Scientia Iranica, Vol. 15, No. 6, pp. 589–604

SCIENTIA
IRANICA

Self-Organization in a Particle Swarm Optimized Fuzzy Logic Congestion Detection Mechanism for IP Networks

C.N. Nyirenda[1] and D.S. Dawoud[1,*]

The Fuzzy Logic Congestion Detection (FLCD) algorithm is a recent proposal for congestion detection in IP networks which combines the good characteristics of both traditional Active Queue Management (AQM) algorithms and fuzzy logic based AQM algorithms. The Membership Functions (MFs) of the FLCD algorithm are designed using a Multi-Objective Particle Swarm Optimization (MOPSO) algorithm, in order to achieve optimal performance on all the major performance metrics of IP congestion control. The FLCD algorithm achieves better performance when compared to the basic Fuzzy Logic AQM and Random Explicit Marking (REM) algorithms. Since the optimization process is undertaken offline and is based on a single optimization script, the performance of the FLCD algorithm may not be optimal under different network conditions, due to the fact that the IP environment is characterized by dynamic traffic patterns. This paper proposes two online self-learning and organization structures that enable the FLCD algorithm to learn the system conditions and adjust the fuzzy rule base in accordance with prevailing conditions. The self-organized FLCD algorithm is compared with the unorganized FLCD, the basic Fuzzy Logic AQM and the Adaptive Random Early Detection (RED) algorithms using simulations with dynamic traffic patterns. Performance results show that the self-organized FLCD algorithm is more robust than the other algorithms. Compared to the unorganized FLCD, the new scheme improves the UDP traffic delay for short round trip times and also reduces packet loss rates. In terms of jitter, fairness and link utilization, it exhibits a similar performance to the unorganized FLCD algorithm.

Keywords: Active queue management; Congestion control; Fuzzy logic; Multi-objective particle swarm optimization; Pareto set.

INTRODUCTION

The internet has experienced tremendous growth over the past two decades and, due to that growth, has some congestion problems. The resulting effects are long delays in data delivery, jitter (delay variations), wasted resources, due to lost or dropped packets and even possible congestion collapse [1], where all communication in the entire network ceases. Long delays in data delivery and jitter reduce the quality of interactive applications, such as telephony, video conferencing and interactive games, since these applications require to be delivered quickly and within certain delay constraints. For these reasons, congestion control mechanisms have been introduced in the Internet. These mechanisms can broadly be classified into three groups:

1. End-to-end algorithms, which control the flow of traffic between the end hosts [2,3];
2. Router based congestion detection, also known as Active Queue Management (AQM) [4-6];
3. The generation and transmission of congestion notification signals to traffic sources [7,8].

Since the focus of this paper is on router based congestion detection (AQM), the remainder of this section will give an overview of the research trends

1. *Radio Access Technologies Centre, School of Electrical, Electronics and Computer Engineering, University of KwaZulu-Natal, Durban, South Africa.*
*. *To whom correspondence should be addressed: dawoudd@ukzn.ac.za.*

in AQM and present the motivating factors for work presented in later sections.

Active Queue Management (AQM) denotes a class of algorithm designed to alleviate the problems of congestion, while, at the same time, ensuring high link utilization and fairness. Research in this area was inspired by the original RED proposal [4]. In 1998, the Internet Engineering Task Force (IETF) recommended the deployment of RED for congestion prevention. Since this IETF recommendation, a plethora of AQM algorithms have been proposed [5,6,9-11]. These algorithms have been classified [6] as heuristic, optimization based and control theoretic. In spite of all these developments, there has been a slow deployment of AQM algorithms in commercial circles. The slow pace of AQM deployment can partly be attributed to the fact that these algorithms perform well only for specific objectives and under specific scenarios [6]. Therefore, there is a need to design AQM schemes that will achieve a reasonably high performance for all key AQM objectives.

It has been shown that as capacity or delay increases, the traditional AQM schemes [6] eventually become oscillatory and prone to instability [12]. It is further pointed out [13] that these schemes demonstrate instability with the introduction of high bandwidth-delay links, because they still require a careful configuration of non-intuitive control parameters. As a result, they are non-robust to dynamic network changes. They exhibit greater delays than the target mean queuing delay with a large delay variation, plus large buffer fluctuations and, consequently, cannot control the router queue. It has also been stressed that in practical queuing systems, the mean arrival rate and the mean service rate are frequently fuzzy, i.e. they cannot be expressed in exact terms [14,15]. The European Network for Intelligent Technologies (EUNITE) Roadmap [16] points out that the application of fuzzy control techniques to the problem of congestion in IP-based networks is suitable, due to difficulties in obtaining a precise mathematical model using conventional analytical methods. Fuzzy Logic AQM algorithms, presented in [13,17], use instantaneous queue length and the variation of the queue (traffic incoming rate) as inputs. Three membership functions, for the two inputs and the output, are used. The system output is a probability with which packets are either dropped or marked if Explicit Congestion Notification (ECN) [18] is enabled. ECN is a means of explicitly notifying end-hosts of network congestion by marking, instead of dropping, packets. The performance of these fuzzy AQM algorithms is generally better than that of traditional approaches, such as PI and Adaptive RED. However, their major shortcoming lies in the fact that their control rules and membership functions are obtained through a manual tuning process, which is based on designer insight. The human factor involved in this operation makes it difficult for these algorithms to achieve optimal performance for all the key AQM objectives. The other problem is that these algorithms are generally designed with an assumption that the Internet is predominantly composed of TCP traffic, whose sources respond to congestion notification signals from routers by reducing their sending rates. Actually, this is not the situation because, apart from the non-responsive UDP traffic, which accounts for $(22 \pm 11)\%$ of Internet traffic [19], the Internet is nowadays facing a growing list of non-responsive flows and anomalies, such as Denial of Service (DoS) attacks and routing loops [20]. These flows do not reduce their sending rates during times of congestion, as responsive TCP flow reduces their rates. Therefore, fairness diminishes exponentially as the number of non-responsive flows increase. In [10], Wan et al. propose an Adaptive Fuzzy RED (AFRED) algorithm that employs an online adaptation mechanism. This algorithm uses the instantaneous queue length as the only input variable to determine the packet marking or dropping probability. A novel principle exhibited by AFRED is that the real packet drop ratio (pdr) can at least show the congestion degree coarsely, since heavy (or light) congestion will trigger lots of (or few) packet drops. This proposal falls short in two areas. Firstly, it uses the instantaneous queue length as a sole input variable. As explained in [21], queue size is not a good indicator of the severity of congestion and the level of congestion notifications issued may be too great and bursty, leading to excessive packet loss. Secondly, it uses only packet loss in the adjust process. Other important performance metrics, such as link utilization, fairness, delay and jitter are not considered. In [22], Che et al. propose a prediction-based Fuzzy Approximate Fair Dropping (FAFD) scheme to address the fairness problems of the Approximate Fair Dropping (AFD) [23]. Based on the current queue length and predicted traffic intensity, the control parameters are adjusted by a fuzzy inference system, which is optimized by a Genetic Algorithm (GA). This algorithm yields higher bandwidth utilization and reduces queuing delay jitter while achieving better fairness than the original AFD scheme. It must also be pointed out that, to the authors' knowledge, this is the first fuzzy logic AQM approach in using an optimization algorithm to tune its parameters. However, it is easy to see that the major objective of this algorithm is the issue of fairness in relation to the AFD approach. A fully fledged AQM algorithm must incorporate a wide range of performance metrics, such as loss rate, link utilization, delay and jitter in its design. A Fast Adaptive Fuzzy Controller (FAFC), proposed in [24], uses Lyapunov's Direct Method for stability analysis, based on the mathematical model for the internet [25],

which supports multiple TCP sessions. A classical Proportional Integral Derivative (PID) controller is used for online adaptation. The FAFC algorithm exhibits better queue stability and lower packet loss rates compared to RED and the Proportional Integral Derivative (PID) AQM algorithm [26]. However, the internet mathematical model [25] used in this algorithm is based on the principle that the Internet is predominantly TCP. It neglects the effect of non-responsive flows (such as UDP) and network anomalies, such as Denial of Service attacks and rooting loops [20]. Therefore, the issue of fairness is not addressed in the design.

In order to address some of the aforementioned shortcomings, the Fuzzy Logic Congestion Detection (FLCD) [27] algorithm was recently proposed. The novelty of the FLCD algorithm consists of the fact that it attempts to address the five major performance metrics of internet congestion control, i.e. loss rate, link utilization, delay, jitter and fairness. The FLCD algorithm ensures fairness by employing a Fuzzy CHOKe (CHOose and Keep for responsive flows, CHOose and Kill for unresponsive flows) [28]. CHOKe is a simple stateless algorithm, proposed by Pan et al., that attempts to ensure a fair bandwidth allocation to all flows that share the FIFO based outgoing link of a congested router. It accomplishes this by dropping more packets from high-bandwidth unresponsive flows. The essence of this algorithm is that, when a packet arrives, a random packet is picked from the queue. If the randomly chosen packet is from the same source as the newly arrived packet, both packets are dropped. In the FLCD implementation, the CHOKe algorithm is activated only when congestion is severe. This helps to ensure that packets are not dropped unnecessarily during times of less congestion. The ultimate effect of this implementation is that both link utilization and fairness are maximized at the same time. The other four metrics are used in modeling the internet congestion problem as a multi-objective problem. The parameters of the FLCD algorithm are optimized by using the Adaptive Multi-Objective Particle Swarm Optimization (AMOPSO) algorithm [29], in order to achieve optimal performance of all major objectives of IP congestion control. In [28], the AMOPSO algorithm has been observed to achieve better results when compared to several multi-objective versions of the Genetic Algorithm. The performance of the FLCD algorithm was compared with that of the basic REM [5] and the Fuzzy Logic AQM [17] algorithms. Performance results in [27] show that the FLCD algorithm provides high link utilization whilst maintaining lower jitter and packet loss. It also exhibits higher fairness compared to its basic variant and REM.

