Optimized Rule-Based Delay Proportion Adjustment for Proportional Differentiated Service

Sunthiti Patchararungruang, *Student Member, IEEE,* Saman K. Halgamuge, *Member, IEEE,* and Nirmala Shenoy, *Member, IEEE,*

S. Patchararungruang and S.K. Halgamuge are with the Mechatronics Research Group, Mechanical and Manufacturing Engineering, The University of Melbourne, AUSTRALIA, e-mail: sunthiti,sam@mame.mu.oz.au.

N. Shenoy is with the Department of Information Technology, Rochester Institute of Technology, New York, U.S.A., e-mail: ns@it.rit.edu.

Abstract

We present a novel method to adjust output queue delay proportion fairly among the traffic classes of different priorities in relative Differentiated Services (DiffServ). The delay proportion adjustment is based on acceleration of incoming traffic in each class. It aims to reduce the undesirable effects of queue-delay propagation towards higher priority classes, caused by the introduction of bursty data into lower priority classes. We use a fuzzy controller to make the decision for the amount of proportion adjustment, as it is very flexible and adjustable. We suggest an efficient extension to particle-swarm-optimization algorithm for the purpose of optimizing the fuzzy system. The simulation shows that dependency of high-priority-class delay, which is a value that indicates QoS of the traffic, on lower priority classes is significantly reduced by the proposed delay proportion adjustment.

Index Terms

Relative Differentiated Services, Fuzzy Controller, QoS, and Particle-Swarm Optimization

I. INTRODUCTION

As the Internet becomes an important infrastructure of global communication, the best-effort service cannot meet diverse expectations of applications. Some applications such as world-wide-web (www) and file transfers prefer low data-loss rate while tolerate high delay. However, multimedia applications require low delays but tolerate certain amount of data loss. These preferred conditions needed by the applications for providing good services is called quality-of-service (QoS).

There are two paradigms proposed to provide QoS for Internet applications. They are Integrated Services (IntServ) [1] [2] [3] and Differentiated Service (DiffServ) [4] [5]. IntServ intends to ensure end-to-end and per-flow QoS. All connections should reserve the resources needed and routers should maintain reservation parameters on a per-flow basis. Although [2] [3] proposed to alleviate some of difficulties in IntServ, implementation of IntServ is still difficult for a complex and large network such as the Internet. To reduce the complexity of QoS management, DiffServ was introduced to provide QoS within a domain using aggregation of flow and per-class service.

In reality, QoS parameters are not easy to control because the Internet is a shared network. QoS variation depends on load of the network related to traffic from all hosts connected to the Internet. The more stringent the guarantee requirements are needed, the more complex the implementation becomes. Currently, there are two approaches proposed to realize the DiffServ: absolute DiffServ [6] [7] [8] and relative DiffServ [9] [10] [11]. Absolute DiffServ approach attempts to provide services with absolute profiles (e.g. a certain amount of bandwidth) but without per-flow parameter maintenance in the network core. The user receives an absolute service profile similar to that of leased line as long as user's traffic is under the given profile limitation [6]. Relative DiffServ seeks to provide per-hop relative services to the traffic classes it supports. It cannot guarantee the amount of network resources provided to each class but intends to locally ensure that the QoS of higher priority classes will be better than that of the lower ones. Because of implementation difficulty of absolute DiffServ, the most recent research usually focuses on relative DiffServ. Currently, there are two important approaches to guarantee the QoS relatively to the higher and lower priority classes. They are price relative [9] and proportional DiffServ schemes using Waiting Time Priority

(WTP) queue [10] and Weighted Fair Queue (WFQ) [12]. Optimized rule-based controller for proportional DiffServ with bias features is the main focus of this paper.

Proportional DiffServ (Prop-DiffServ) has been recently investigated extensively [12] [13]. It attempts to keep QoS ratio of each class at a value configured by an administrator. Some contemporary studies adapt the strategies of reducing the management complexity by pre-defining all class parameters. Therefore, call-admission-control (CAC) is unnecessary and users cannot change their traffic preferences. A drawback of this approach is that QoS of a particular class depends on QoS of other classes. If QoS of a class drops, Prop-DiffServ algorithm will drop QoS of others to maintain the QoS ratio between them. This situation is unsuitable for critical application such as tele-medicine because other traffics can disturb its QoS and lead to severe hazard. Hence, we felt the need for defining the QoS ratio so that high priority traffics are more tolerable to traffic fluctuations of lower priority classes.

This paper presents a way of using an optimized fuzzy controller to adjust the delay proportion in Prop-DiffServ according to the particular class traffic condition to provide fairness of delay distribution among the other classes. We use Particle-Swarm Optimization (PSO) algorithm [14] with some useful modifications to optimize the fuzzy controller. Simulation results show that the proposed solution is effective. Organization of the reminder of the paper is as follows: Section II gives an overview of the background theories. Section III presents the statement of the problems then followed by proposed solutions in the Section IV. Section V shows the implementation of fuzzy controller into the proposed solution and optimization process of the fuzzy controller membership functions. Section VI and VII describe the simulation model and provide the discussion of results. The conclusion of this work is given in Section VIII.

II. BACKGROUND

A. Proportional Differentiated Services

In Prop-DiffServ [10], services for each class should be proportional to the ratio set in the service contract to the customer. The contract can be pre-defined or negotiated. Suppose that a domain provides N service classes and $\bar{q}_i\,(t, t+\tau)$ is the average QoS value for class i in the time interval $(t, t+\tau)$, where τ is fixed time interval between each QoS measurement and should be greater than 0. In Prop-DiffServ, the following equation should be satisfied for any pair of classes.

$$\frac{\bar{q}_i\,(t, t+\tau)}{\bar{q}_j\,(t, t+\tau)} = \frac{c_i}{c_j} \quad ;\; i \neq j \tag{1}$$

Where c_i and c_j are the service differentiation parameters of class i and j respectively. They are fixed by the contract.

In this paper, the QoS parameter we focus on is queueing delay only. Therefore, (1) can be rewritten as:

$$\frac{\bar{d}_i\,(t, t+\tau)}{\bar{d}_j\,(t, t+\tau)} = \frac{\delta_i}{\delta_j} \quad ;\; i \neq j \tag{2}$$

Where $\bar{d}_i\,(t, t+\tau)$ is the average queueing delay for class i in the time interval $(t, t+\tau)$ and δ_i is fixed delay differentiation parameter (DDP) [10] of class i

B. Fuzzy Controller

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 [15]. 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]. We call $\mu_A(x)$ the grade of membership. Hence, the fuzzy set A on a universe of discourse X is defined as follows:

$$A = \{(x, \mu_A(x)) \,|x \in X\} \tag{3}$$

A fuzzy controller is a nonlinear mapping system performed by using fuzzification, fuzzy inference, and defuzzification components as shown in Fig. 1.

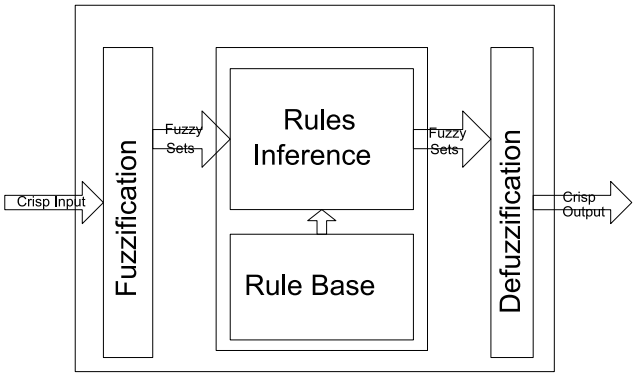


Fig. 1. Schematic of a fuzzy controller

Fuzzification is the process of calculating suitable sets of degree of membership, called "fuzzy set", for crisp inputs. The next step is fuzzy inference machine. It evaluates output fuzzy sets from the input sets using predefined knowledge base called "fuzzy rules". A simple fuzzy rule can be written as:

$$if \quad x \ is \ A \quad then \quad y \ is \ B \tag{4}$$

Where A and B are linguistic values defined by fuzzy sets on the universes of discourse X and Y respectively. The if-part, "$x \ is \ A$", is called the antecedent and then-part, "$y \ is \ B$", is called the consequence. If a rule antecedent contains more than one variable, a degree of membership is calculated for each variable and these degrees are combined using t-norm to form the degree of activation of the rule. Consider a rule r, which has k variables in its antecedent and uses minimum t-norm, its degree of activation can be evaluated using (5).

$$\mu(r) = \min_{l}^{k} (\mu_l(x_l)) \tag{5}$$

Where $\mu(r)$ is degree of activation of rule r and $\mu_l(x_l)$ is degree of membership of input x_l. The degree of activation is inferred as the degree of membership of output variable upon its fuzzy set, which defined in corresponding consequence. The output of inferring m rules upon the i^{th} output variable is the aggregation of the individual rule output upon that output variable.

Finally, the implied output sets are combined to formulate a crisp output. The center of gravity (CoG) technique, which computes the weighted-average of the center of gravity of each membership function is used primarily [16]. The CoG of the system with R rules can be calculated using (6). In this paper, we use CoG as defuzzification method in our system.

$$y(x) = \frac{\sum_{i=1}^{R} b_i \mu(r_i)}{\sum_{i=1}^{R} \mu(r_i)} \tag{6}$$

Where b_i is the center of the membership function recommended by the consequence of rule i.

The membership functions of the fuzzy controller are initialised by the user based on a priori knowledge. When such knowledge is not available, it is essential to use an optimisation strategy to modify or tune them. Particle-Swarm Optimization is a recently proposed optimisation method that can be used for this purpose.

C. Particle-Swarm Optimization

Particle-Swarm Optimization (PSO) is a population based optimization algorithm introduced by Kennedy and Eberhart [14]. Unlike other population based optimization algorithms such as genetic algorithms, PSO does not directly transfer any information among the individuals in the population. It is based on the simulation of animal social behavior in finding food sources. Therefore, in PSO, each particle (i.e., individual) moves within the solution space to find the global best solution. Particle's moving vector is dynamically adjusted according to its own and other's experiences.

At a discrete time t, the position and the moving vector of particle j in the d-dimensional search space can be represented as $\vec{S}_{jt} = (s_{j1t}, s_{j2t}, s_{j3t}, \ldots, s_{jdt})$ and $\vec{V}_{jt} = (v_{j1t}, v_{j2t}, v_{j3t}, \ldots, v_{jdt})$ respectively. The ever best position of particle j, which has the best fitness value obtained by user-defined fitness function, at time t is represented as $\vec{P}_{jt} = (p_{j1t}, p_{j2t}, p_{j3t}, \ldots, p_{jdt})$. Let $\vec{P}_{gt}$ denotes the global best position at time t, which is, actually, the $\vec{P}_{jt}$ with the best fitness value for all particles. The new moving vector of each particle for each move is calculated according to the following equation.

$$\begin{aligned} \vec{V}_{j(t+1)} &= \vec{V}_{jt} + c_1 \cdot \vec{R}_1 \cdot \left(\vec{P}_{jt} - \vec{S}_{jt}\right) \\ &\quad + c_2 \cdot \vec{R}_2 \cdot \left(\vec{P}_{gt} - \vec{S}_{jt}\right) \end{aligned} \tag{7}$$

Where c_1 and c_2 are scalar constants known as acceleration coefficients. $\vec{R}_1$ and $\vec{R}_2$ are two sparely generated uniformly distributed random vectors each of which is in the range of [0, 1].

