Optimal Call Admission Control in Generalized Processor Sharing (GPS) Schedulers

Nandita D, Joy Kuri, H.S. Jamadagni

Abstract—**Generalized Processor Sharing(GPS) is an idealized fluid discipline with a number of desirable properties. Its packetized version PGPS is considered to be a good choice as a packet scheduling discipline to guarantee Quality-of-Service in IP and ATM networks. The existing Connection Admission Control(CAC) frameworks for GPS result in a conservative resource allocation. We propose an Optimal CAC algorithm for a GPS scheduler, for leaky-bucket constrained connections with deterministic delay guarantees. Our numerical results show that the Optimal CAC results in a higher network utilization than the existing CACs.**

Keywords—**Generalized Processor Sharing, Call Admission Control, Optimal Bandwidth Allocation, Non-Rate Proportional Resource Allocation.**

I. Introduction

A. Motivation

PROVISION of Quality-of-Service (QoS) guarantees is an important issue in the design of Integrated Services packet networks. Call admission control is an integral part of the problem. Clearly, without call admission control, providing QoS guarantees will be impossible. The task of call admission control is put forth in the following question:

Given a new call/session that arrives to a network, can it be accepted by the network at its requested QoS, without violating existing guarantees made to the on-going calls?

This question turns out to be complicated, as the issue of call admission control (CAC) is closely related to the other aspects of a network, such as service models, scheduling disciplines, traffic characterization and QoS specification. A key challenge in the design of a CAC algorithm is to support a large number of sessions with different performance requirements, while minimizing cost as measured by network resources. Session performance is mainly characterized by packet delay and loss probability, with link bandwidth and buffer space being the network resources that must be expended to achieve performance.

Packet-scheduling disciplines are necessary to satisfy the QoS requirements of delay-sensitive applications, and ensure that real-time applications and best effort traffic can coexist on the same network structure. The CAC algorithm at a switch depends on the packet-scheduling discipline employed. Among the scheduling algorithms that have been proposed in the literature, the class of schemes based on Generalized Processor Sharing (GPS) are most popular.

GPS [1], [2] is an idealized fluid discipline with a number of very desirable properties, including the provision of minimum bandwidth guarantees to each connection regardless of the behavior of the other connections (perfect isolation), and the provision of deterministic, easily computable end-to-end delay bounds for sessions whose traffic is leaky-bucket constrained. Because of its powerful properties, GPS has become the reference for an entire class of GPS-related packet-scheduling disciplines, and has been extensively studied in the literature. The existing CAC frameworks for GPS, though simple, result in a substantial portion of the available bandwidth being wasted. A CAC framework for GPS which meets the connections'(deterministic) delay requirements and allots bandwidth optimally, is the subject matter of this paper.

Center for Electronics Design and Technology, Indian Institute of Science, Bangalore, India. E-mail: {dnandita, kuri, hsjam}@cedt.iisc.ernet.in

B. Relation to Previous Work

The GPS discipline was first proposed as Weighted Fair Queueing (WFQ) in [3]. In their seminal papers [1], [2] on GPS, Parekh and Gallager have analyzed the GPS scheduling discipline in a deterministic setting where the traffic of each session is regulated by a (σ, ρ)- leaky bucket regulator. In the single node and the multiple node case they obtained closed form expressions for the bounds on delay and backlog for a certain class of GPS schedulers called *Rate Proportional Processor Sharing* (RPPS) schedulers. In [4], Zhang *et al* have obtained closed-form expressions for end-to-end performance bounds for a broader class of GPS networks known as *Consistent Relative Session Treatment* (CRST) GPS networks. In [5], Yaron and Sidi studied GPS networks with exponentially bounded burstiness arrivals. In [6] Zhang *et al* have investigated the behavior of GPS in a stochastic setting. The above mentioned papers deal with the problem of obtaining a bound on delay and backlog for a session in GPS schedulers, *given* a particular rate allocation for the session.

The inverse problem of mapping the QoS requirements (delay in particular) of the sessions to bandwidth allocations in GPS schedulers is of practical importance. Kesidis *et al* in [7] and Z.L.Zhang *et al* in [8] address this problem in a stochastic setting (stochastic arrival processes and statistical guarantees) with packet loss probability as the QoS metric. In [9] the above problem is addressed for *Leaky Bucket Regulated* connections and statistical delay guarantees. In [10] Robert Szabó *et al* addressed the call admission control in GPS schedulers for *leaky bucket* regulated traffic with deterministic delay guarantees. The CACs established in [10] iteratively search on a vector space for suitable bandwidth allocations. This could result in an over allocation of bandwidth and sometimes a call block, even if the bandwidth necessary to guarantee the call's QoS requirements is available. The CACs are also computationally complex.