Although the FLCD algorithm exhibits good performance, it is prone to poor performance under certain network conditions, because its optimization process is implemented offline, based on a single optimization script, whose topology and traffic mix can obviously not manage to capture all the traffic dynamics, pattern variations and network topologies. Therefore, this paper enhances the performance of the FLCD algorithm by proposing two online self-learning and organization structures that would enable the FLCD algorithm to learn system conditions and adjust itself accordingly, thereby achieving optimal performance in dynamic traffic environments and a wide range of topologies. The first structure adjusts the update interval, in line with the prevailing link propagation delay. This would help to improve the FLCD algorithm's performance, with respect to TCP traffic transmissions, which depend on the value of the Round Trip Time (RTT). The second structure implements a self-learning and adaptation mechanism, based on concepts learnt from the self-organized fuzzy controllers in [30-33]. This mechanism learns the link conditions and periodically proposes adjustments to the fuzzy rule base.

The rest of the paper is organized as follows. First, a brief overview of the Fuzzy Logic Control Theory and a brief overview of the Particle Swarm Optimized FLCD algorithm are given. Then, the online self-learning and organization structures are discussed, and simulation results and a comparative analysis are presented. Finally, the conclusion of this paper is presented.

OVERVIEW OF THE FLCD ALGORITHM

Overview of the Fuzzy Logic Theory and the Mamdani Inference Mechanism

Fuzzy logic is a generalization of classical logic, in which there is a smooth transition between true and false. The basics of fuzzy logic are derived from the fuzzy set theory [34]. In conventional (crisp) sets, members are always fully categorized and there is no ambiguity about membership while in fuzzy sets, the transition from membership to non-membership being gradual rather than abrupt. A fuzzy set, $A \in X$, is characterized by a membership function, $\mu A(x)$, which associates each element in X with a real number in the interval [0, 1.0]. $\mu A(x)$ is known as the grade of membership. Hence, the fuzzy set on the universe of discourse X is defined as:

$$A = \{(x, \mu A(X)) | x \in X\}. \quad (1)$$

Fuzzy rules are the backbone of a fuzzy logic system. A simple fuzzy rule can be written as follows:

$$\text{if } x \text{ is HIGH then } y \text{ is POSITIVE}, \quad (2)$$

where HIGH and POSITIVE are linguistic values defined by fuzzy sets on the universe of discourse, X and Y, respectively. The if-part, if x is HIGH, is known as the antecedent and the then-part, y is POSITIVE, is known as the consequent. A set of linguistic rules used to map fuzzy inputs to outputs is known as the rule base. The functional components of a fuzzy controller are shown in Figure 1.

The fuzzifier calculates suitable sets of degree of membership, called fuzzy sets, for the crisp (discrete) inputs. The inference engine evaluates output fuzzy sets from input sets using the predefined fuzzy rules contained in the rule base. The defuzzifier transforms the output fuzzy set into a crisp number to be useful in the real world. The inference engine calculates the degree of activation for every rule in the rule base. Fuzzy systems employ two types of inference mechanism: The Mamdani approach [35] and the Takagi-Sugeno approach [36]. The FLCD algorithm employs the Mamdami approach. Therefore, an overview of this inference mechanism will now be presented. If the antecedent for rule j contains one variable, the rule's degree of activation is equal to the degree of membership of that single variable. If $\mu_j^1(x_1)$ denotes the degree of membership of input x_1 for rule j, then, μ_j, the degree of activation of rule j, is expressed as follows:

$$\mu_j = \mu_j^1(x_1). \tag{3}$$

If the antecedent for rule j contains more than one variable in the following form:

$$\text{Rule } j\text{: if } A_j^1 \text{ and } A_j^2 \text{ and} \cdots \text{and } A_j^n \text{ then } b_j, \tag{4}$$

where A_j^k is a fuzzy set with membership function, $\mu_j^k : \Re \rightarrow [0,1]$, $j = 1, \cdots, m$, $k = 1, \cdots, n$, $b_j \in \Re$, then, in this case, the degree of activation for rule j is determined using the minimum t-norm as follows:

$$\mu_j = \mu_j^1(x_1) \otimes \mu_j^2(x_2) \otimes \cdots \otimes \mu_j^n(x_k), \tag{5}$$

where $\mu_j^n(x_k)$ is the degree of membership of input x_k.

The degree of activation is inferred as the degree of membership of the output variable upon its fuzzy set, which is defined in the corresponding consequence.

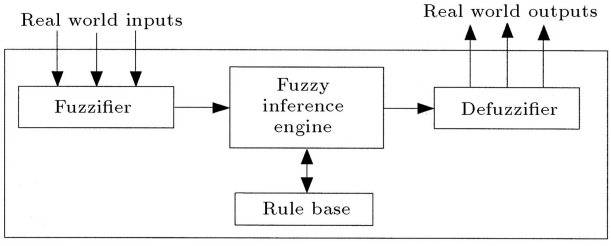

Figure 1. Fuzzy logic controller scheme.

The output of inferring m rules is the aggregation of the individual rule outputs. The implied output sets are combined to formulate a crisp output through a routine known as defuzzification. The widely applied defuzzification method is the Centre Of Gravity (COG) technique, which computes the weighted-average of the centre of gravity of each membership function. The COG of the system with m rules is as follows:

$$y(x) = \frac{\sum_{j=1}^{m} b_j \mu_j}{\sum_{j=1}^{m} \mu_j}, \tag{6}$$

where b_j is the centre of the membership function recommended by the consequent of rule j. The membership functions for the fuzzy controller are initialized by the user, based on a priori knowledge.

FLCD Architecture

The FLCD algorithm is composed of the Fuzzy Logic Control Unit (FLCU), the Probability Adjuster (PA) and the CHOKe Activator (CA). Figure 2 shows the FLCD architecture. A single FIFO buffer, in which all packets are treated equally, is assumed. The queue status is sampled at a period, τ, of 0.002 seconds, just as in [5], in order to obtain the queue-occupation size (backlog), $q(t)$, and the traffic arrival rate, $r(t)$. The backlog, $q(t)$, is translated into the backlog factor, α, which is the ratio of backlog with respect to the Buffer Size, BS, as follows:

$$\alpha = q(t)/BS. \tag{7}$$

In contrast to the proposal in [17], which uses the change in queue length to determine the packet arrival rate, the FLCD algorithm determines the packet arrival rate by counting the actual number of packets that arrive at the buffer (both those that are queued and those that are dropped) during sampling period, τ. When the buffer is prevalently full, the variation of the queue is very small, such that it fails to capture the packet arrival rate, as most packets are dropped before they get queued. Let n denote the number of packets that arrive at the buffer during period, τ, and Let ω_1 denote the measuring weight. The weighted average packet arrival rate, $\overline{r(t)}$, is determined as follows:

$$\overline{r(t)} = \omega_1^* \overline{r(t-\tau)} + (1-\omega_1)^* n. \tag{8}$$

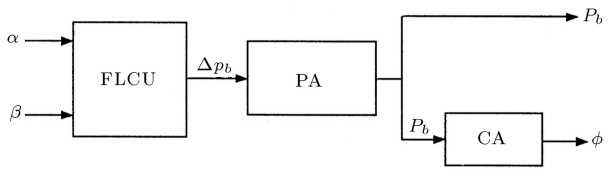

Figure 2. The FLCD architecture.

Also, let r_m denote the upper bound on the weighted packet arrival rate, $\overline{r(t)}$. Therefore, the packet arrival factor, β, is determined as follows:

$$\beta = \begin{cases} \overline{r(t)}/r_m & \overline{r(t)} < r_m \\ 1.0 & \overline{r(t)} \geq r_m \end{cases} \quad (9)$$

The FLCU determines the change in packet marking/dropping probability Δp_b, by using the fuzzified values of parameters α and β. The set of linguistic rules that governs the inference process [35] in the FLCU is shown in Table 1. These rules are derived, based on expert designer knowledge.