The first part in (7) is called "inertia" part, which is the current velocity of the particle providing momentum for the particle to move along the same speed. The second part represents the thinking of an individual particle and, therefore, called "cognitive" part. The cognitive part accelerates the particle towards its own best position. The last part known as the "social" part encourages the particle to move towards the best position of all particles in order to converge to the global optimum value. To prevent particles from moving too fast and fly over optimum point, the parameter $\vec{V}_m$ is defined to be the upper limit of velocity. Whenever a movement vector element exceeds the correspondent element of $\vec{V}_m$, that element will be set to its upper limit.

The position of each particle is updated at each time step (i.e., iteration) using the following equation.

$$\vec{S}_{j(t+1)} = \vec{S}_{jt} + \vec{V}_{jt} \tag{8}$$

We should also limit the position of each particle to prevent it from moving out of the defined solution space.

III. Statement of the Problems

A. Unfairness in Delay Proportional Differentiated Service

From Fig. 2, let $o(t)$ be the queue service rate (or outgoing rate), $r(t)$ be the queue incoming rate, $q(t)$ be the backlog at any time t, and the packet L be the last packet in the buffer at time t. The backlog can be derived from incoming and outgoing rate with (9).

$$q(t) = \int_0^t (r(x) - o(x))\, dx \tag{9}$$

Considering that queue is infinite and no data loss occurred in the queue system, to clear up the last packet in the backlog at time t, it requires time d_L to be serviced. The time d_L is called queue delay for $q(t)$ and can be defined as:

$$q(t) = \int_t^{t+d_L} o(x)dx \tag{10}$$

By substituting (9) in (10), it can be concluded that:

$$d_L = \left\{ \zeta \;\middle|\; \int_0^t r(x)dx = \int_0^{t+\zeta} o(x)dx \right\} \tag{11}$$

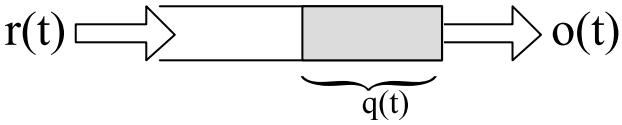


Fig. 2. Diagram of a packet queue.

Generally, we can determine the service rate of a queue but cannot control queue incoming rate, it can be concluded that, $d_L \propto \int_0^t r(x)dx$. However, the delay d_L is known at time $t+\zeta$, which is not yet reached. Therefore, computation according to d_L at time t needs to predict $o(t, t+\zeta)$. A simple method is to assume that $o(t)$ is maintained from t to $t+\zeta$. In (2), average queueing delay of classes i and j are dependent on each other because the value of δ_i/δ_j are fixed. This ratio causes queue delay of any particular class dependent on incoming rate of all classes regardless of any priority. Since the Prop-DiffServ was designed to support only relative differentiated services, class priority can be given by only the definition of "better" or "worse" service relative to each others. This condition sometimes causes unfair resource sharing for high priority class when any lower priority class suffers degradation of its queue delay. Fig. 3 shows result from a simulation scenario where all traffic share equal traffic load (class priority decrease with the increase of class number) and Fig. 4 and 5 show results from the same scenario except we pump a burst from 100 to 150 seconds into class 1 and 2 respectively. It can be seen that both Fig. 4 and 5 provide the very

same result regardless of to which class the burst is belonged. The figures also show that Class 0 which is supposed to have higher priority than class 1 and 2 also suffers from the burst applied to class 1 and 2. We will discuss our solution to solve this unfair condition in Section IV.

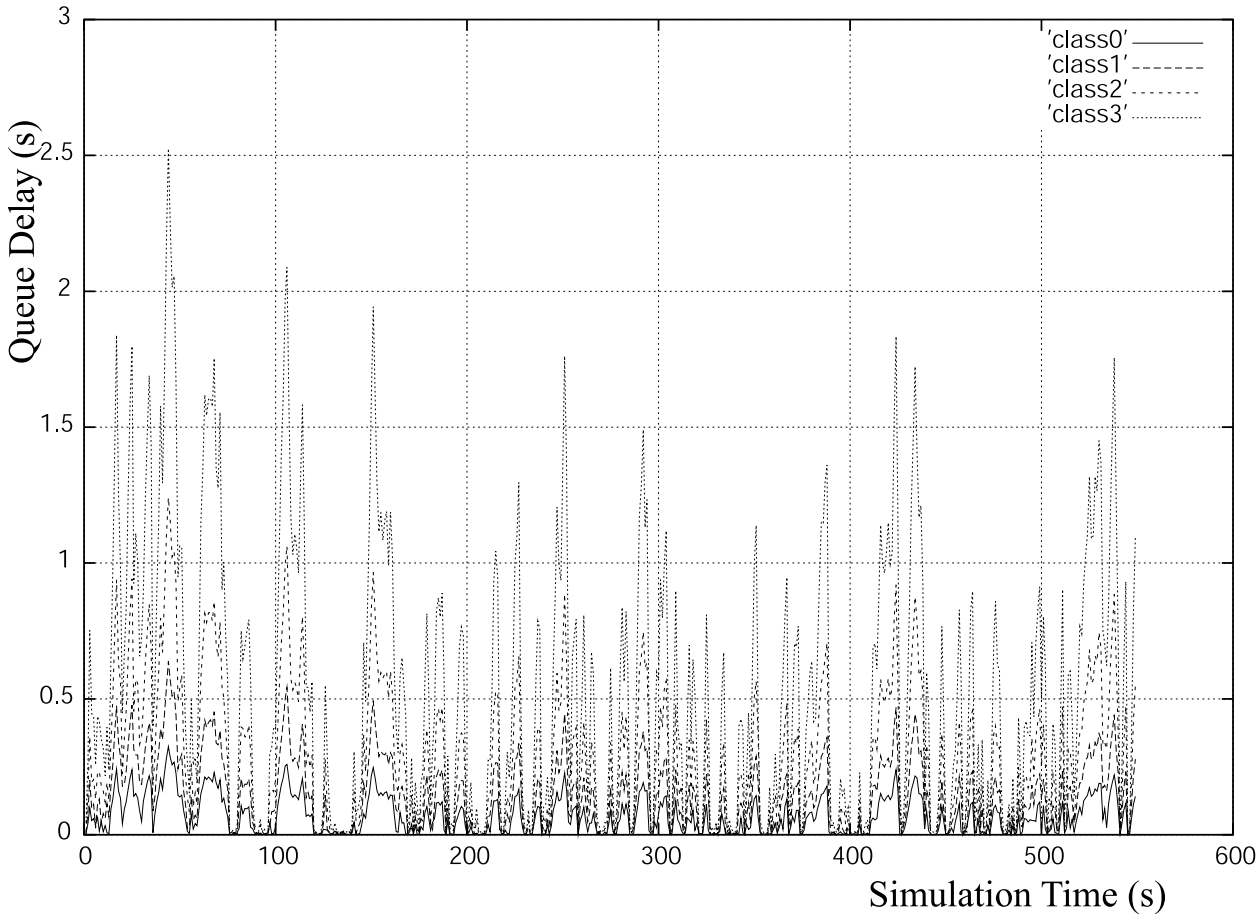


Fig. 3. Packet Delay without burst

Since proportional DiffServ was first proposed in [10], there are some extensions reported in [12] and [13]. None of those studies consider solving this problem. In [13], authors attempt to improve quality of maintaining the delay proportion. In [12], the original proportional DiffServ is implemented into WFQ scheduler without any improvement on the differentiation between classes. An interesting delay differentiated services is recently proposed in [17]. This Scheme can manage both absolute and relative differentiation. However, it is a completely new scheme compared to WTP used in [10] and [13] and WFQ used in [12]. This new scheme needs replacement of the original algorithms implemented in widely used routers.

B. Possible Improvements to Particle Swarm Optimization

Since the PSO was introduced, many improvement have been proposed [18]–[21]. One of the most important attempts is to include a time-varying parameter called inertia weight (ω) into each part of moving vector computation

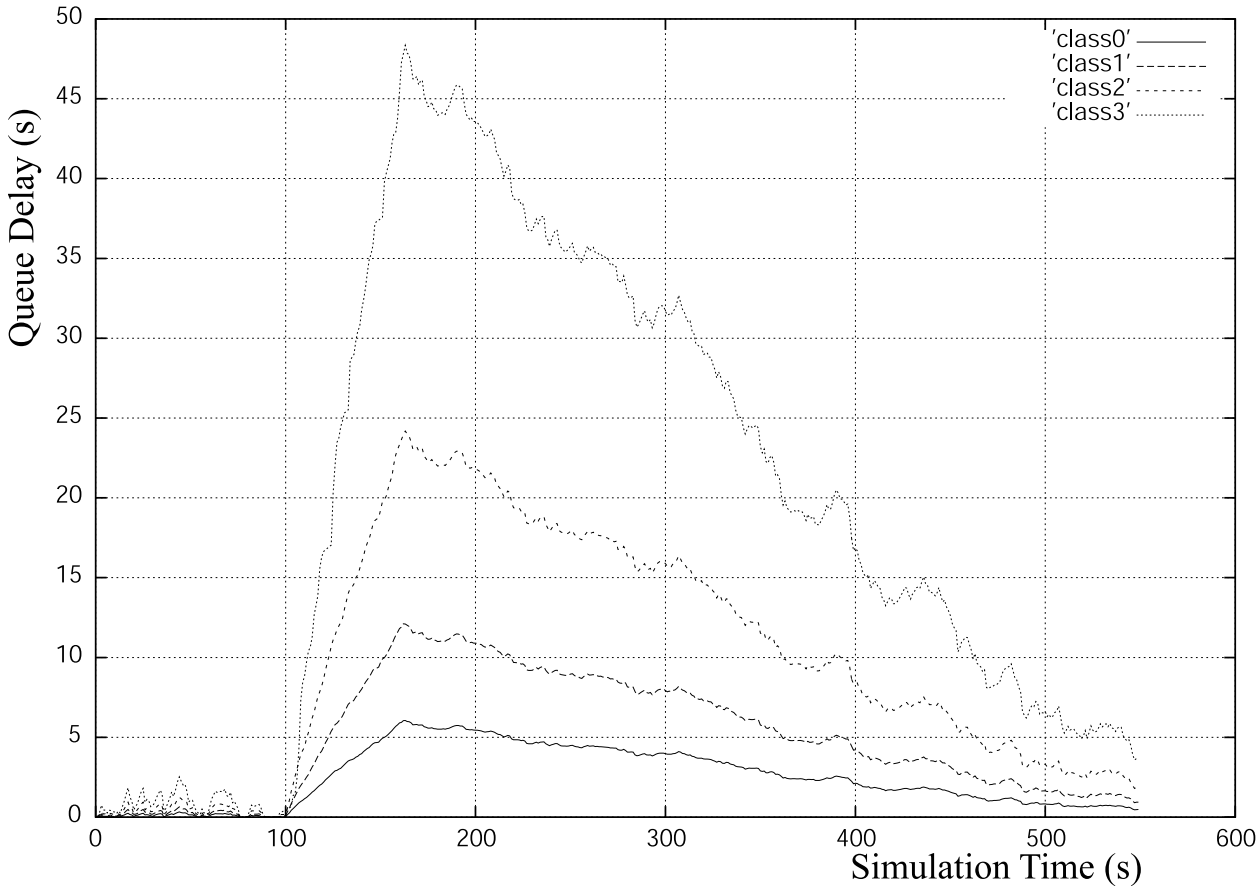


Fig. 4. Packet Delay with Burst Coming from Class 1

[20], [21]. The purpose of this inertia weight is to reduce movement of a particle based on its inertia and converge to optimum point at the end of optimization iteration. A recent improvement in [22] attempts to increase the opportunity of each particle to search around its own best solution instead of getting stuck to social optimum experienced at the beginning of optimization process. Two time-varying parameters (c_1 and c_2) are introduced in [22] and operate together with the popular inertia weight. This modification can be represented as follows.

