Derivatives of an Unknown Mapping and Its Derivatives Using Multiplayer Feedforward Networks, Neural Networks, vol. 5, 1992

[Finn94] R. L. Finney, G. B. Thomas and M. D. Weir, Calculus, Addison-Wesley, 1994

[Hagan96] M. Hagan et al, Neural Network Design, PWS Publishing Company, 1996

[Moody96] J. Moody and P. J. Antsaklis, The Dependence Identification Neural Network Construction Algorithm, IEEE Transactions on Neural Networks, 7(1), January 1996

[Oh98] S.H. Oh, S.Y. Lee, S. Shin and H. Lee, Adaptive Learning Rate and Limited Error Signal for Multilayer Perceptrons with nth Order Cross-Entropy Error, Proc. of the IEEE World Congress on Computational Intelligence, The 1998 IEEE International Joint Conference on Neural Network, vol. 3, May 1998

[Goh94] Y.S. Goh and E.C. Tan, Pruning Neural Networks During Training by Backpropagation, Proc. of IEEE Region 10's Ninth Annual International Conference, Theme: Frontiers of Computer Technology, vol. 2, 1994

[Lin02] Wilfred W.K. Lin, An Adaptive IEPM (Internet End-to-End Performance Measurement) Based Approach to Enhance Fault Tolerance and Performance in Object Based Distributed Computing over a Sizeable Network (Exemplified by the Internet), MPhil Thesis, Department of Computing, Hong Kong PolyU, 2002

[Tan03] A.S. Tanenbaum, Computer Networks, 4th Edition, Prentice Hall, 2003

[Brad98] B. Braden et al., Recommendation on Queue Management and Congestion Avoidance in the Internet, RFC2309, April 1998

[Ren02] F. Ren, Y. Ren and X. Shan, Design of a Fuzzy Controller for Active Queue Management, Computer Communications, vol. 25, 2002

[Fir00] V. Firoiu and M. Borden, A Study of Active Queue Management for Congestion Control, Proc. of the INFOCOM2000, March 2000

[Mit98] I. Mitrani, Probabilistic Modelling, Cambridge University Press, 1998

[GAC02] Allan K.Y. Wong, Wilfred W.K. Lin, May T.W. Ip, and Tharam S. Dillon, Genetic Algorithm and PID Control Together for Dynamic Anticipative Marginal Buffer Management: An Effective Approach to Enhance Dependability and Performance for Distributed Mobile Object-Based Real-time Computing over the Internet, Journal of Parallel and Distributed Computing (JPDC), vol. 62, September 2002

[IEPM99] L. Cottrel, M. Zekauskas, H. Uijterwaal, and T. McGregor, Comparison of Some Internet Active End-to-End Performance Measurement Projects, http://www.slac.stanford.edu/comp/net/wan-mon/iepm-cf.html, July 1999

[M^{3}RT02] Allan K.Y. Wong, May T.W. Ip and Tharam S. Dillon, M3RT: An Internet End-to-End Performance Measurement Approach for Real-Time Applications with Mobile Agents, Proc. of the ISPAN'02 Conference, 2002

[M^2RT01] Allan K.Y. Wong, Tharam S. Dillon, Wilfred W.K. Lin and May T.W. Ip, M2RT: A Tool Developed for Predicting the Mean Message Response Time for Internet Channels, Computer Networks, vol. 36, 2001

[FLC03] Wilfred W.K. Lin and Allan K.Y. Wong, A Novel Adaptive Fuzzy Logic Controller (FLC) to Improve Internet Channel Reliability and Response Timeliness, Proc. of the 8th IEEE Symposium on Computers & Communications (ISCC'2003), 2003

[Aweya98] J. Aweya, Neurocontroller for Buffer Overload Control in a Packet Switch, IEE Proc. of Communications, 145(4), August/1998

[Floyd93] S. Floyd and V. Jaconson, Random Early Detection Gateways for Congestion Avoidance, IEEE/ACM Trans. on Networking, August 1993

[May00] M. May, T. Bonald and T. Bolot, Analytic Evaluation of RED Performance, Proc. Of INFOCOM 2000, March 2000

[Chris00] M. Christiansen. K. Jeffay, D. OTT and F.D. Smith, Tuning the RED for Web Traffic, ACM SIGCOMM, August 2000

[Lin97] D. Lin and R. Morris, Dynamics of Random Early Detection, Proc. of the SIGCOMM'97, September 1997, 127-138

[Bod00] U. Bodin, O. Schelen and S. Pink, Load-Tolerant Differentiation with Active QueueManagement, ACM SIGCOMM Computer Communication Review, 30(3), July 2000, 4-16

[Cisco] Distributed Weighted Random Early Detection, Cisco Specification, www.cisco.com/univercd/cc/td/doc/product/software/ios111/cc111/wred.pdf

[Pax95] V. Paxson and S. Floyd, Wide-Area Traffic: The Failure of the Poisson Modeling, IEEE/ACM Transactions on Networking 3(3), 1995, 226-244

[Med00] A. Medina, I. Matta and J. Byers, On the Origin of Power Laws in Internet Topologies, ACM SIGCOMM, 30(2), 2000

[Mit99] E. Mitchell, An Introduction to Genetic Algorithms, MIT Press, 1999

[Ott99] T. Ott, T. Lakshman and L. Gong, SRED: Stabilized RED, Proc. of the IEEE INFOCOM1999, March 1999, 1346–1355