The PA computes the new packet marking probability, p_b, as follows:

$$p_b = p_b(t - \tau) + \Delta p_b(t). \quad (10)$$

Packets are either marked (if ECN is enabled) or dropped with probability p_b. Responsive flows react to these events by reducing their sending rates, thereby, reducing congestion at the bottleneck link. In order to address the issue of fairness, in light of non-responsive flows and network anomalies, such as Denial of Service (DoS) attacks and routing loops [20], which may dramatically flood the network as the responsive flows back off, the CHOKe Activator (CA) was incorporated, which uses $p_b(t)$ to generate fuzzy parameter $\phi \in [0, 1]$. Let p_{thresh} denote the CHOKe threshold, then, the fuzzy parameter, ϕ, is derived as follows:

$$\phi = \begin{cases} 0 & p_{\text{thresh}} > p_b \\ \left(\frac{p_b - p_{\text{thresh}}}{1 - p_{\text{thresh}}}\right)^2 & p_{\text{thresh}} \leq p_b \end{cases} \quad (11)$$

When $p_{\text{thresh}} > p_b$ (low congestion), ϕ is 0.0. During this period there is no CHOKe activity. When $p_{\text{thresh}} \leq p_b$ (high congestion), the value of ϕ increases rapidly. As a result, more packets from non-responsive and TCP unfriendly flows are dropped at the bottleneck link. An arriving packet is picked probabilistically, based on the value of ϕ. This packet is compared with a randomly chosen packet from the buffer. If they have the same flow ID, they are both dropped. The randomly chosen packet is kept in the buffer (in the same position as before) and the arriving packet is queued if the buffer is not full; otherwise it is dropped.

Table 1. The FLCU rule base.

if α is low and β is low **then** Δp_b is Negative Big.
if α is low and β is medium **then** Δp_b is Negative Small.
if α is low and β is high **then** Δp_b is Zero.
if α is normal and β is low **then** Δp_b is Negative Small.
if α is normal and β is medium **then** Δp_b is Zero.
if α is normal and β is high **then** Δp_b is Positive Small.
if α is high and β is low **then** Δp_b is Positive Small.
if α is high and β is medium **then** Δp_b is Positive Big.
if α is high and β is high **then** Δp_b is Positive Big.

Figures 3 and 4 show the membership functions (MF1 and MF2) for fuzzifying input parameters α and β, respectively. Figure 5 shows the membership func-

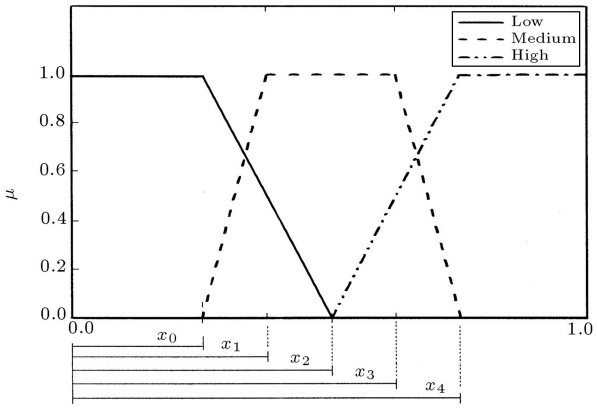

Figure 3. Membership function (MF1) for backlog factor.

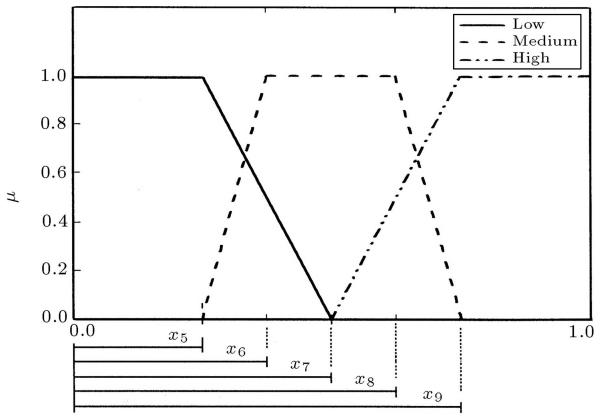

Figure 4. Membership function (MF2) for packet arrival factor.

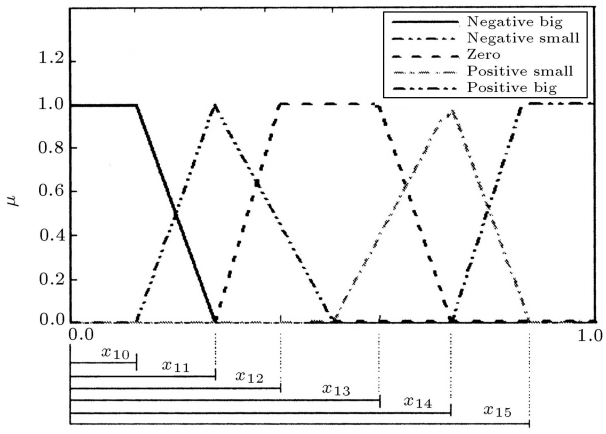

Figure 5. Membership function (MF3) for change in packet marking probability.

tion (MF3) used in the defuzzification process, in order to generate the change in packet marking/dropping probability Δp_b.

The 18-dimensional parameter vector, P, which determines membership functions, is expressed as follows:

$$P = [x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}, x_{14}, x_{15}, x_{16}, x_{17}]. \quad (12)$$

The definition of these elements is presented as follows:

1. x_0, x_1, x_2, x_3, x_4 are parameters for the backlog factor (α) membership function (MF1), as shown in Figure 2;
2. x_5, x_6, x_7, x_8, x_9 are parameters for the packet arrival (β) rate membership function (MF2), similar to Figure 2;
3. $x_{10}, x_{11}, x_{12}, x_{13}, x_{14}, x_{15}$ are parameters for the change in packet marking probability (Δp_b) membership function (MF3), as depicted in Figure 3;
4. x_{16}, x_{17} denote the maximum negative and positive variations (ΔP_{neg} and ΔP_{pos}) of the change in packet marking probability. The output from the defuzzification process, which falls in the range $[0, 1.0]$, is scaled to $[\Delta P_{\text{neg}}, \Delta P_{\text{pos}}]$.

Parameters for individual membership functions must always be sorted in ascending order. For instance, for MF1, the following;

$$x_0 < x_1 < x_2 < x_3 < x_4, \quad (13)$$

must always be true. The same applies to MF2 and MF3.

Parameter Optimization Using MOPSO

The parameters in Equation 12 are modeled as a single 18-dimensional particle, based on which a number of similar particles are randomly created and initialized within a decision variable space whose parameters are predefined. The Adaptive MOPSO (AMOPSO) algorithm [29], which is a special case of the Particle Swarm Optimization (PSO) algorithm [37], is then used to optimize these particles. This algorithm was validated by using three standard test functions. Its performance was compared with the Pareto Archived Evolution Strategy (PAES) [38], the Nondominated Sorting Genetic Algorithm II (NSGAII) [39] and the MOPSO algorithm proposed in [40]. Results in [29] show that the AMOPSO algorithm generates significantly improved Pareto fronts when compared to the other algorithms. The AMOPSO algorithm uses a clustering technique to divide the population of particles into several swarms, in order to have a better distribution of solutions in the decision variable space. Each particle is viewed as a potential solution. The concept of PSO is that each particle randomly searches the decision variable space by updating itself with its own memory and the social information gathered from other particles. This is done over a number of generations/iterations. Unlike basic PSO, which optimizes the particles based on a single objective function, MOPSO is tailored for multiple objective functions, which are usually competing and non-commensurable. In this case, the optimization process generates a pool of non-dominated solutions called the Pareto Optimal Set. The optimization of the FLCD algorithm is based on four objective functions:

1. Maximizing link utilization;
2. Minimizing packet loss rates;
3. Minimizing link delay;
4. Minimization jitter.

The upper bound on packet arrival rate, r_m, is set to 5 packets per sampling period τ of 0.002 seconds. Parameter ω_1 is set to 0.9, in order to avoid drastic changes in the weighted average arrival rate of packets. These settings are done, based on the premise that the other parameters of the algorithm will be realized by the optimization process. For more details on the optimization process, the interested reader is referred to [27].

After the optimization process, a fuzzy inference algorithm is used to draw the best compromise solution from the Pareto optimal set as follows:

$$P = [0.01, 0.02, 0.03, 0.04, 0.29, 0.95, 0.96, 0.97, 0.98, 0.99, 0.01, 0.02, 0.03, 0.34, 0.61, 0.64, -0.0005, 0.0005]. \quad (14)$$

These parameters are used in configuring the membership functions of the practical FLCD algorithm.

SELF-LEARNING AND ORGANIZATION

Self-learning and organization enable the FLCD algorithm to fine tune itself, in light of traffic variations, unmodelled system dynamics and other external disturbances, without disrupting the structure of the optimized membership functions. In order to achieve this, two concepts are introduced: An RTT based sampling rate and a self-learning and adaptation mechanism.

RTT Based Sampling Rate

The motivation to implement an RTT based sampling rate stems from the fact that the rate at which TCP

injects packets into the network is largely dependent on the Round Trip Time (RTT), because TCP is an acknowledgement based end-to-end algorithm. The FLCD algorithm [27] is optimized at sampling rate τ of 0.002 s at link propagation delay D of 0.04 s. If this sampling rate is used on links with shorter propagation delays, the incoming queue would be undersampled. This situation would lead to higher loss rates, due to buffer overflow, because the traffic arrival rate is high. On the other hand, if this sampling rate is used on links with longer propagation delays, the incoming queues would be oversampled, such that the packet arrival rate would always be very low. The effect of this scenario is that the change in packet marking probability will always be low, because the contribution of the packet arrival factor to the fuzzy output value is always low. The system will not be able to increase the packet marking probability in times of congestion, so that it will easily degenerate into a drop-tail mechanism with large losses and underutilization. Assuming that the FLCD router is implemented on the link which offers the largest contribution to the end-to-end propagation delay, such that the other link delays are negligible, the sampling rate can be modified, based on the link propagation delay, by using the following linear relationship:

$$\tau' = \frac{\tau^* D}{0.04}, \tag{15}$$

where τ' and D denote the sampling rate and the propagation delay for the new link. In this paper, it is assumed that the FLCD algorithm is implanted at the interface which contributes t.