$$\begin{aligned} \vec{V}_{j(t+1)} &= \omega \cdot \vec{V}_{jt} + c_1 \cdot \vec{R}_1 \cdot \left(\vec{P}_{jt} - \vec{S}_{jt}\right) \\ &\quad + c_2 \cdot \vec{R}_2 \cdot \left(\vec{P}_{gt} - \vec{S}_{jt}\right) \end{aligned} \tag{12}$$

Where ω, c_1, and c_2 are defined as:

$$\omega = (\omega_{\max} - \omega_{\min}) \frac{(i_{max} - i_t)}{i_{max}} + \omega_{\min} \tag{13}$$

$$c_1 = (c_{1\ min} - c_{1\ max}) \left(\frac{i_t}{i_{max}}\right) + c_{1\ max} \tag{14}$$

$$c_2 = (c_{2\ max} - c_{2\ min}) \left(\frac{i_t}{i_{max}}\right) + c_{2\ min} \tag{15}$$

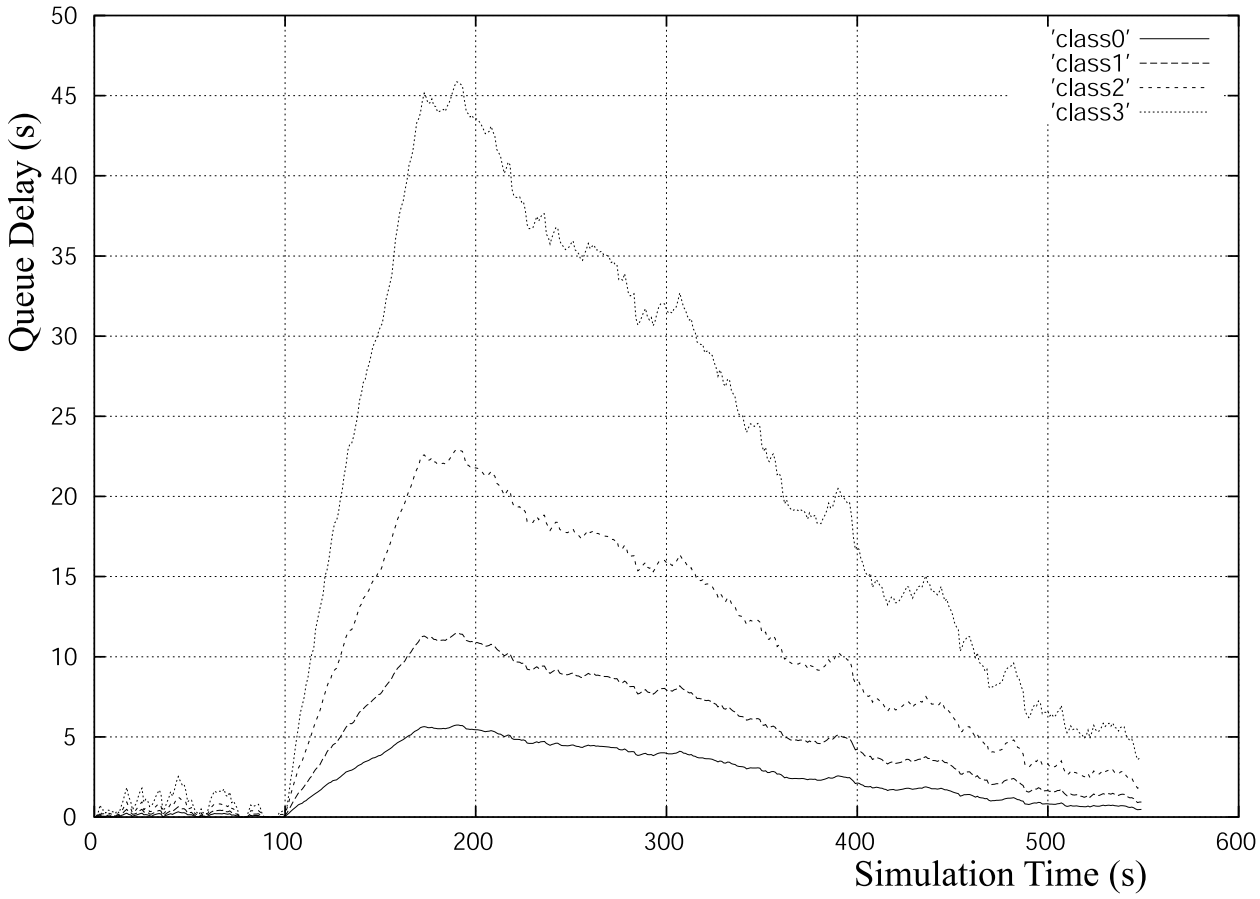


Fig. 5. Packet Delay with Burst Coming from Class 2

The parameters ω_{max}, $c_{1\ max}$, $c_{2\ max}$, ω_{min}, $c_{1\ min}$, and $c_{2\ min}$ are the upper and lower limits of variation, i_t is the current iteration number, and i_{max} is the maximum number of allowable iterations. The recommenced values for variation limits stated in [20] [21] [22] are $\omega_{max} = 0.9$, $c_{1\ max} = 2.5$, $c_{2\ max} = 2.5$, $\omega_{min} = 0.4$, $c_{1\ min} = 0.5$, and $c_{2\ min} = 0.5$

The main idea proposed in [22] is about giving more weight to the cognitive part at the beginning of optimization process to encourage a particle to search nearby its local-best location. Then, the weight of the cognitive-part is reduced and more priority is given to the social part to finalize the global optimum result. However, particle movement is tightly bound to the experienced optima because the movement vector is calculated from local and global optima the system has already experienced. The major drawback is that the system has high chance of divergence if the true global optimum is far away from any optimum experienced by a particle. This drawback has less influence if the particles are uniformly distributed throughout the problem space. However, there is no guarantee on uniform distribution even if we use uniformly distributed random number in initial particle placement process. Fig. 6 shows a worst-case scenario where particles move away from global optimum. In Fig. 6, P_i is best position of the particle i, S_i is the current position of the particle i, and P_g is the current global best position of all

particles. This example consists of 4 particles with 5 optima. In this case, particles may swarm around their local optima and cannot reach the global optimum value. The worst case occurs usually when the following conditions are met.

- Global optimum is separated from most of local optima by a large margin.
- Particles are not fully uniformly distributed at the initialisation.
- the social optimum of the swarm is far away from the global optimum to be reached.

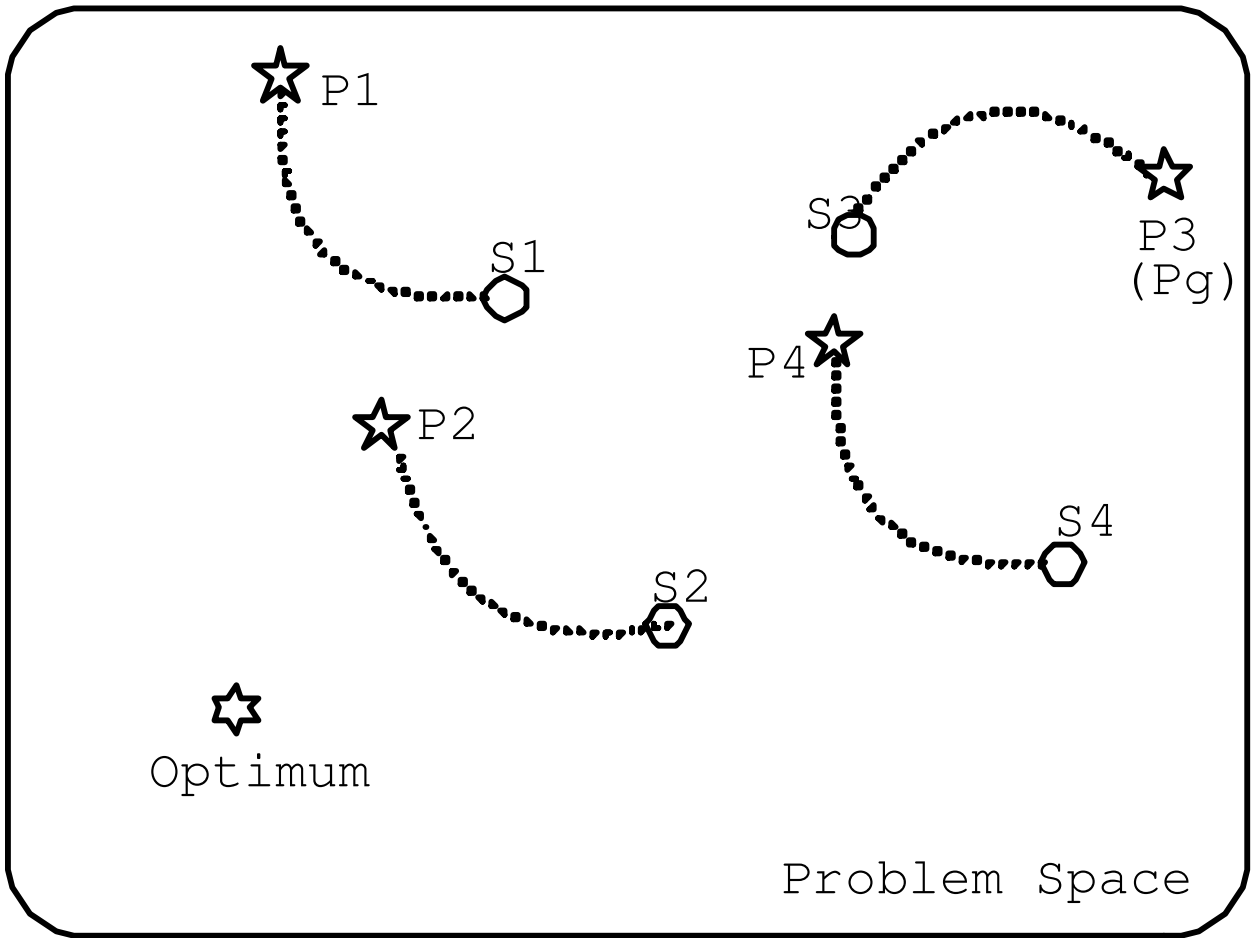


Fig. 6. Worst-case scenario with non-converging PSO

IV. Proposed Algorithms

A. Variable Delay-Proportion

In this paper, we propose a scheme to reduce the effect of unfairness caused by bursty data coming into lower priority classes. The constraint of the delay ratio between class i and j where class i has higher priority with respect

to class j is:

$$\frac{\bar{d}_i\,(t,t+\tau)}{\bar{d}_j\,(t,t+\tau)} = \frac{\delta_i}{\delta_j \cdot f\left(\frac{u_j}{u_i}\right)} \quad ;\ i \neq j \tag{16}$$

Where u_i and u_j are resource utilization factor of class i and j respectively and $f(u_j/u_i)$ is a bias function which gives 1.0 when $u_j \leq u_i$ or otherwise a positive bias value which is greater than 1. Eq. (16) shows that the delay ratio between class i and j will be equal to δ_i/δ_j when $u_j \leq u_i$ which means that the system operates as normal proportional DiffServ. However, if there exists traffic class k which has lower priority than class j and utilize the system at most at the same level as class j, which means that $f(\frac{u_k}{u_j})$ is 1.0, we should maintain $\frac{\bar{d}_j(t,t+\tau)}{\bar{d}_k(t,t+\tau)}$ at the ratio δ_j/δ_k regardless of any bias applying to class j due to difference of utilization between class i and j. It means that the bias of class j should propagate its effect upon class k and all lower priority classes. For the highest priority class, class 0, there is no propagated bias from others and its traffic fluctuation can affect all other classes; thus, there is no bias function for this class.