The papers mentioned above do not address the problem of reserving bandwidth *optimally* in GPS schedulers to provide deterministic delay guarantees.

C. Research Contributions of This Paper

A large sub-class of GPS based schedulers is Rate Proportional Servers (RPS) [1]. In RPS, the rates of the sessions are

set according to the session bandwidth demands, i.e., the rate allotted to a session is at least equal to its long term average rate, ρ. This may lead to a waste of network resources. The CAC algorithm presented in this paper does:

• A *General Resource Allocation* or a *Non-Rate Proportional* resource allocation in GPS schedulers which can result in higher network utilization.

• An *Optimal* bandwidth allocation in GPS schedulers to guarantee the connections' delay requirements.

The rest of the paper is organized as follows: Section II describes the GPS scheduling policy and gives the necessary background. Section III describes the optimal call admission control (CAC). In Section IV we present numerical results to demonstrate the effectiveness and the performance of the Optimal CAC. Section V concludes the paper with a discussion on future research issues.

II. Preliminaries

A. Definitions and Notation

We assume a packet switch where each output link uses the Generalized Processor Sharing (GPS) scheduling policy to schedule packets from the set of sessions arriving to the link.

Leaky Bucket characterization of a traffic stream is based on specifying a two parameter (σ, ρ) envelope on the volume of the arriving traffic [11].

Definition 1: Let $A_i(\tau, t)$ denote arrivals from session i during the interval $(\tau, t]$. Given $\sigma > 0$, and $\rho > 0$, the traffic stream $A(t)$ is called $(\sigma, \rho) - regular$, if for any interval $[t_1, t_2)$

$$A[t_1, t_2) \leq \sigma + \rho(t_2 - t_1)$$

In this paper we assume that the arrival processes are *Leaky Bucket* constrained. A session i also specifies a delay guarantee d_i, which is the maximum difference allowed between the departure epoch and the arrival epoch of any bit of session i, at the link. Let r_i denote the bandwidth allotted to session i at the link to satisfy it's delay guarantee d_i.

Let $W_i(\tau, t)$ denote the amount of service received by session i during the same interval. Let $Q_i(t)$ represent the amount of session i traffic queued in the server at time t, that is

$$Q_i(t) = A_i(0, t) - W_i(0, t)$$

A session i is backlogged at time t if $Q_i(t) > 0$.

Definition 2: A *backlogged period* for a session i is any period of time during which packets belonging to that session are continuously queued in the system [1].

A session i is said to be *greedy* starting from time τ if it sends traffic at the maximum possible rate for all times $\geq \tau$ [1].

Definition 3: An *All-Greedy GPS System* is one in which all the sessions are greedy starting from time 0, that is [1]:

$$A_i(0, \tau) = \sigma_i + \rho_i \, \tau, \quad \tau \geq 0$$

B. Generalized Processor Sharing (GPS) Scheduling

Generalized Processor Sharing (GPS) [1] is a scheduling discipline that can be regarded as the limiting form of a weighted round robin policy, where the traffic from the sessions is treated as an infinitely divisible fluid. A GPS server serving N sessions is characterized by N positive real numbers, $\phi_1, \ldots, \phi_N$. These numbers denote the relative amount of service given to each session in the sense that if $W_i(\tau, t)$ is defined as the amount of session i traffic served by the GPS server during an interval $[\tau, t)$, then:

$$\frac{W_i(\tau, t)}{W_j(\tau, t)} \geq \frac{\phi_i}{\phi_j}, \quad j = 1, \ldots, N \tag{1}$$

for any session i that is continuously backlogged in the interval $[\tau, t]$. Thus, (1) is satisfied with equality for two sessions i and j that are both backlogged during the interval $[\tau, t]$. Note from (1) that whenever session i is backlogged it is guaranteed a minimum service rate of

$$r_i = \frac{\phi_i}{\sum_{j=1}^{N} \phi_j} C \tag{2}$$

where C is the capacity of the server. This rate is called the session i *backlog clearing rate* [1] since a session i backlog of size q is served in at most $\frac{q}{r_i}$ time units. The server is said to be stable if $\sum_{i=1}^{N} \rho_i < C$, i.e., the sum of the long term sustained rates of the sessions is less than the service rate.