The modification of the sampling rate necessitates the adjustment of the upper bound on packet arrival rate r_m in Equation 9. The MOPSO optimization process uses a static value of 5 packets at sampling period τ of 0.002 seconds for this parameter. In order to cater for dynamic situations, while, at the same time, preserving all the performance characteristics yielded by the optimization process, 5.0 is used as a startup value for r_m. If the weighted average packet arrival rate, $\overline{r(t)}$, is greater than r_m, r_m is adjusted by a weighting procedure, otherwise it remains unchanged. This process is illustrated in the following:

$$r_m(t) = \begin{cases} 5.0 & t = 0 \\ \omega_1^* r_m(t-\tau') + (1-\omega_1)^* \overline{r(t)} & \overline{r(t)} > r_m(t-\tau') \\ r_m(t-\tau') & \overline{r(t)} < r_m(t-\tau') \end{cases} \tag{16}$$

where $\omega_1 = 0.9$ is the measuring weight, just as in Equation 8.

The Self-Learning and Adaptation Mechanism

This mechanism adjusts the FLCD algorithm in line with prevailing system conditions. The implementation of online adaptation and self-learning fuzzy systems is an active research area [30-33]. The general trend in these systems is that the rule consequents and membership functions defined in the premises of fuzzy rules are tuned using various algorithms, based on the prevailing plant conditions. For instance, the approach in [33] uses two control blocks: The Adaptation Block (A-Block), which is responsible for adapting the consequents of the main controller's rules to minimize the error arising from the plant output and the Global Learning-Block (GL-Block), which compiles real input-output data obtained from the plant. The A-Block is responsible for the coarse tuning of fuzzy rules at the initial stages. As the process advances, the A-Block gives way to the GL-Block, which fine-tunes both membership functions (premises) and consequents.

The approach in this proposal fine-tunes the rule consequents for only two reasons:

- The membership functions and parameters of the FLCD algorithm have already been optimized offline. With these membership functions and parameters, optimal performance on all the major AQM objectives is guaranteed. Further tuning of membership functions would disrupt their optimal parameter settings, thereby, defeating the whole purpose of the optimization process in [17]. This is not the case with the proposal in [23], in which there is no model of the plant, such that the controller's rules and parameters defining it are optimized from a "void" fuzzy controller.
- It has been reported in [18] that the modification of membership functions uses a lot of memory resources. This process would also take up more of the router's processing time. Therefore, the performance of Internet routers could be adversely affected.

The approach in this paper is built on the principle of monotonicity, which is evident in [30-33]. It evaluates the current state of the plant and proposes correction of the rules responsible for the existence of such a state, either as a reward or a penalty. In [32] and [33], this modification is proportional to the degree with which the rule was activated in achieving the control output, $u(t-d)$, now being evaluated at instant t. The system has to wait d iterations, in order to evaluate $u(t-d)$. This calls for the definition of a queue, with the depth given by the delay of the plant, where the degrees of activation of the rules are stored. While such an arrangement works well in [32] and [33], it is not suitable for the FLCD algorithm. There are two reasons for this assertion. Firstly, the

implementation of a dynamic queue would not only consume the precious memory resources of the router, but would also increase the processing overhead on Internet routers. As mentioned before, this must be minimized at all costs, because the router's primary function is to route packets. Secondly, the evaluation of plant delay in real time is quite a complex process, due to the dynamics of Internet traffic. In light of these observations, the weighted average degrees of activation are used in the adaptation mechanism. If $F_{AB}(t)$ denotes the adjustment parameter and $\overline{\mu_j(t)}$ denotes the weighted average degree of activation for rule j at instant t, then, the proposed change in the output scalar ($b_j(t)$) for rule j would be expressed as follows:

$$\Delta b_j(t) = \overline{\mu_j(t)}.F_{AB}(t). \tag{17}$$

The weighted average degree of activation for rule j is realized as follows:

$$\overline{\mu_j(t)} = \omega_1^* \overline{\mu_j(t-\tau')} + (1-\omega_1)^* \mu_j(t). \tag{18}$$

In real time, the system must be stable under different traffic patterns and network topologies. Therefore, as proposed in [18], the variation of queue length must play a role in the adjustment mechanism. The system must also be capable of adjusting itself, based on the observed packet losses. It has been pointed out in [41] that packet loss can show the degree of congestion coarsely at least. Therefore, in this proposal, $F_{AB}(t)$ is implemented as a sum of the queue error factor $Q_f(t)$ and the packet loss factor $P_f(t)$ i.e. $F_{AB}(t) = Q_f(t) + P_f(t)$. These factors contribute equally to $F_{AB}(t)$, such that:

$$Q_f(t) = P_f(t) = \frac{F_{AB}(t)}{2}. \tag{19}$$

The two parameters, $Q_f(t)$ and $P_f(t)$, defining $F_{AB}(t)$, are discussed in the next subsections.

Evaluation of Queue Error Factor $Q_f(t)$

Let $Q(t)$ denote queue length at instant t. The queue variation at instant t, with respect to Q_{ref}, is expressed as follows:

$$\Delta Q(t) = Q(k) - Q_{\text{ref}}. \tag{20}$$

When $\frac{dQ(t)}{dt} > 0$, it is known that the level of congestion is increasing, hence, the need to increase the packet marking/dropping probability by adjusting the rule consequents in the positive direction. As a result, TCP sources will reduce their sending rates while more UDP packets will be dropped. When $\frac{dQ(t)}{dt} < 0$, it is known that the level of congestion is abating, hence, the need to reduce the packet marking/dropping probability by adjusting the rule consequents in the positive direction. As a result, TCP sources will increase their sending rates, while less UDP packets will be dropped, thereby, increasing the overall utilization of the link. Based on these concepts, queue error factor $Q_f(t)$ can be expressed as follows:

$$Q_f(t) = \begin{cases} C_1.\frac{\Delta Q(t)}{BS} & \Delta Q(t) < 0 \\ \\ C_2.\frac{\Delta Q(t)}{BS} & \Delta Q(t) > 0 \end{cases} \tag{21}$$

where BS is the Buffer Size, just as in Equation 7, while C_1 and C_2 are constants for negative and positive adjustment, respectively. Constants, C_1 and C_2, are directly proportional to the maximum negative and positive variations, (ΔP_{neg} and ΔP_{pos}), of the change in packet marking probability. These relationships are presented mathematically, as follows:

$$C_1 = S_1.|\Delta P_{\text{neg}}|, \tag{22}$$

$$C_2 = S_2.\Delta P_{\text{pos}}, \tag{23}$$

where S_1 and S_2 denote the negative and positive error scaling factors, respectively.

If S_1 and S_2 are too small, the resulting $Q_f(t)$ is also very small, such that the contribution of the adaptation mechanism is negligible. If S_1 and S_2 are too big, the resulting $Q_f(t)$ would be too big, thereby, driving the system into instability. By definition, both $\overline{\mu_j(t)}$ and $b_j(t)$ vary within the range $[0.0, 1.0]$. When $\overline{\mu_j(t)} = 1.0$, $\Delta b_j(t)$ in Equation 17 becomes equal to $F_{AB}(T)$. For stable operation of the system, the range of variation for $\Delta b_j(t)$ is limited to 8% of the maximum value of $b_j(t)$. Therefore, the range of variation for $Q_f(t)$ can be limited to 4%, because $Q_f(t) = F_{AB}(t)/2$, as given by Equation 19. This implies that $Q_f(t)$ would fall within the range $[-0.02, 0.02]$ and Q_{ref} would be between $0.25BS$ and $0.75BS$. To cater for extreme cases, Q_{ref} is fixed within the interval $(0.0, BS)$. If $Q_{\text{ref}} \rightarrow BS$, $\Delta Q(t)$ would fall in the interval $(-BS, 0.0)$, while $Q_f(t)$ is in the range $[-0.02, 0.02]$. Therefore, for an extreme negative variation, let $\Delta Q_f(t) = -BS$ and $Q_f(t) = -0.02$ in the negative component of Equation 21. This yields C_1 as follows:

$$C_1 = 0.02. \tag{24}$$

The optimization process in [27] defines -0.0005 as a value for ΔP_{neg}. Substituting $C_1 = 0.02$ and $|\Delta P_{\text{neg}}| = 0.0005$ into Equation 22 yields the negative variation scaling factor as follows:

$$S_1 = 40.0. \tag{25}$$

If $Q_{\text{ref}} \rightarrow 0$, $\Delta Q(t)$ would fall in the interval $(0.0, BS)$, while $Q_f(t)$ is in the range $[0.0, 0.02]$. Therefore, for an

extreme positive variation, $\Delta Q_f(t) = BS$ and $Q_f(t) = 0.02$ in the positive component of Equation 21. This yields C_2 as follows:

$$C_2 = 0.02. \tag{26}$$

The optimization process in [17] defines 0.0005 as a value for ΔP_{pos}. Substituting $C_1 = 0.02$ and $\Delta P_{\text{pos}} = 0.0005$ into Equation 23 yields:

$$S_2 = 40.0. \tag{27}$$

The values for S_1 and S_2 are the same in this case, because $\Delta P_{\text{pos}} = |\Delta P_{\text{neg}}|$, but cases would easily arise from the optimization process whereby $\Delta P_{\text{pos}} \neq |\Delta P_{\text{neg}}|$. In such cases, the values of S_1 and S_2 would be different.