From the definition above, the combined bias value of a class has two parts. The first is the bias according to resource utilization of that class. The other part is the propagated bias from higher priority classes. The combined bias should also be propagated to all lower priority classes. Let β_i be the first part of combined bias and b_i be the propagated bias of class i. The general equation derived from (16) can be written as:

$$\frac{\bar{d}_{i+1}\,(t,t+\tau)}{\bar{d}_i\,(t,t+\tau)} = \begin{cases} \frac{\delta_{i+1}\cdot\beta_{i+1}\cdot b_{i+1}}{\delta_i\cdot\beta_i\cdot b_i} & ;\quad i>0 \\ \\ \frac{\delta_1\cdot\beta_1}{\delta_0} & ;\quad i=0 \end{cases} \tag{17}$$

From the definition of b_i, it can be concluded that:

$$b_i = \beta_{i-1}\cdot b_{i-1} \tag{18}$$

Eq. (18) is a recursive equation that can be rearranged as:

$$b_i = \prod_{k=1}^{i-1} \beta_k \tag{19}$$

In (19), the bias parameter needed to evaluate is β of every class except class 0. To find the value of β, we have to supervise resource utilization of each class. Since traffic in the Internet is very dynamic, if the system changes β_i according to the current condition on a per-packet basis, this algorithm will cause a lot of toggling or oscillation due to the unsteadiness of delay proportion adjustment. Moreover, operating the scheme with every packet causes a lot of processing overhead.

Generally, the traffic causing excessive delay upon high priority classes is resulted from bursty data coming into low priority classes. We therefore concentrate on this major part of bursty data coming to lower priority classes by applying predictive technique to detect incoming bursty data. Moreover, the principle concept of proportional DiffServ is based on periodic measurement with interval τ [10]. Therefore, we use the same interval τ to periodically update b_i. This means that adjustment at time t is based on average values of parameters from time $t-\tau$ to t. Another advantage of using averages of parameters is that the system will not be highly sensitive to traffic variation.

Since the adjustment uses historical measurement, the system cannot instantly detect bursty traffic. The delay of high priority class may increase according to the lower priority burst. Instantly after detecting the incoming burst in the next measurement, the system adjusts the particular β_i and then the delay of high priority class decreases to normal again. Therefore, selecting the value of τ affects the amount of delay increased and the time needed to converge to a normal value.

To calculate β_i, we should specify parameters related to "utilization" and overall delay. The chosen parameters are current backlog and incoming rate. Because the mathematical representation of network-traffic characteristic is complex, we cannot easily calculate the appropriate β_i. To make the system less complex and more efficient, we use fuzzy logic controller to obtain the β_i. However, the design process of fuzzy-controller parameters is based on empirical trial-and-error. We improve the design process by instinctively specifying initial parameters and, then, by applying modified PSO to tune up the parameters to their optimum values. We choose PSO rather than other optimization algorithms such as genetic algorithm, which is widely used as in [23] [24] [25] [26], because PSO is simpler and requires less computational resources and, hence, faster. In addition to the enhancement proposed to the PSO, we also included possible solutions (parameter values) as particles to the randomly initialized first set of particles. We believe that high computational efficiency was achieved due to these measurement.

B. Modification of Particle-Swarm Optimization

To overcome the drawback of PSO that binds particles with their history, we introduce a new part called "instinct" part into movement vector calculation. This part is computed regardless from any particles' history. This part acts as local instinct of a real animal which make its movement unpredictable. We can use this part to reduce non-uniformness of initial particle placement as it operates as cascade random process. The final movement vector calculation can be written as:

$$\begin{aligned} \vec{V}_{j(t+1)} &= \omega \cdot \vec{V}_{jt} + c_0 \cdot \vec{R}_0 \cdot \vec{V}_m \\ &\quad + c_1 \cdot \vec{R}_1 \cdot \left(\vec{P}_{jt} - \vec{S}_{jt} \right) \\ &\quad + c_2 \cdot \vec{R}_2 \cdot \left(\vec{P}_{gt} - \vec{S}_{jt} \right) \end{aligned} \tag{20}$$

The coefficient c_0 can be computed by (21). The second term $c_0 \cdot \vec{R}_0 \cdot \vec{V}_m$ is the instinct component. The parameter $\vec{R}_0$ is a random vector generated with the same method as used for $\vec{R}_1$ and $\vec{R}_2$ but within the range [-1, 1]. It allows the particle to move in reverse direction, in contrast to the range [0, 1] used for $\vec{R}_1$ and $\vec{R}_2$. However, the instinct part should have less effect in later optimization iteration when the particles should conclude to the global optimum point. On the other hand, this part should be a major part in movement vector in early iteration because the optimum point at very early optimization iteration is likely to be local optimum rather than true global optimum. Hence, our modification utilize some iteration at the beginning of optimization iteration as instinctively searching period. The coefficient σ, c_1, and c_2 are changed and presented as follows:

$$c_0 = \begin{cases} \omega & ; i_t < w_1 \\ 0 & ; otherwise \end{cases} \tag{21}$$

$$c_1 = \begin{cases} c_{min} & ; i_t < w_1 \\ c_{max} & ; w_1 \le i_t < w_2 \\ \frac{c_{max}-c_{min}}{1-w_2}\left(\frac{i_t-w_2}{i_{max}}\right) + c_{max} & ; i_t \ge w_2 \end{cases} \quad (22)$$

$$c_2 = \begin{cases} c_{min} & ; i_t < w_2 \\ \frac{c_{min}-c_{max}}{1-w_2}\left(\frac{i_t-w_2}{i_{max}}\right) + c_{\min} & ; otherwise \end{cases} \quad (23)$$

Eq. (22) and (23) shows that optimization process is divided into 3 stages according to w_1 and w_2, which are predefined and fixed.

1) When $i_t < w_1$: In this stage, both c_1 and c_2 are set to c_{min}. It means that cognitive and social parts have least effect to vector calculation. Though the initial position of each particle is randomized, we cannot guarantee that particles are perfectly distributed in solution space. This stage is to improve particle distribution by activating the instinct part. The other benefit of this part is make the particle move with its own trial-and-error instinct rather than rush to any optimum point founded. Generally, w_1 is not greater than 10% of i_{max}.

2) When $w_1 \le i_t < w_2$: In this stage, c_1 is set to its maximum value while c_2 is at its minimum and ω continues to decrease from the first stage. Therefore, the highest dominant part is cognitive part. The purpose of this stage is to encourage the particle to look around by itself and find its own optimum location mainly according to its moving experience.

3) When $i_t \ge w_2$: This stage is the final stage. In this stage, ω is still decreasing to its minimum value, c_1 starts to decrease, and c_2 starts to increase. It means that we reduce particle movement according to individual experience and encourage particles to move to global optimum solution.

We have not attempted to specify the best value of w_1 and w_2. However, during our optimization process, we specified them as 5% and 10% of i_{max} respectively and the optimization result is excellent.

The other modification applied to the original PSO is the replacement mechanism. Whenever a particle moves out of the solution space in any dimension, the original PSO will set the position in that dimension to the boundary value. In our PSO, instead of using the original method, we replace the position in that dimension using (24), which means that the particle will be randomly placed into the search space along the concerned dimension.

$$s_d = (s_{d\,max} - s_{d\,min})\, rand + s_{d\,min} \quad (24)$$

Where d is the concerned dimension, $rand$ is a random number, $s_{d\,max}$ and $s_{d\,min}$ is the upper and lower boundary of the space in the d^{th} dimension.

The performance improvement of the new PSO was observed by four widely used benchmarks. They are shown in Table I. The first two functions are uni-modal functions and the other two are multi-modal functions. They are designed with a considerable amount of local optima. The simulations were carried out with the same configurations as in [22] in which the parameters are listed in Table II and III. Each benchmark is simulated with 10, 20, and 30 problem dimension (n) with population size set to 40 for all simulation.

We performed the simulation for 50 trials on each function. The output shown in Table IV is the average number of iterations required to converge to the error specified in Table III. In Table IV, the comparison data are obtain

TABLE I

BENCHMARKS FOR SIMULATIONS

Function	Mathematical Representation
Sphere	$f_1(x) = \sum_{i=1}^{n} x_i^2$
Rosenbrock	$f_2(x) = \sum_{i=1}^{n-1} \left[100\left(x_{i+1} - x_i^2\right)^2 + (x_i - 1)^2\right]$
Rastrigrin	$f_3(x) = \sum_{i=1}^{n} \left[x_i^2 - 10\cos(2\pi x_i) + 10\right]$
Grewank	$f_4(x) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$

TABLE II

PARAMETERS FOR BENCHMARKS

Function	Init Range	$\vec{S}_{min}$	$\vec{S}_{max}$	$\vec{V}_m$
$f_1(x)$	$(50, 100)^n$	$[-100]^n$	$[100]^n$	$[100]^n$
$f_2(x)$	$(15, 30)^n$	$[-100]^n$	$[100]^n$	$[100]^n$
$f_3(x)$	$(2.56, 5.12)^n$	$[-10]^n$	$[10]^n$	$[10]^n$
$f_4(x)$	$(300, 600)^n$	$[-600]^n$	$[600]^n$	$[600]^n$

directly from [22]. We can see that the new modification mostly perform better than the algorithm in [22]. The only case where our new modification is not competitive is the case of Rastrigrin benchmark with 10 problem dimension. It is because of the very narrow initialisation range of particles making the two PSO versions compared to converge very fast but our new modification need some iterations at the beginning for instinct part before converge to the optimum value.

V. IMPLEMENTATION

A. Fuzzy Controller Implementation

As the operation of our fuzzy controller does not affect the normal queue-scheduling algorithm, we implement the controller as an independent agent within the normal Prop-DiffServ router. It measures queue statuses of each class and tunes the queue scheduler, so that queue priorities and service rates are changed. The diagram of our controller is shown in Fig. 7.