Evaluation of the Packet Loss Factor $P_f(t)$

In contrast to the queue error factor, which is proactive, this factor is a reactive one. It is based on the notion that an increasing number of lost packets entails that congestion is increasing, hence, the need to increase the packet marking probability. The evaluation of $P_f(t)$ is based on the weighted packet loss rate, $\overline{pdr(t)}$, which is evaluated after every τ' seconds, in keeping with the RTT based sampling rate proposed earlier. This can be expressed as follows:

$$pdr(t) = \frac{ndp(t)}{n(t)},$$

$$\overline{pdr(t)} = \omega_2^* \overline{pdr(t-\tau')} + (1-\omega_2)^* pdr(t), \tag{28}$$

where $ndp(t)$ and $n(t)$ denote the number of dropped packets and the number of arrival packets in the interval $[(t-\tau'), t]$, respectively. $pdr(t)$ denotes the actual packet drop ratio in the interval $[(t-\tau'), t]$, while ω_2 is the weight which assumes the same value as in the earlier sections.

Let $pdr_{\max}$ and $pdr_{\min}$ denote the maximum and minimum packet drop rate. When $\overline{pdr(t)} = pdr_{\min}$, the rule consequents must remain static, because the congestion level is deemed to be within the proper limits. When $\overline{pdr(t)} \rightarrow pdr_{\max}$, it is known that congestion is becoming more severe, hence, there is a need to adjust the rule consequents in a positive direction. This will increase the packet marking/dropping probability and, as a result, the amount of traffic injected into the network will decrease. Based on these concepts, the packet loss factor, $P_f(t)$, can be expressed as follows:

$$P_f(t) = C_3 . \frac{\overline{pdr(t)} - pdr_{\min}}{pdr_{\max} - pdr_{\min}}. \tag{29}$$

Constant C_3 is directly proportional to the maximum positive variations (ΔP_{pos}) of the change in packet marking probability. This relationship is presented mathematically as follows:

$$C_3 = S_3 . \Delta P_{\text{pos}}, \tag{30}$$

where S_3 denotes the loss positive loss scaling factor.

In Equation 29, $P_f(t)$ falls within the range $[0, C_3]$, because $\overline{pdr(t)}$ is restricted to $[pdr_{\min}, pdr_{\max}]$. The range of variation for $\Delta b_j(t)$ and, resultantly, for $F_{AB}(t)$ is limited to 8% of the maximum value of $b_j(t)$. Therefore, the range of variation for $P_f(t)$ can be limited to 4%, because $P_f(t) = F_{AB}(t)/2$, as given by Equation 19. This implies that $P_f(t)$n would fall within the range $[0.0, 0.04]$. Therefore:

$$C_3 = 0.04. \tag{31}$$

Substituting $C_3 = 0.04$ and $\Delta P_{\text{pos}} = 0.0005$ into Equation 30 yields:

$$S_3 = 80. \tag{32}$$

The self-learning and adaptation architecture and algorithm are shown in Figures 6 and 7, respectively.

SIMULATION RESULTS AND COMPARATIVE EVALUATION

Two experiments were conducted on the NS-2.8 simulation platform, in order to compare the Self-Organized FLCD (Self-Org FLCD) algorithm with basic Fuzzy AQM [17], the unorganized FLCD [27] and the Adaptive RED (ARED) [42] algorithms. ARED is an adaptive version of the basic RED [4] algorithm upon whose proposals in [24,41] are benchmarked. ARED is more stable than the basic RED algorithm. It has been pointed out in [42] that ARED is capable of restoring the average queue back to the target range within 10 seconds when traffic rate increases by ten times. The basic RED algorithm does not manage to recover the average queue with such a sharp increase in traffic.

The following metrics are used: Packet loss rate, link utilization, jitter, delay and link fairness. These metrics are presented in the Appendix. The reference queue length is set to 40% of the full buffer size in all three algorithms. All simulations use NS-2.28's NewReno TCP variant with an initial congestion window cwnd of 3 segments (per [43]), a Maximum Segment Size (MSS) of 1500 bytes and the receiver acknowledging each segment. The full buffer size is set to 90 packets and ECN marking is used. The standard web traffic generator included with NS-2.28 is used for the simulations, with the following parameter settings: An average of 30 web pages per session, an inter-page parameter of 0.8, an average page size of 10 objects, an average object size of 400 packets and a Pareto

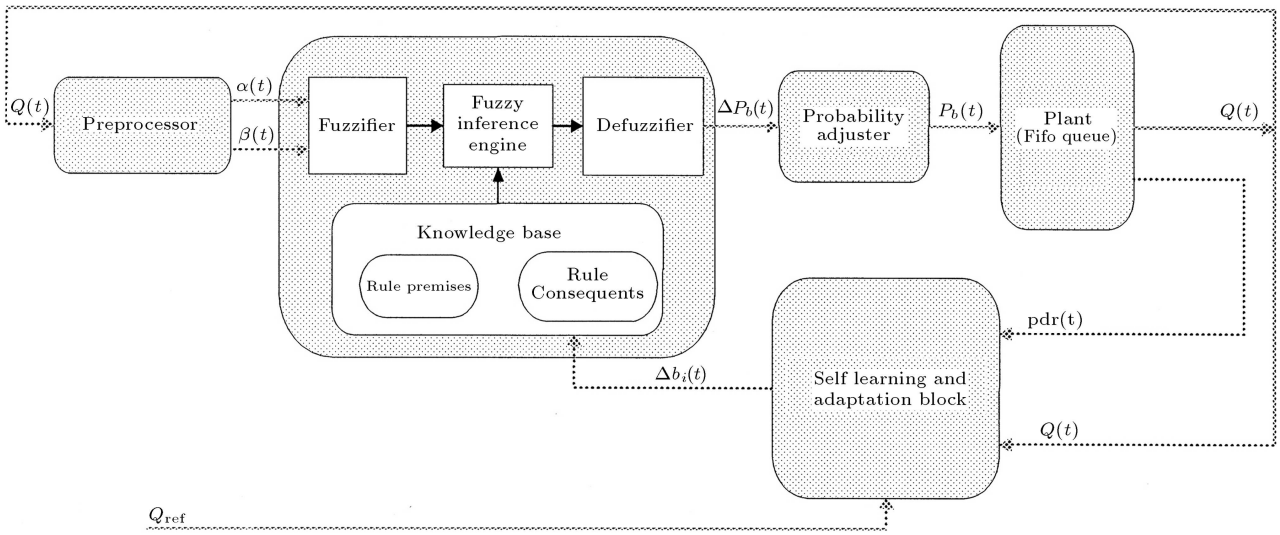

Figure 6. Self-learning and adaptation mechanism for the FLCD architecture.

Initialization:
$\tau' \leftarrow \tau^* D/0.04$;
$pdr_{\max} \leftarrow 0.0$;
$pdr_{\min} \leftarrow 0.0$;
Every τ' seconds:
Evaluate $Q_f(t)$:
if $(\Delta Q(t) < 0)$ $Q_f(t) \leftarrow C_1.\Delta Q(t)/BS$ else $Q_f(t) \leftarrow C_2.\Delta Q(t)/BS$;
Evaluate $P_f(t)$:
Evaluate $\overline{pdr(t)}$;
if $(\overline{pdr(t)} < pdr_{\min}) pdr_{\min} \leftarrow \overline{pdr(t)}$ else $pdr_{\max} \leftarrow \overline{pdr(t)}$;
$P_f(t) \leftarrow C_3.(\overline{pdr(t)} - pdr_{\min})/pdr_{\max} - pdr_{\min}$;
Evaluate $F_{AB}(t)$:
$F_{AB}(t) \leftarrow Q_f(t) + P_f(t)$;
Evaluate $\Delta b_j(t)$ and update $b_j(t)$ for rules $1, 2, \cdots, m$:
for $(j = 0; j < m; j++)\{$
$\overline{\mu_j(t)} \leftarrow \omega_1^* \overline{\mu_j(t-\tau')} + (1-\omega_1)^* \mu_j(t)$;
$\Delta b_j(t) \leftarrow \overline{\mu_j(t)}.F_{AB}(t)$;
Generate $b_j(t)$using the Fuzzy Inference Engine (FIE):
$b_j(t) \leftarrow \text{FIE}\alpha(t), \beta(t))$;
$b_j(t) \leftarrow b_j(t) + \Delta b_j(t);\}$
Evaluate $p_b(t)$:
$\prod \leftarrow 0.0$; $\sum \leftarrow 0.0$;
for $(j = 0; j < m; j++)\{$
$\prod \leftarrow \prod + b_j(t)^* \mu_j(t)$;
$\sum \leftarrow \sum + \mu_j(t);\}$
$\Delta p_b(t) \leftarrow \prod / \sum$;
$p_b(t) \leftarrow \Delta p_b(t-\tau') + \Delta p_b(t)$;

Figure 7. Self-learning and adaptation algorithm.

II shape parameter of 1.002. Thirty web servers are connected to Router 1, with a corresponding number of web clients connected to Router 2. Fifteen web clients are also attached to Router 1 and fifteen web servers to Router 2 to provide background traffic on the return path. Simulations for these experiments are implemented on the network topology in Figure 8.

Experiment 1: Queue Evolution in Dynamic Traffic Environments

In this experiment, the sensitivity of the four schemes is compared when flows are introduced and dropped dynamically during a simulation period of 500 seconds. 50 FTP flows from Router 1 to Router 2 are simulated.

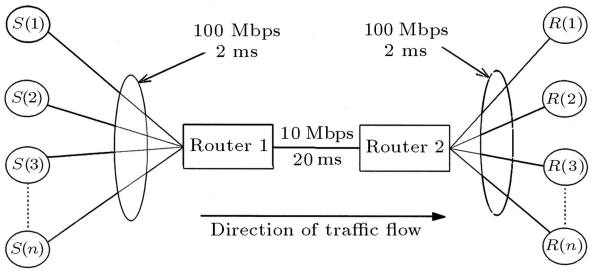

Figure 8. Network topology.

These flows start randomly within the first 5 seconds and remain active throughout the simulation period. At time = 120 seconds, the number of FTP flows from Router 1 to Router 2 is 50. This number is increased by 50 at each 1 second interval until the simulation time reaches 144 seconds. Between 144 seconds and 220 seconds, the number of FTP flow remains constant. When time reaches 220 seconds, the number of FTP flows is reduced by 50 at each 1 second interval until time = 244 seconds, after which the number of FTP flows remains constant. 10 UDP flows from Router 1 to Router 2 are activated at the intervals [120 seconds-130 seconds] and [350 seconds-370 seconds]. UDP traffic rate is set at 0.5 Mbps. 10 web sessions are activated on each client-server connection. Table 2 shows the queue evolution statistics for the four schemes and Figure 9 shows the queue length evolution dynamics for the four schemes.

Table 2 and Figure 10 show that the self-organized FLCD algorithm is more FD robust than the other approaches. The length of the queue sticks to 36 packets (40% of full buffer size). The unorganized FLCD algorithm ranks second, while the basic Fuzzy algorithm and the ARED algorithm rank third and fourth, respectively. Right from the onset, even without the introduction of dynamic traffic, the ARED queue stabilizes at a much higher value. The TCP traffic inflow during slow-start completely overwhelms it, such that it fails to recover the queue length to the desired target. The three other approaches also register high queue length immediately after startup, but they manage to recover and maintain the queue within the precincts of the target. Of the three

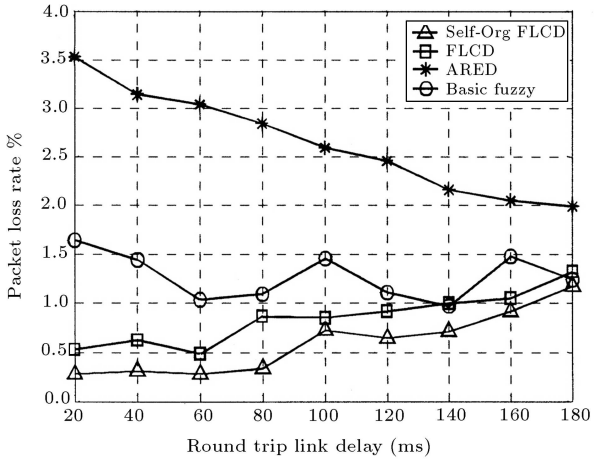

Figure 9. Loss rate with varying round trip link delay.

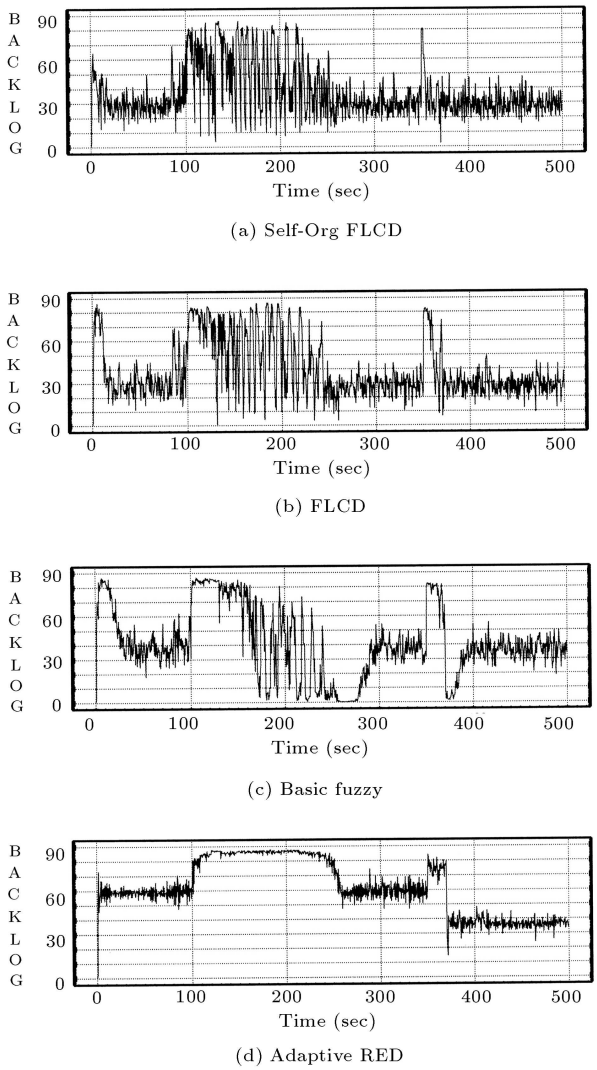

Figure 10. Queue evolution for the four schemes.

Table 2. Queue length evolution statistical results.

Metric	ARED	Fuzzy (Basic)	FLCD	Sel-Org FLCD
Average queue length (packets)	61.6	40.915	38.814	36.74934
Queue variance (packets)	363.348	596.7366	425.872	319.50838

well performing approaches, the self-organized FLCD approach registers the shortest queue length during the startup phase (approximately 60 packets), while the unorganized FLCD algorithm (approximately 82 packets) and the basic Fuzzy algorithm (approximately 85 packets) rank second and third, respectively. In terms of recovery time during the startup phase, the self-organized FLCD algorithm still ranks first with a recovery time of approximately 5 seconds, while the unorganized FLCD algorithm (approximately 10 seconds) and the basic Fuzzy algorithm (approximately 15 seconds) rank second and third, respectively. When UDP traffic is introduced at the intervals [120 seconds-130 seconds] and [350 seconds-370 seconds], the queue's high period is smallest in the self-organized FLCD algorithm compared to the other approaches. With the introduction of dynamic TCP traffic, all the algorithms, except ARED, which just shifts the queue even higher close to the buffer limit, become unstable as they try to limit the queue length to the set target. Once again, the self-organized FLCD algorithm performs better than the unorganized FLCD algorithm and the basic Fuzzy algorithm in that attempts to bring the queue down to 36 packets right from the time dynamic TCP traffic starts entering the link. The self-organized FLCD algorithm also exhibits good recovery performance when the dynamic TCP traffic stops flowing. The basic fuzzy algorithm suffers severe underutilization when dynamic TCP traffic stops flowing. After approximately 390 seconds up to 500 seconds, all the four schemes limit the queue length to approximately 36 packets. It is worthy pointing out that ARED is more stable during this period, but that advantage is offset by its very high average queue length. The effect of long queues is twofold. Besides enhancing the need for larger buffers, long queues have an effect of increasing packet delays.

Experiment 2: Performance in Dynamic Traffic Environments with Varying Propagation Delays

In this experiment, the performances of the four schemes are compared when the round trip delay of the bottleneck link is varied. The network topology shown in Figure 8 is used. The round trip link delay is varied by using 20 ms, 40 ms, 60 s up to 180 ms. The simulations run for 200 seconds. Fifty FTP flows are simulated, competing for bottleneck link from Router 1 to Router 2. These flows start within the first 5 seconds. Four web sessions are activated on each client-server connection. Ten UDP flows are introduced at the intervals [50 s-60 s] and [150 s-160 s], while the FTP flows start randomly within the first 5 seconds and run up to the end of the simulation. At time 60 s, 200 new FTP flows start, with 40 starting every 7.5 seconds. When time reaches 140 seconds, the new FTP flows are removed from the traffic mix in steps of 40 flows every 7.5 seconds. Figures 10-14 show the results.