We measure two parameters, traffic-incoming rates, denoted by r_i, and queue-occupation sizes (backlog), denoted by q_i, from the queues and process them by a preprocessor. The results from the preprocessor, which are σ_i and θ_i where i is queue class number, are used as the first two inputs of the fuzzy controller. The last parameter used

TABLE III

ERROR LIMITS FOR CONVERGENCE

Function	Error Limit
$f_1(x)$	0.001
$f_2(x)$	100
$f_3(x)$	100
$f_4(x)$	0.1

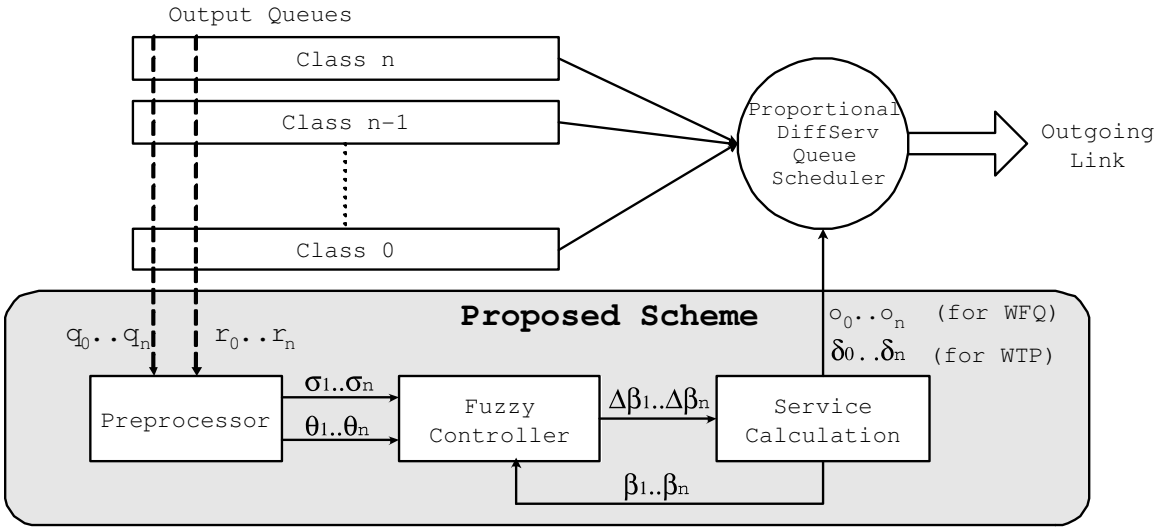


Fig. 7. Diagram of the proposed scheme

as an input of fuzzy controller is the current β_i. The output of the fuzzy controller is the change of β_i denoted by $\Delta\beta_i$ is then fed to Service Calculation, which calculate the new β_i and use it to calculate the new scheduling parameters for each type of queue scheduler.

The idea of calculating σ_i and θ_i as the inputs is to maintain the fairness of queue usage. If incoming traffic rate of a class increases much over its higher priority class, it will affect the delay of the higher priority class. Hence, we should increase β of that class. However, the input packets of the previous burst may take time to be served. Although the incoming rate of the concerned class becomes normal, the residual packets still cause the high delay effect upon its higher priority classes. Therefore, we need queue occupation to be an input to our fuzzy controller. However, because a lower priority queue gets a lower serving rate than its higher priority ones, even with the same incoming rate, its queue occupation is always higher than that of the higher priority queues. Therefore, we allow it to occupy more queue space than the next higher priority class. In [12], authors proposed a relationship equation between queue service rates, backlogs, and delay-differentiation-parameter (DDP) ratio [10]. The equation is shown

TABLE IV

COMPARISON OF PSO PERFORMANCE BETWEEN [22] AND MODIFICATION.

Function	n	Average number of iterations required for convergence: Original PSO	With modification in [22]	Our Modification	Improvement with respect to [22]
$f_1(x)$	10	1331	**639**	648	-1.4%
	20	1815	821	**808**	1.6%
	30	2156	951	**908**	4.7%
$f_2(x)$	10	1872	748	**680**	10.0%
	20	3081	1225	**991**	23.6%
	30	3718	1516	**1241**	22.2%
$f_3(x)$	10	54	**24**	280	-1066.7%
	20	1834	868	**720**	20.6%
	30	2525	1173	**931**	26.0%
$f_4(x)$	10	2874	1422	**944**	50.6%
	20	2597	1261	**1158**	8.9%
	30	2980	1262	**1254**	6.4%

as (25). Therefore, we adapt (25) to calculate normalized backlog ratio between classes, defined as θ_i. The equation for σ_i and θ_i are shown as (26) and (27).

$$\frac{o_i(t)}{o_j(t)} = \frac{\frac{1}{\delta_i}[q_i(t) + \frac{1}{2}r_i(t)\tau]}{\frac{1}{\delta_j}[q_j(t) + \frac{1}{2}r_j(t)\tau]} \tag{25}$$

$$\sigma_i = \frac{r_i}{r_{i-1}} \quad ; i > 0 \tag{26}$$

$$\theta_i = \frac{\delta_{i-1}o_{i-1}(t)[q_i(t) + \frac{1}{2}r_i(t)\tau]}{\delta_i o_i(t)[q_{i-1}(t) + \frac{1}{2}r_{i-1}(t)\tau]} \quad ; i > 0 \tag{27}$$

In ideal case, both σ_i and θ_i should be equal to 1. Any deviation of either of them means that the resource utilization among traffic classes is unfair. Hence, β_i should be adjusted.

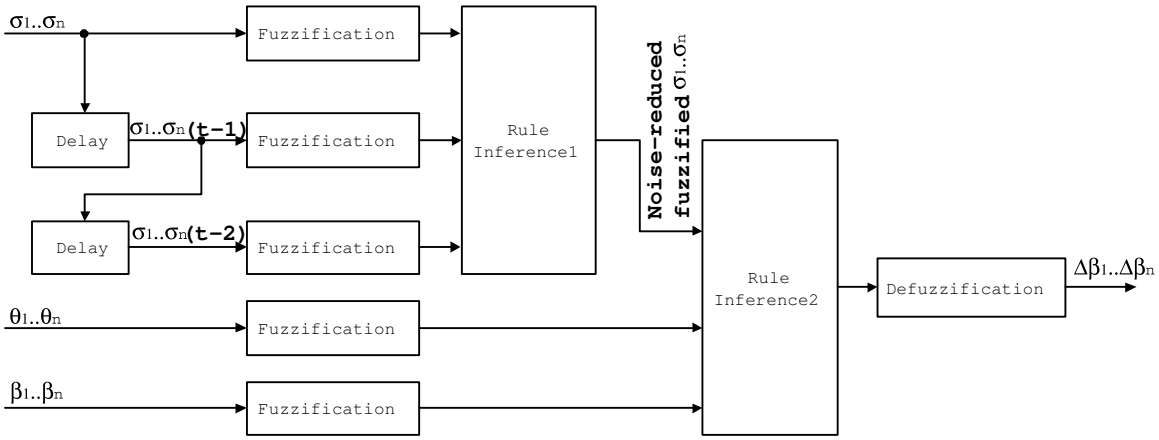


Fig. 8. Schematic of proposed fuzzy controller

After fuzzified, the inputs are used to infer the output using fuzzy rules shown in appendix. The output will be defuzzified by center-of-gravity method [16]. The result, $\Delta\beta_i$, is used to update β_i using (28).

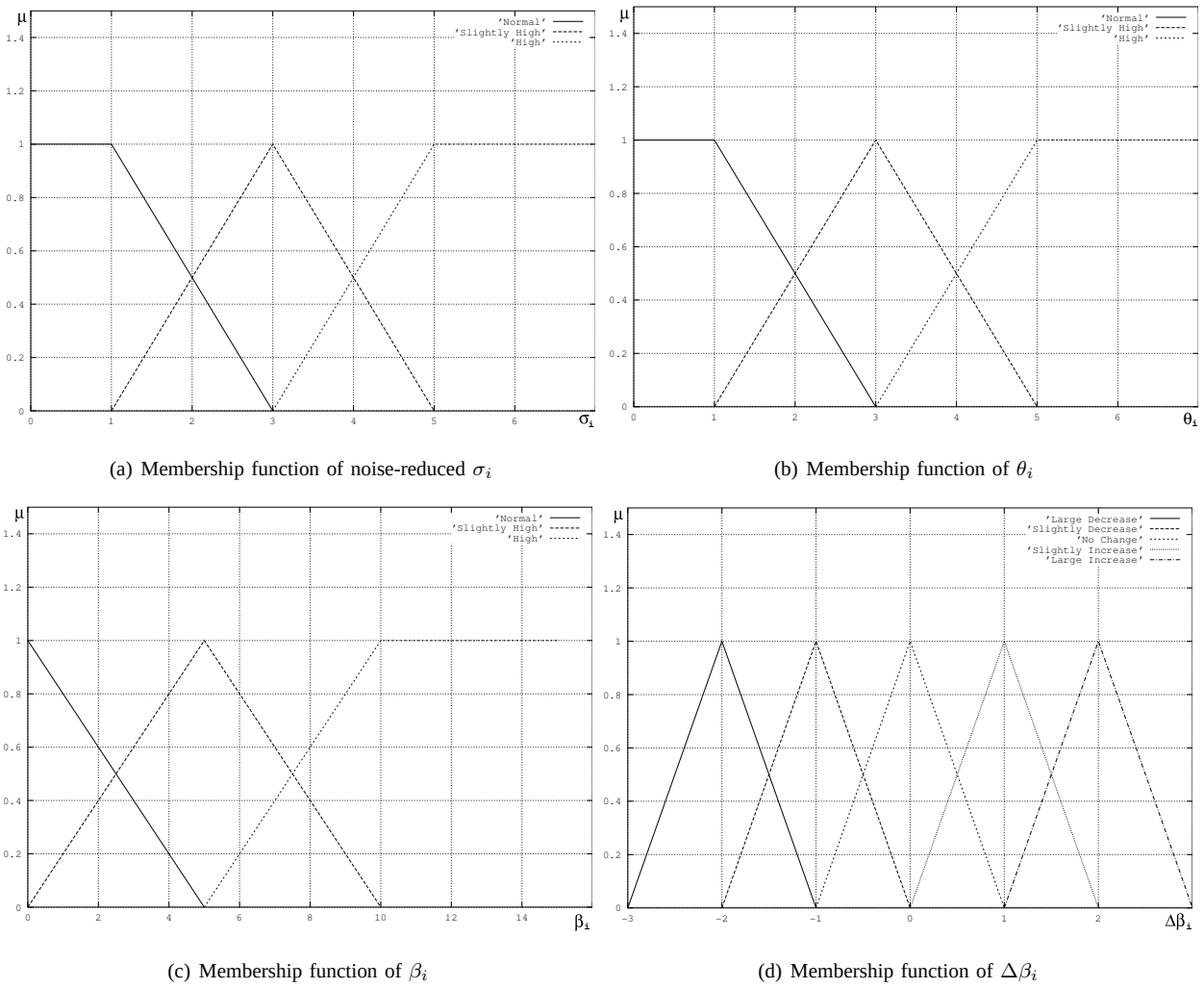
$$\beta_{i(next)} = \beta_{i(current)} + \Delta\beta_i \tag{28}$$

From the fuzzy rules in the Appendix II, the key parameter to adjust β_i is σ_i. Fluctuation of traffic rate (i.e., r_i) can cause instability of the whole controller. Therefore, we constructed our fuzzy controller using two stage hierarchical fuzzy controller. The first fuzzy stage, Rule Inference 1, is to reduce oscillation of σ_i. The other stage is our main controller stage, Rule Inference 2, which calculates the $\Delta\beta_i$ using the rules in the Appendix II. The schematic of our fuzzy controller is shown in Fig. 8. The rules used in Rule Inference 1 are listed in Appendix I.