Figure 10 shows that the self-organized FLCD algorithm has the lowest packet loss rate, ranging from 0.28% to 1.181%. The organized FLCD algorithm comes second, with a loss rate ranging from 0.532% to 1.324%. The basic fuzzy algorithm comes third, with a loss rate ranging from 0.972% to 1.647%. The ARED algorithm comes fourth, with a loss rate ranging from 1.995% to 3.533%. The self-organized FLCD algorithm's low packet loss is due to the role played by the packet loss factor, which is embedded into the self-learning and adaptation mechanism. This factor helps to increase the packet marking probability when packet losses have been detected. This helps to reduce further packet loss. ARED's poor packet loss performance is due to the fact that the ARED control law fails to keep the queue within the desired precincts. When dynamic

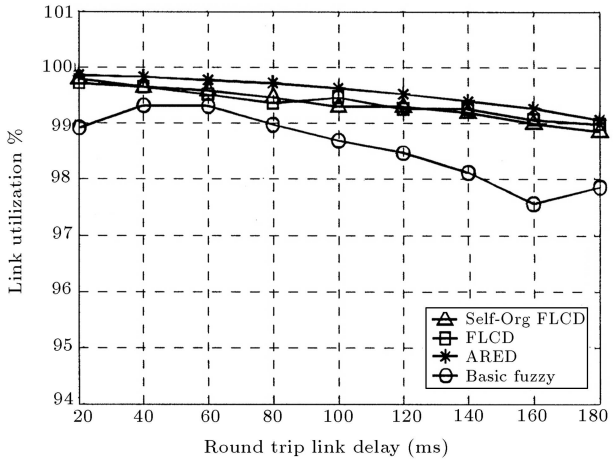

Figure 11. Link utilization with varying round trip link delay.

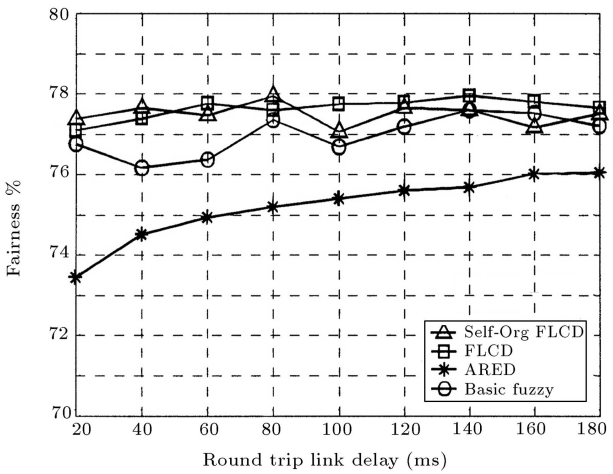

Figure 12. Fairness with varying round trip link delay.

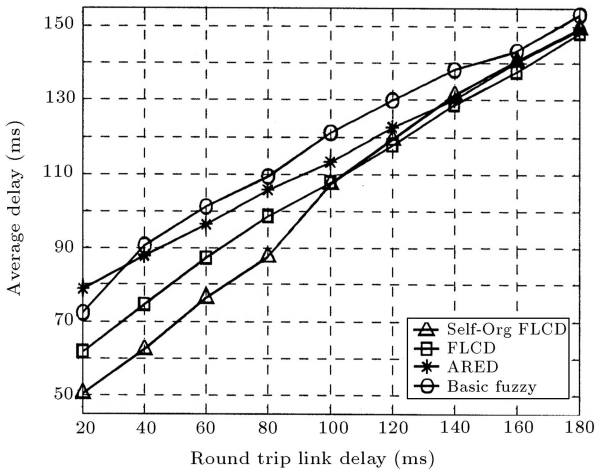

Figure 13. UDP packet delay with varying round trip link delay.

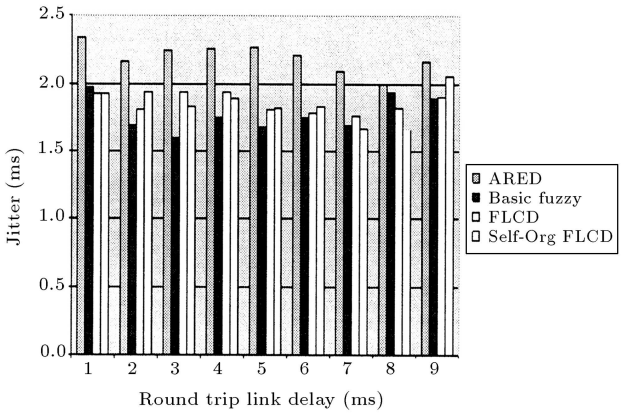

Figure 14. UDP traffic jitter with varying round trip link delay.

traffic sets in, buffer overflows are inevitable, as seen in Figure 9.

Figure 11 shows that ARED and both FLCD approaches achieve high link utilization throughout the simulation run. The average link utilization values are as follows: 99.34% for self-organized FLCD, 99.36% for unorganized FLCD, 99.57% for ARED and 98.58% for basic fuzzy. The basic fuzzy algorithm exhibits lower link utilization, because it suffers from severe underutilization soon after the dynamic TCP traffic stops flowing. It fails to recover the queue to the target after such a sharp decrease in traffic (see Figure 9).

Figure 12 shows that both FLCD approaches achieve the highest average link fairness (77.6% for FLCD and 77.5% for self-organized FLCD). The basic fuzzy algorithm follows them closely with an average of 76.98%, while ARED comes last with an average of 75.2%. From this, it is observed that the self-organized FLCD algorithm does not jeopardize the fairness element of the FLCD algorithm.

Figure 13 shows that the self-organized FLCD algorithm achieves the lowest UDP packet delay (76.8 ms) for round trip link delays shorter than 100 ms. The FLCD algorithm (86 ms) comes second. The ARED (98 ms) algorithm and the basic fuzzy algorithm delays (98.93 ms on average) exhibit similar UDP traffic delays. However, when the round trip delay exceeds 100 seconds, the FLCD algorithms and ARED exhibit a similar UDP traffic delay performance, while the basic fuzzy algorithm exhibits slightly longer delay. The self-organized FLCD algorithm exhibits better UDP delay performance for shorter round trip time because of the RTT based update mechanism, which forms part of the self-learning and organization structure. This mechanism enables the FLCD algorithm to frequently update the packet marking probability for links with shorter RTTs. The effect of this is that buffer overflows are minimized. It becomes easier for the FLCD algorithm to keep the queue close to its target, thereby, improving the end-to-end delay.

Figure 14 shows that the basic fuzzy algorithm exhibits the lowest jitter (with an average of 1.78 ms). The FLCD schemes rank second, with averages of 1.854 ms for FLCD and 1.847 seconds for the self-organized FLCD algorithm. The ARED algorithm comes last, with an average jitter of 2.19 ms.

CONCLUSION

This paper has proposed online self-organization structures for the Fuzzy Logic Congestion Detection (FLCD) algorithm. These structures include an RTT based sampling mechanism and a self-learning and adaptation mechanism. The latter modifies the algorithm's update interval in line with the prevailing outgoing link propagation delay, while the former fine-tunes the algorithm according to the prevailing system conditions. The effectiveness of the proposed approach is proved by comparing the performance of the self-organized FLCD algorithm with that of the unorganized FLCD, the Adaptive RED and the basic fuzzy algorithms under dynamic traffic patterns. Performance results show that the proposed approach shows a more robust performance compared to the other approaches. Apart from enhancing the robustness of the FLCD algorithm, these structures also reduce UDP traffic delay for short round trip propagation delays. They also help to reduce the FLCD algorithm's loss rate. It is also worth mentioning that the addition of the self-organization structures to the FLCD algorithm does not jeopardize other performance metrics, such as utilization, jitter and fairness.

As part of future research work, the authors plan to work on the fine tuning of some of the parameters of the FLCD algorithm, e.g. proposed change in the output scalar ($b_j(t)$).

ACKNOWLEDGMENT

This work was supported in part by the Telkom South Africa.