The initial fuzzy membership functions for β_i, $\Delta\beta_i$, θ_i, and noised-reduced σ_i are shown in Fig. 9. The membership function of noised-reduced σ_i is also used as output membership function of Rule Inference 1; hence, the two rule inference engines can be connected directly without additional calculation. The membership functions of σ_i, $\sigma_i(t-1)$, $\sigma_i(t-2)$ used in noise reduction stage are shown in Fig. 10. We use triangular shape for each fuzzy membership function.

B. Optimization of fuzzy controller membership functions

As the fine-tuning of fuzzy membership functions is generally based on trial-and-error, we attempt to use our modified PSO as a tool to reduce trial time. By using PSO, we only specify initial positions of membership functions and let the PSO to do the fine-tuning. Since input membership functions cause major effect to system performance, we only optimize them and leave output membership function and fuzzy rules as defined at the beginning. Since a crisp input consists of 3 fuzzy membership functions, it can be represented as a vector shown in (29) where the meaning of each element is shown in Fig. 11. We maintain system stability by defining that the point, which a fuzzy membership function deceases, should be the starting point of another membership function. For example, in



(a) Membership function of noise-reduced σ_i

(b) Membership function of θ_i

(c) Membership function of β_i

(d) Membership function of $\Delta\beta_i$

Fig. 9. Initial fuzzy-membership-functions

Fig. 11, a membership function, $Normal$, starts decreasing at x_1; thus, x_1 should be the starting point of another membership function, $SlightlyHigh$.

$$P = [x_1, x_2, x_3, x_4] \tag{29}$$

The points x_1, x_2, x_3, and x_4 are the relative distances where membership functions have no strength (i.e., $\mu = 0$). Then, the concatenation of all input representative vectors is used as the position in PSO solution space. We have 3 crisp inputs with each having 4-dimension representative vector as a position in PSO solution space, which is represented as a 12-dimension vector. For example, the initial fuzzy membership functions in Fig. 9 are

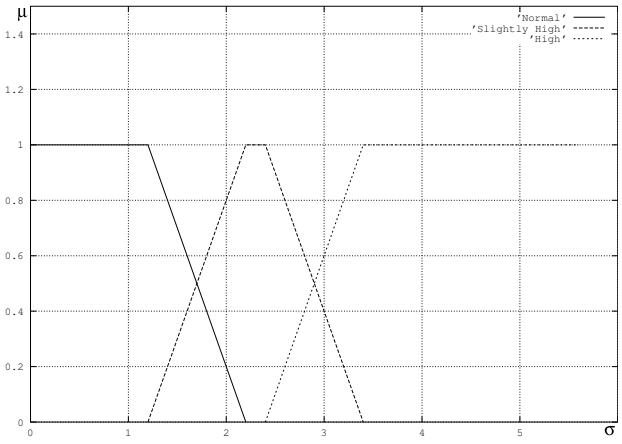


Fig. 10. Membership function of σ_i, $\sigma_i(t-1)$, and $\sigma_i(t-2)$

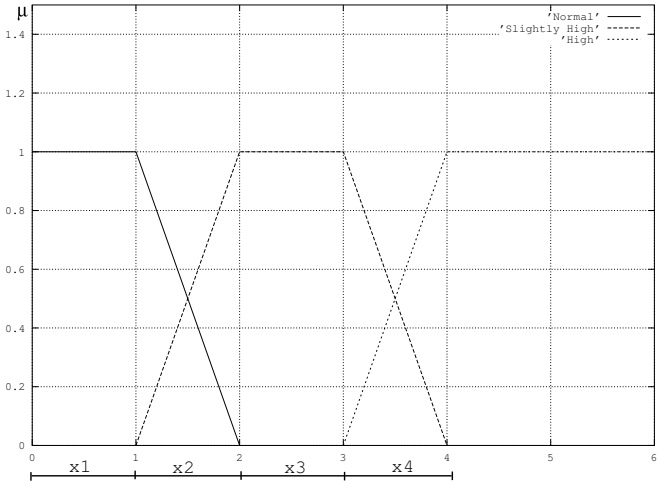


Fig. 11. Representation of membership-function's position

represented as:

$$\begin{aligned} P &= [1.0, 2.0, 0.0, 2.0, \\ &\quad 1.0, 2.0, 0.0, 2.0, \\ &\quad 0.0, 5.0, 0.0, 5.0] \end{aligned} \tag{30}$$

To evaluate the quality of each particle, we operate the simulation model, shown in Fig. 12 with the configurations of S0 to S3 same as S0 to S3 in Section VI. The measured queue-delays of all classes when simulated without any incoming burst and adjustment are saved as the base-delay values. After that, we run the simulator with the fuzzy controller turned on and burst entering from each class from 1 to 3. The membership functions of the fuzzy controller in latter simulation are generated by the position represented in each particle. The quality number of the particular particle is the maximum delay difference at corresponding traffic-class and simulation-time between

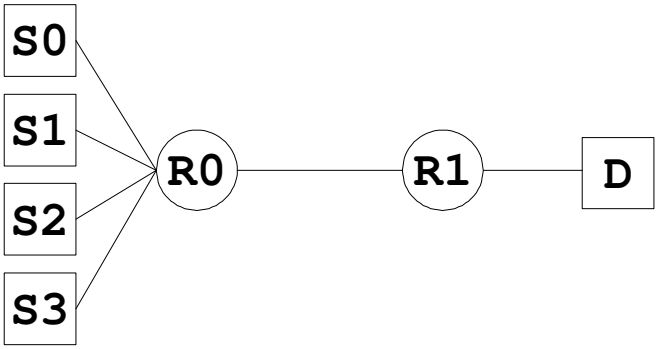


Fig. 12. Simulation model used to tune fuzzy membership functions

classes with higher priority than the controlled class, the class that we pump bursty data into, and the base-delay. The best particle is the one with the least quality number.

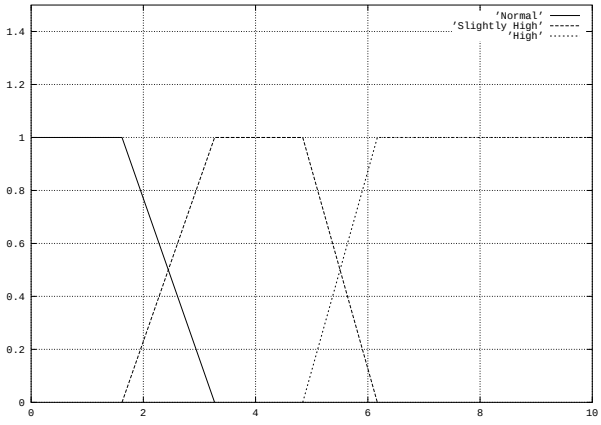
After running PSO for 1000 iterations, the optimal position in solution space for the system with $\tau = 2$ is given out as follows. The tuned membership functions which have the fitness value of 1.28 are shown in Fig. 13.

$$\begin{aligned} P &= [1.62, 1.65, 1.58, 1.33, \\ &\quad 1.52, 0.91, 1.42, 1.73, \\ &\quad 2.98, 2.65, 3.38, 0.99] \end{aligned} \tag{31}$$

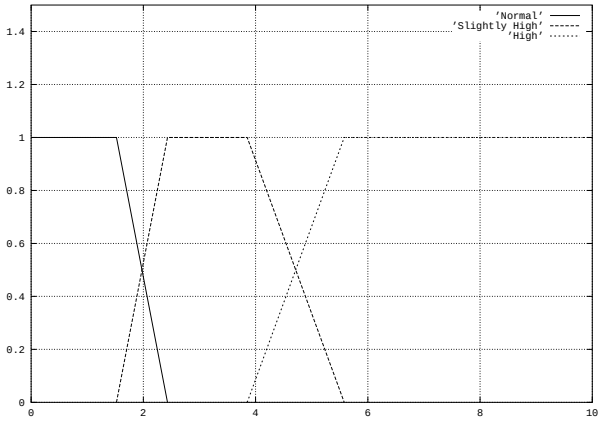
VI. Simulation Configuration

In order to study our scheme, simulations were conducted using the topology shown in Fig. 14. We have used NS2 network simulator developed by National Berkeley Laboratory as the simulation platform [27]. The topology consists of five main nodes: four User-Traffic sources (S0 to S3) and one User-Traffic sink (SK), and three DiffServ routers (R0 to R2). To simulate a more realistic traffic scenario, R0 and R1 are each attached to four traffic sources which generate Cross-Traffic flows for each traffic class. Each Cross-Traffic flow has destination at Cross-Traffic sink attached to the next hop router (e.g. the Cross-Traffic flows from sources attached to R0 have destination at the Cross-Traffic sink attached to R1). All connections in the system have capacity of 1 Mbps with $10\mu s$ delay. All User-Traffic sources send traffics to the User-Traffic sink through R0, R1, and R2 sequentially. As we consider only queue delay, the buffers in all routers are assumed to be of infinite size. Packets between routers are classified into 4 classes where class 0 is the highest priority and class 3 is the lowest one. We assume that packets generated from S0 are categorized into class 0, packets generated from S1 are grouped into class 1, and so on for the other sources. In contrast to [12] and [13], in which the implementation of the controller on proportional DiffServ scheme was not influenced by the idea of fairness, we apply the new controller with fairness to [10].

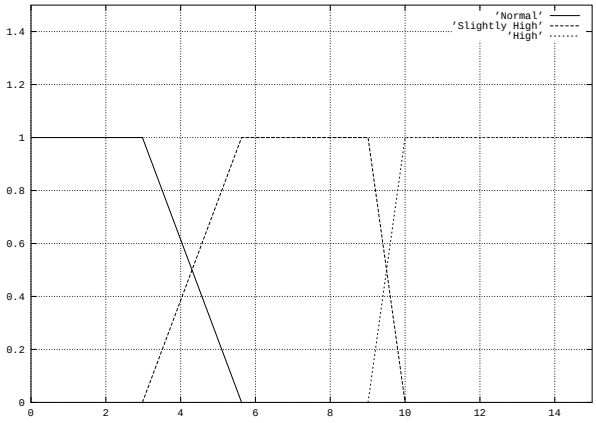
Each of the both User-Traffic and Control-Traffic sources, generates 150 kbps exponential and 50 kbps constant-rate traffic as the normal operation. We conducted the simulation for 500 seconds. To simulate burst bulky data, we generate 500 kbps constant rate traffic to a selected User-Traffic source starting at 100 second for 50-second duration.



(a) Membership function of σ_i



(b) Membership function of θ_i



(c) Membership function of β_i

Fig. 13. Optimized fuzzy-membership-functions

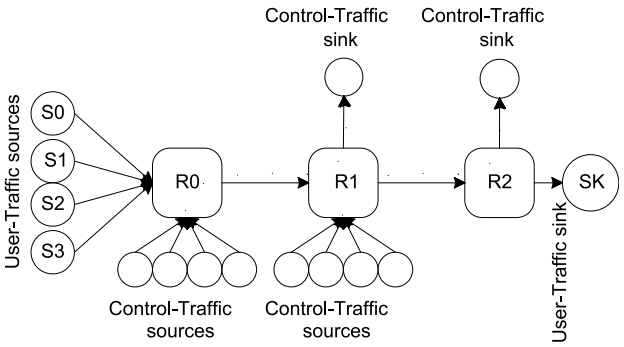
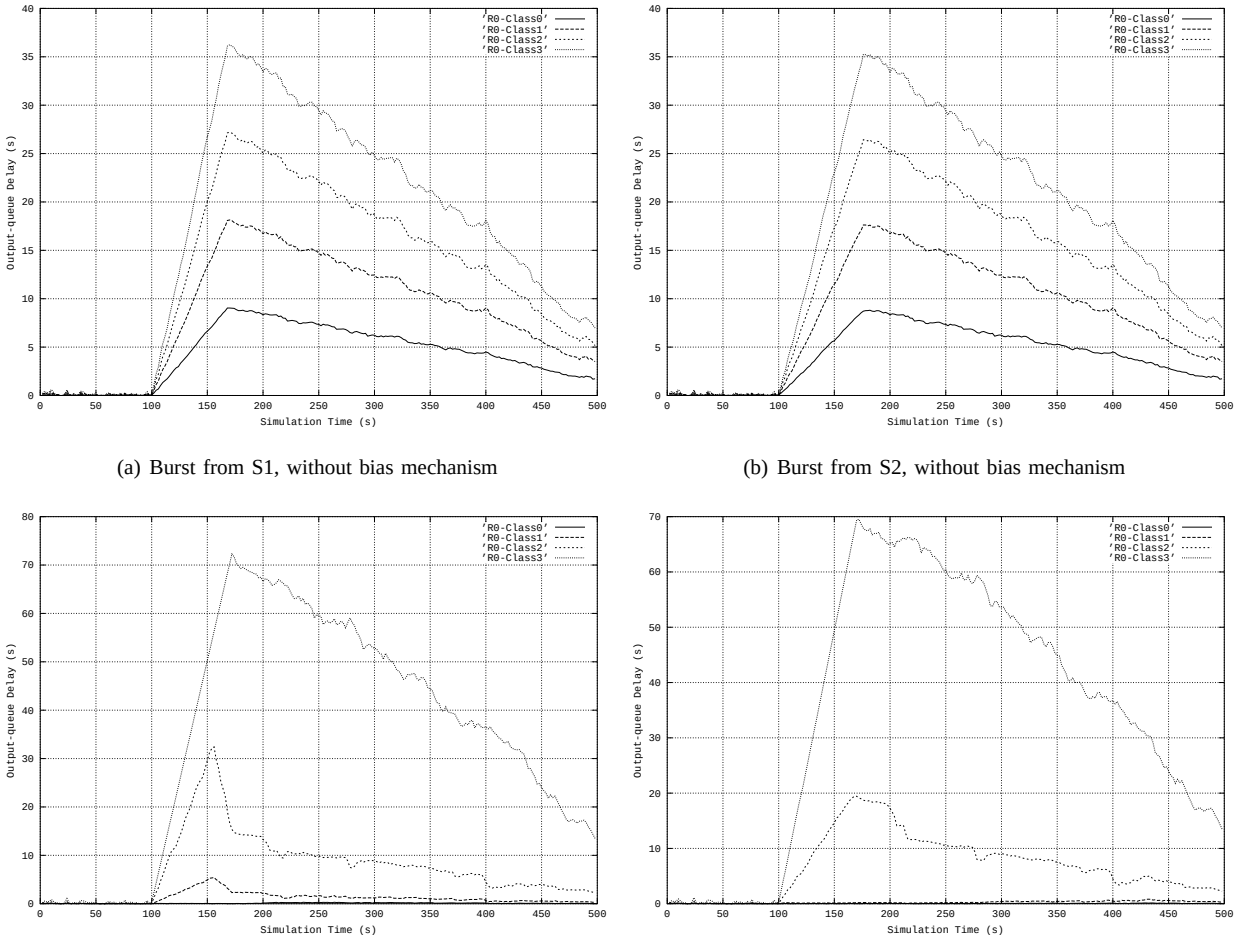


Fig. 14. Simulation configuration

We use the packet size 1024 bytes for the purpose of solution. Moreover, we configure the delay differentiation ratios between classes as $\delta_0 : \delta_1 : \delta_2 : \delta_3$ to 1:2:4:8.

VII. Simulation Result



(a) Burst from S1, without bias mechanism

(b) Burst from S2, without bias mechanism

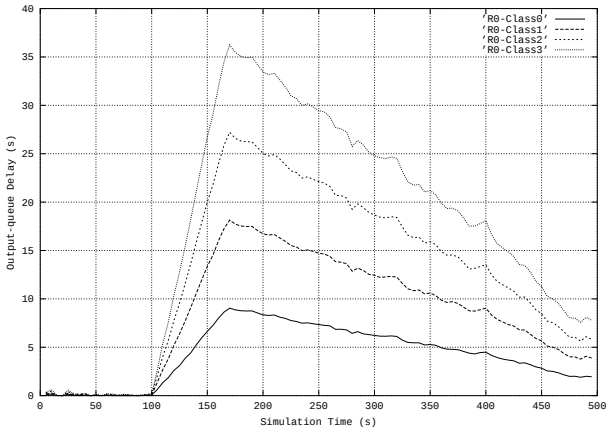
(c) Burst from S1, with bias mechanism at R0

(d) Burst from S2, with bias mechanism at R0

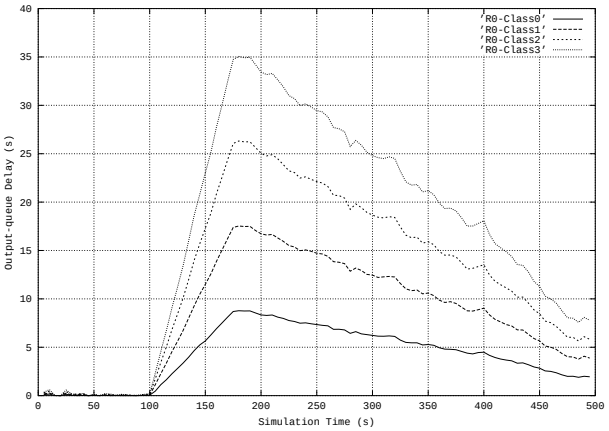
Fig. 15. Simulation results with $\tau = 2$

The results of the simulation are shown as graphs of queue delay in router R0. The simulation aims to compare the delay in conventional Prop-DiffServ and the Prop-DiffServ with the PSO-tuned fuzzy controller. We also show the response of the system with different value of τ by comparing the result between $\tau = 2$ and $\tau = 5$. We simulate the topology with four scenarios:

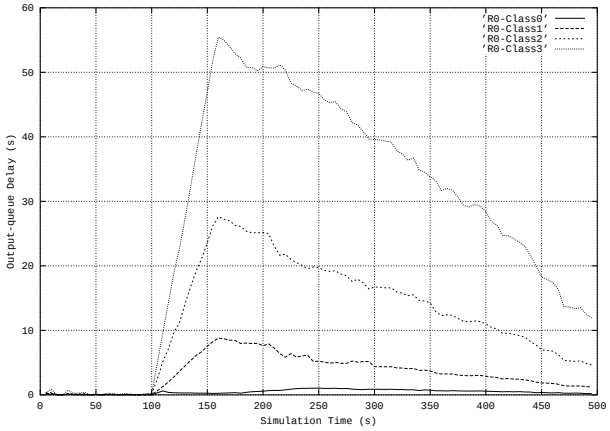
1) Bursty traffic enters from S1 without our bias mechanism.
2) Bursty traffic enters from S2 without our bias mechanism.
3) Bursty traffic enters from S1 with our bias mechanism in R0.



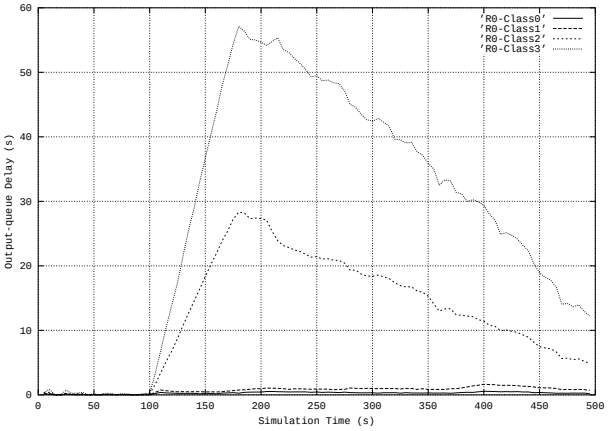
(a) Burst from S1, without bias mechanism



(b) Burst from S2, without bias mechanism



(c) Burst from S1, with bias mechanism at R0



(d) Burst from S2, with bias mechanism at R0

Fig. 16. Simulation results with $\tau = 5$

4) Bursty traffic enters from S2 with our bias mechanism in R0.

Four graphs from each scenario with $\tau = 2$ are presented in Fig. 15(a) to Fig. 15(d) respectively and those with $\tau = 5$ are shown in Fig. 16(a) to Fig. 16(d). The X-axis of all graphs is simulation time in seconds and the Y-axis is queue delay in seconds. In the graphs, the results from the first two scenarios, Fig. 16(a), 16(b), 15(a), and 15(b), are very similar because the system strictly maintains the delay ratio regardless of the traffic source. When we turn on the fuzzy controller in R0 and re-simulate the scenarios again, the output graphs, shown in Fig. 16(c), 16(d), 15(c), and 15(d), indicate that the effect of bursty traffic upon the classes, which have higher priority than the particular class, is significantly reduced. Comparing between Fig. 16(a) and 16(c), the area under class-0 line in Fig. 16(c) (the first line from bottom) is less than that of Fig. 16(a). The same is true for Fig. 16(b) and 16(d), Fig. 15(a) and 15(c), and Fig. 15(b) and 15(d) where we change the value of τ and class to which the bursty traffic coming in. In Fig. 16(d) and 15(d), the area under class 1 (the second line from bottom) and class 0 are less than

the correspondent areas in Fig. 16(b) and 15(b).

The effect of τ can be seen by comparing corresponding graphs in Fig. 16(c)(d) and 15(c)(d). Because the system operates predictively, it cannot immediately response to the incoming burst. The larger τ system has the problem to adjust the proportion on time. This causes the system with $\tau = 5$ has higher delay in classes which have higher priority than the class with burst. This means that the less τ the system has, the better adjustment performance it is. However, reducing τ means increasing processing time. The optimized τ for each system should be carefully considered by the network administrators.

Another simulation was performed to demonstrate the good performance of our scheme when there are multiple burst flows. During this simulation, which is extended to 1000 seconds, S1 pumps bursty traffic from 100 to 300 second and S2 generates burst with the same characteristic from 200 to 400 second. The value of τ is set to 2 throughout this simulation. The result is shown in Fig. 17. From the output graph, it can be seen that class 1, the dash line near the x-axis, now suffers some amount of higher delay compared to previous simulations with burst coming from S2. It is because class 1 is also a culprit of pumping bursty data into the system. However, the ratio between class 1 and class 2 is higher than 2:4 because class 2 also pumps bursty data too.