REFERENCES

1. Nagle, J. "Congestion control in IP/TCP internetworks", *RFC 896*, FACC (1984).
2. Jacobson, V. "Congestion avoidance and control", *ACM SIGCOMM'88*, pp. 314-329 (1988).
3. Floyd, S. and Henderson, T. "The new reno modification to TCP's fast recovery algorithm", *RFC 2582* (1999).
4. Floyd, S. and Jacobson, V. "Random early detection gateways for congestion avoidance", *IEEE/ACM Transactions in Networking*, **1**(4), pp. 397-413 (1993).
5. Athuraliya, S. et al. "REM: Active queue management", *IEEE Network Magazine*, **15**(3), pp. 48-53 (2001).
6. Bitorika, A. et al., *A Comparative Study of Active Queue Management Schemes*, Department of Computer Science, Trinity College Dublin, Ireland (2003).
7. Hadi, S.J. et al. "A proposal for backward ECN for the internet protocol (IPv4/IPv6)", *Internet Draft*, available at: http://www.citesecrx.ist.psu.edu. (1998).
8. Ramakrishnan, K. and Floyd, S. "The addition of explicit congestion notification (ECN) to IP", *RFC-3168* (2001).
9. Hollot, C.V. et al. "Analysis and design for controllers for AQM routers supporting TCP flows", *IEEE Transactions on Automatic Control*, **47**(6), pp. 945-959 (2002).
10. Wang, C. et al. "LRED: A robust active queue management scheme based on packet loss ratio", *IEEE INFOCOM 2004*, Hong Kong (2004).
11. Feng, W. et al. "Blue: A new class of active queue management algorithms", *Technical Report CSE-TR-387-99*, University of Michigan (1999).
12. Low, S.H. et al. "Dynamics of TCP/AQM and a scalable control", *In Proc. IEEE INFOCOM* (2002).
13. Chrystomou, C. et al. "Fuzzy logic congestion control in TCP/IP best-effort networks", *Australian Telecommunications Networks and Applications Conference (ATNAC 2003)*, Melbourne, Australia (2003).
14. Li, R.J. and Lee, E.S. "Analysis of fuzzy queues", *Comput. Math Applications*, **17**, pp. 1143-1147 (1987).
15. Prade, H.M. "An outline of fuzzy or probabilistic models for queuing systems", *Proc. Symp. Policy Anal. Inform. Syst.*, in Durham, NC, pp. 147-153 (1980).
16. Spott, M. et al. "Roadmap contribution IBA C applications in telecommunications, multimedia and services", *European Network on Intelligent Technologies (EUNITE) for Smart Adaptive Systems (SAS)* (2004).
17. Fengyuan, R. et al. "Design of a fuzzy logic controller for active queue management", *Computer Communications*, **25**, pp. 874-883 (2002).
18. Ramakrishnan, K. and Floyd, S. "The addition of explicit congestion notification (ECN) to IP", *RFC-3168* (2001).
19. Fomekov, M. et al. "Longitudinal study of internet traffic from 1998-2003", *CAIDA Tech Report* (2003).
20. Hengartner, U. et al. "Detection of routing loops and analysis of routing loops in packet traces", *ACM SIGCOMM Internet Measurement Workshop*, Marseille, France (2002).
21. Wydrowski, B.P. "Techniques in internet congestion control", PhD Thesis, University of Melbourne (2003).
22. Che, L. and Qiu, B. "Fuzzy approximate fair dropping using genetic algorithm optimization", In *10th Asia-Pacific Conference on Communications*, **1**, pp. 361-365 (2004).
23. Pan, R. et al. "Approximate fairness through differential dropping", to appear in *Computer Communication Review*, **33**(1), available at: http://www.research.att.com/~breslan/papers (2003)
24. Aoul, Y.H. et al. "FAFC: Fast adaptive fuzzy AQM controller for TCP/IP networks", *IEEE GLOBECOM 2004*, Dallas, Texas, USA (2004).
25. Crowcroft, J. and Oeschslin, P. "Services using a weighted proportionally fair sharing TCP", *ACM Computer Communications Review*, **28**, pp. 53-67 (1998).
26. Yanfei, F. and Chuang, F.L. "Design of a PID controller for active queue management", *ISCC 2003* (2003).
27. Nyirenda, C.N. and Dawoud, D.S. "Multi-objective particle swarm optimization for fuzzy logic based active queue management", *IEEE International Conference in Fuzzy Systems (FUZZ-IEEE 2006), World Congress on Computational Intelligence (WCCI 2006)*, Vancouver, Canada, pp. 2231-2238 (2006).
28. Pan, R. et al. "A stateless active queue management scheme for approximating fair bandwidth", *INFOCOM'00*, pp. 942-951 (2000).
29. Pulido, G.T. and Coello Coello, C.A. "Using clustering techniques to improve the performance of a multi-objective particle swarm optimizer", *Genetic and Evolutionary Computation Conference*, Springer-Verlag, Lecture Notes in Computer Science, 3102, Seattle, Washington, USA, pp. 700-712 (2004).
30. Procyk, P. and Mamdani, E. "A linguistic self organizing process controller", *Automatica*, **15**(1), pp. 15-30 (1979).
31. Skrjanc, I. et al. "Direct fuzzy model reference adaptive control", *International Journal on Intelligent Systems*, **17**, pp. 943-963 (2002).
32. Pomares, H. et al. "A new approach for the design of fuzzy controllers in real time", In *Proc. 8th Int. Conf. Fuzzy Systems*, Seoul, Korea, pp. 522-526 (1999).

33. Pomares, H. et al. "Online global learning in direct fuzzy controllers", *IEEE Transactions on Fuzzy Systems*, **12**(2) (2004).
34. Zadeh, L.A. "Fuzzy logic", *IEEE Computer*, **21**(4), pp. 83-93 (1988).
35. Mamdani, E.H. "Application of fuzzy algorithm for control of simple dynamic plant", In *Proceedings of IEEE*, **121**, p 12 (1974).
36. Takagi, T. and Sugeno, M. "Fuzzy identification of systems and its applications to modeling and control", *IEEE Trans. on Systems, Man and Cybernetics*, **15**, pp. 116-132 (1985).
37. Kennedy, J. and Eberhart, R. "Particle swarm optimization", *IEEE International Conference on Neural Networks*, pp. 1942-1948 (1995).
38. Knowles, K.D. and Corne, D. "Approximating the nondominated front using the pareto archived evolution strategy", *Evolution Computation*, **8**(2), pp. 149-172 (2000).
39. Deb, K. et al. "A fast elitist multi-objective genetic algorithm: NGSA-II", *IEEE Transactions on Evolutionary Computation*, **6**, pp. 182-187 (2002).
40. Coello Coello, C.A., Pulido, G.T. and Lechunga, M.A. "Handling multiple objectives with particle swarm optimization", In *Proceedings of the 2002 Congress on Evolutionary Computation. In IEEE Transactions on Evolutionary Computation*, **8**(3), pp. 256-279 (2004).
41. Chonggang, W. et al. "AFRED an adaptive fuzzy-based control algorithm for active queue management", In *Proceedings of the 28th Annual IEEE International Conference on Local Computer Networks* (2003).
42. Floyd, S. et al. "Adaptive RED: An algorithm for increasing the robustness of RED's active queue management", *Technical Report ICSI* (2001).
43. Allman, M. et al., *Increasing TCP's Initial Window*, RFC 3390 (2002).
44. Bitorika, A. et al., *A Framework for evaluating Active Queue Management Schemes*, Trinity College Dublin, Department of Computer Science, Tech. Rep. (2003).
45. Schulzrinne, H. et al., *RTP: A Transport Protocol for Real-Time Applications*, draft-ietf-avt-rtp-new-12.txt, available at: http://cs.uccs.edu/~cs525 (2003).

APPENDIX

METRICS FOR PERFORMANCE EVALUATION

The five general network performance metrics are: Utilization, fairness, packet loss (drop), jitter and delay. The first three metrics, which cater for all traffic flows, have been adopted from [44], while the last two (jitter and delay) have been defined based on the condition that they only cater for real-time flows, whose performance is heavily dependent on delay and jitter.

Utilization, Drop and Fairness Metrics

The total simulation time is denoted by T and the network capacity by C. Let F denote the total (both UDP and TCP) flows indexed by $i \in [1, F]$ traversing the bottleneck link at time T. For flow i, the following variables are defined:

- S_i, the total size of the data received,
- S'_i, the total size of the data sent.

Therefore, the first three metrics are as follows:

1. Utilization metric:

$$\frac{\sum_{i=1}^{F} S_i}{CT}, \tag{A1}$$

2. Drop metric:

$$1 - \frac{\sum_{i=1}^{F} S_i}{\sum_{i=1}^{F} S'_i}, \tag{A2}$$

3. Fairness metric:

$$\frac{\left(\sum_{i=1}^{F} S_i\right)^2}{F \sum_{i=1}^{F} S_i^2}. \tag{A3}$$

Delay and Jitter Metrics

Let N denote the number of UDP (real-time) flows indexed by $j \in [1, N]$ traversing the bottleneck link at time T. For each UDP flow, j, the following variables are defined:

- R_j, the total size of the UDP data received,
- $\overline{D_j}$, the average delay,
- $\overline{J_j}$, the average jitter,
- P_j, the total number of packets.

In UDP flow j, for each packet indexed by $k \in [1, P_j]$, the following variables are defined:

- J_k, jitter between packet k and packet $k+1$,
- D_k, delay for packet k,
- s_k, the time packet k was sent from the sender,
- r_k, the time packet k was received at the receiver.

Delay Metric

For packet k in UDP flow j, the delay is given by:

$$D_k = r_k - s_k. \tag{A4}$$

Average delay is then computed as:

$$\overline{D_j} = \frac{\sum_{k=1}^{P_j} D_k}{P_j}. \tag{A5}$$

Therefore, the weighted average delay, which takes into account the amount of UDP traffic that has been transferred successfully, becomes:

$$\frac{\sum_{j=1}^{N} R_j \overline{D_j}}{\sum_{j=1}^{N} R_j}. \tag{A6}$$

Jitter Metric

The jitter metric is derived based on the definition of jitter for real-time flows [45]. The jitter of a packet stream is defined as the mean deviation of the difference in packet spacing at the receiver compared to the sender, for a pair of packets. Jitter between packet k and packet $k+1$ is expressed as:

$$\begin{aligned} J_k &= |(r_{k+1} - r_k) - (s_{k+1} - s_k)| \\ &= |(r_{k+1} - s_{k+1}) - (r_k - s_k)|. \end{aligned} \tag{A7}$$

Average delay is then computed as:

$$\overline{J_j} = \frac{\sum_{k=1}^{P_j - 1} J_k}{P_j - 1}. \tag{A8}$$

Therefore, the weighted average jitter, which takes into account the amount of UDP traffic that has been transferred successfully, becomes:

$$\frac{\sum_{j=1}^{N} R_j \overline{J_j}}{\sum_{j=1}^{N} R_j}. \tag{A9}$$