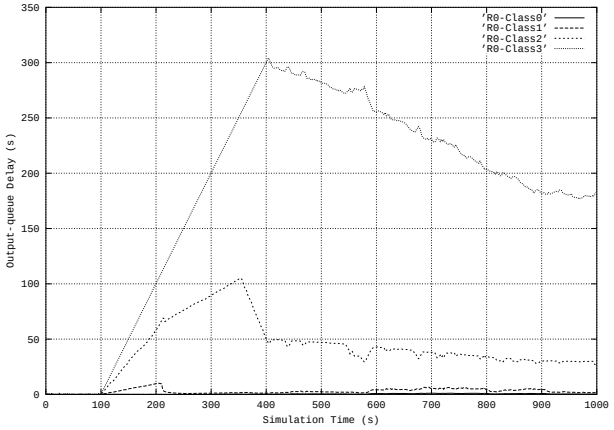


Fig. 17. Simulation result with burst coming from S1 and S2 and $\tau = 2$

VIII. Conclusion

We investigated the unfairness problem in proportional differentiated services. This unfairness enables a low priority traffic-class to degrade service quality of higher priority classes by inducing large amount of traffic into itself. We also present a solution to the problem by using an independent fuzzy controller that can also be considered as an agent. This new fuzzy agent periodically monitors incoming traffic condition of all traffic classes and adjusts delay proportion to minimize the unfairness problem. The fuzzy agent used in this work is a hierarchical fuzzy controller designed to manage highly dynamic changes in network traffic. Moreover, to calibrate the agent parameters to their optimum values, we introduce a new modified particle-swarm optimization method that works essentially

off-line. We verify that the fuzzy agent is nearly optimized by using different and more complex network topologies than the ones used in the tuning process to evaluate the final parameters. Simulation shows that the optimized fuzzy agent can effectively reduce the unfairness problem even when the bursty traffic is fed into the compound classes. The results mean that the fuzzy agent can provide more fairness upon load distribution among different traffic classes. We assume that the queue in each router is infinite and do not consider other QoS parameters such as packet-loss. Our solution can be further improved by including more parameters such as packet loss into the system.

APPENDIX I
NOISE REDUCTION RULES

Let σ_c denote σ received from preprocessor module and σ_s be the output of these rules. The symbol $\sigma(\tau)$ means the σ at discrete time τ.

- IF $\sigma_c(\tau)$ IS normal AND $\sigma_c(\tau-1)$ IS normal THEN σ_s IS normal
- IF $\sigma_c(\tau)$ IS slightly_high AND $\sigma_c(\tau-1)$ IS slightly_high THEN σ_s IS slightly_high
- IF $\sigma_c(\tau)$ IS high AND $\sigma_c(\tau-1)$ IS high THEN σ_s IS high
- IF $\sigma_c(\tau)$ IS normal AND $\sigma_c(\tau-1)$ IS slightly_high AND $\sigma_c(\tau-2)$ IS normal THEN σ_s IS normal
- IF $\sigma_c(\tau)$ IS normal AND $\sigma_c(\tau-1)$ IS slightly_high AND $\sigma_c(\tau-2)$ IS slightly_high THEN σ_s IS slightly_high
- IF $\sigma_c(\tau)$ IS normal AND $\sigma_c(\tau-1)$ IS slightly_high AND $\sigma_c(\tau-2)$ IS high THEN σ_s IS slightly_high
- IF $\sigma_c(\tau)$ IS normal AND $\sigma_c(\tau-1)$ IS high AND $\sigma_c(\tau-2)$ IS normal THEN σ_s IS normal
- IF $\sigma_c(\tau)$ IS normal AND $\sigma_c(\tau-1)$ IS high AND $\sigma_c(\tau-2)$ IS slightly_high THEN σ_s IS slightly_high
- IF $\sigma_c(\tau)$ IS normal AND $\sigma_c(\tau-1)$ IS high AND $\sigma_c(\tau-2)$ IS high THEN σ_s IS high
- IF $\sigma_c(\tau)$ IS slightly_high AND $\sigma_c(\tau-1)$ IS normal AND $\sigma_c(\tau-2)$ IS normal THEN σ_s IS normal
- IF $\sigma_c(\tau)$ IS slightly_high AND $\sigma_c(\tau-1)$ IS normal AND $\sigma_c(\tau-2)$ IS slightly_high THEN σ_s IS slightly_high
- IF $\sigma_c(\tau)$ IS slightly_high AND $\sigma_c(\tau-1)$ IS normal AND $\sigma_c(\tau-2)$ IS high THEN σ_s IS slightly_high
- IF $\sigma_c(\tau)$ IS slightly_high AND $\sigma_c(\tau-1)$ IS high AND $\sigma_c(\tau-2)$ IS normal THEN σ_s IS slightly_high
- IF $\sigma_c(\tau)$ IS slightly_high AND $\sigma_c(\tau-1)$ IS high AND $\sigma_c(\tau-2)$ IS slightly_high THEN σ_s IS slightly_high
- IF $\sigma_c(\tau)$ IS slightly_high AND $\sigma_c(\tau-1)$ IS high AND $\sigma_c(\tau-2)$ IS high THEN σ_s IS high
- IF $\sigma_c(\tau)$ IS high AND $\sigma_c(\tau-1)$ IS normal AND $\sigma_c(\tau-2)$ IS normal THEN σ_s IS normal
- IF $\sigma_c(\tau)$ IS high AND $\sigma_c(\tau-1)$ IS normal AND $\sigma_c(\tau-2)$ IS slightly_high THEN σ_s IS slightly_high
- IF $\sigma_c(\tau)$ IS high AND $\sigma_c(\tau-1)$ IS normal AND $\sigma_c(\tau-2)$ IS high THEN σ_s IS high
- IF $\sigma_c(\tau)$ IS high AND $\sigma_c(\tau-1)$ IS slightly_high AND $\sigma_c(\tau-2)$ IS normal THEN σ_s IS slightly_high
- IF $\sigma_c(\tau)$ IS high AND $\sigma_c(\tau-1)$ IS slightly_high AND $\sigma_c(\tau-2)$ IS slightly_high THEN σ_s IS slightly_high
- IF $\sigma_c(\tau)$ IS high AND $\sigma_c(\tau-1)$ IS slightly_high AND $\sigma_c(\tau-2)$ IS high THEN σ_s IS high

APPENDIX II
MAIN FUZZY-CONTROLLER'S RULES

- IF σ_s IS high THEN $\Delta\beta$ IS large_increase

- IF σ_s IS slightly_high AND θ IS normal THEN $\Delta\beta$ IS large_increase
- IF σ_s IS slightly_high AND θ IS slightly_high THEN $\Delta\beta$ IS slightly_increase
- IF σ_s IS slightly_high AND θ IS high THEN $\Delta\beta$ IS slightly_increase
- IF σ_s IS normal AND θ IS high THEN $\Delta\beta$ IS no_change
- IF σ_s IS normal AND θ IS normal AND β IS normal THEN $\Delta\beta$ IS no_change
- IF σ_s IS normal AND θ IS normal AND β IS slightly_high THEN $\Delta\beta$ IS slightly_decrease
- IF σ_s IS normal AND θ IS normal AND β IS high THEN $\Delta\beta$ IS large_decrease
- IF σ_s IS normal AND θ IS slightly_high AND β IS normal THEN $\Delta\beta$ IS no_change
- IF σ_s IS normal AND θ IS slightly_high AND β IS slightly_high THEN $\Delta\beta$ IS no_change
- IF σ_s IS normal AND θ IS slightly_high AND β IS high THEN $\Delta\beta$ IS slightly_decrease

REFERENCES

[1] P. White, "RSVP and integrated services in the internet: A tutorial," *IEEE Commun. Mag.*, pp. 100–106, May 1997.

[2] R. Guerin, S. Kamat, V. Peris, and R. Rajan, "Scalable QoS provision through buffer management," in *Proc. ACM SIGCOMM'98*, 1998.

[3] I. Stoika, S. Shenker, and H. Zhang, "Core-stateless fair queuing: Achieving approximately fair bandwidth allocations in high speed networks," in *Proc. ACM SIGCOMM'98*, 1998.

[4] S. Blake and *et.al.*, "An architecture for differentiated services," RFC 2475, Dec 1998.

[5] K. Nichols, S. Blake, F. Baker, and D. Black, "Definition of the differentiated services field (DS field) in the IPv4 and IPv6 headers," RFC 2474, 1998.

[6] V. Jacobson, K. Nichols, and K. Poduri, "An expedited forwarding PHB," RFC 2598, 1999.

[7] R. Sivakumar, T. Kim, N. Venkitaraman, and V. Bharghavan, "Achieving per-flow weighted rate fairness in a core stateless network," in *Proc. IEEE ICDCS 2000*, 2000.

[8] I. Stoica and H. Zhang, "Providing guaranteed services without per flow management," in *Proc. ACM SIGCOMM'99*, Aug 1999.

[9] A. Odlyzko, "Paris metro pricing: The minimalist differentiated services solution," in *Proc. IEEE/IFIP IWQoS'99*, Jun 1999.

[10] C. Dovrolis, D. Stiliadis, and P. Ramanathan, "Proportional differentiated services: Delay differentiation and packet scheduling," in *Proc. ACM SIGCOMM'99*, Aug 1999.

[11] J. Heinanen and *et.al.*, "Assured forwarding PHB group," RFC 2597, 1999.

[12] C. Li, S. Tsao, M. Chen, Y. Sun, and Y. Huang, "Proportional delay differentiation service based on weighted fair queuing," in *Proc. IEEE ICCCN 2000*, 2000.

[13] M. Leung, J. Lui, and D. Yau, "Adaptive proportional delay differentiated services: characterization and performance evaluation," *IEEE/ACM Trans. Networking*, vol. 9, no. 6, pp. 801–817, Dec 2001.

[14] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. IEEE ICNN'95*, 1995, pp. 1942–1948.

[15] L. Zadeh, "Fuzzy sets," *Information and Control*, vol. 8, no. 3, pp. 338–353, 1965.

[16] ——, "Soft computing and fuzzy logic," *IEEE Software*, vol. 11, no. 6, 1994.

[17] N. Christtin, J. Liebeherr, and T. Abdelzaher, "A quantitative assured forwarding service," in *Proc. IEEE INFOCOM 2002*, Jun 2002.

[18] R. Eberhart and Y. Shi, "Particle swarm optimization: Developments, applications, and resources," in *Proc. IEEE International Conference on Evolutionary Computation, Vol.1*, 2000, pp. 84–88.

[19] M. Clerc, "The swarm and the queen: Towards a deterministic and adaptive particle swarm optimization," in *Proc. IEEE International Conference on Evolutionary Computation, Vol.3*, 1999.

[20] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *Proc. IEEE International Conference on Evolutionary Computation*, 1998, pp. 69–73.

[21] ——, "Parameter selection in particle swarm optimization," in *Proc. International Conference on Evolutionary Programming*, 1998, pp. 591–600.

[22] A. Ratnaweera, S. Halgamuge, and H. Watson, "Particle swarm optimization with time varying acceleration coefficients," in *Proc. 1^{st} International Conference on Soft Computing and Intelligent Systems*, 2002.

[23] C. Karr, "Design of an adaptive fuzzy logic controller using a genetic algorithm," in *Proc. 4^{th} International Conference on Genetic Algorithms*, Jul 1991, pp. 450–457.

[24] C. Karr and E. Gentry, "Fuzzy control of ph using genetic algorithms," *IEEE Trans. Fuzzy Syst.*, vol. 12, pp. 46–53, Feb 1993.

[25] A. Homaifar and E. McCormick, "Simultaneous design of membership functions and rule sets for fuzzy controllers using genetic algorithms," *IEEE Trans. Fuzzy Syst.*, vol. 3, pp. 129–139, May 1995.

[26] D. Linkens and H. Nyongesa, "Genetic algorithms for fuzzy control-part I and II," in *Inst. Elect. Eng. Proc. Control Theory Application, vol.142, no. 3*, 1995, pp. 161–176.

[27] "Network simulator (ns) version 2 software." [Online]. Available: http://www.isi.edu/nsnam/ns/