C. Delay Bounds of Leaky Bucket Constrained Sessions in a GPS Server

The arrival process of a session i is assumed to be $(\sigma_i, \rho_i) - regular$.

Parekh and Gallager obtained the following bounds in [1] for the maximum queueing delay, d_i^* and the maximum backlog, q_i^* for a session i:

$$q_i^* \leq \sigma_i \;\; and \;\; d_i^* \leq \frac{\sigma_i}{r_i} \tag{3}$$

Further, it was also shown that any session i in the GPS system achieves its maximum delay and backlog in an *all-greedy regime*. This gives a simple scenario for investigating the worst case behavior.

However, the bounds obtained above are quite loose since the assumption is that the session i is served with a constant rate r_i till its backlog is cleared. Also, the bound holds only for a sub-class of GPS-based systems called *Rate Proportional Servers* in which the guaranteed rate $r_i \geq \rho_i$. In reality, the service rates of backlogged sessions increase as more and more sessions complete their backlogged periods. This is because as each session completes its backlog, the bandwidth released is distributed among the still backlogged sessions in proportion to their weights. Fig.1 illustrates the looseness of the bounds in (3) due to not accounting for the excess bandwidth received by the session i. Robert Szabó *et al* address this problem in [12] and present a two step algorithm for calculating tighter upper bounds for delay d_i^*. This algorithm takes care of the backlog finish times of the sessions and can also be used for arbitrary sets of $\{\phi\}$ values (*Non-Rate Proportional*) and thus overcomes both the problems associated with the bound in (3).

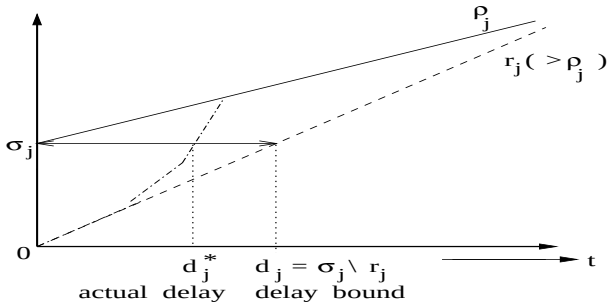

Fig. 1. Illustrates the looseness of the delay bound

The following notation is used in the algorithm and will be used henceforth in the subsequent sections too. Let C denote the server capacity and $\mathcal{L} = \{L(i)|i = 1, \ldots, N\}$ be an ordered set of indices in which $L(i)$ means that the session $L(i)$ backlog is cleared i^{th} in order. The time instant when the backlog of session $L(i)$ becomes zero is denoted by $t_{L(i)}$. By definition, let $t_{L(0)} = 0$, the start of the system busy period with all sessions greedy. Further, let $r_j^{(k)}$ be defined as the session j service rate in the $[t_{L(k-1)}, t_{L(k)}]$ interval. During the interval $[0, t_{L(1)}]$ every session is served by its minimum guaranteed rate i.e.

$$r_j^{(1)} = r_j$$

Between $t_{L(1)}$ and $t_{L(2)}$, session $L(1)$ has no backlog, so it is served at rate $\rho_{L(1)}$. The service rates of the still backlogged sessions during that period are

$$r_j^{(2)} = r_j + \frac{r_j}{\sum_{i=1}^{N} r_i - r_{L(1)}} \ (r_{L(1)} - \rho_{L(1)})$$

$$j \ \in \{1, \ldots, N\} \backslash \{L(1)\}$$

The second term on the right hand side comes from the fact that after clearing the backlog of session $L(1)$, the remaining bandwidth $(r_{L(1)} - \rho_{L(1)})$ is distributed among the still backlogged sessions in proportion to their allotted rates.

It is proved in [12] that during the time interval $[t_{L(k-1)}, t_{L(k)}]$, $k = 1, \ldots, N$, within the system busy period, the backlogged session j is served at a rate

$$r_j^{(k)} = \frac{(C - \sum_{l=1}^{k-1} \rho_{L(l)}) r_j}{\sum_{i=1}^{N} r_i - \sum_{l=1}^{k-1} r_{L(l)}}, \ j \ \in \{1, \ldots, N\} \backslash \{ \bigcup_{m=1}^{k-1} L(m) \} \quad (4)$$

The two step algorithm [12] for calculating tight upper bounds on delay is described in the following:

Step 1: Algorithm for Calculating Backlog Clearing Times
In the first step, the time needed for a session i to empty its backlog is expressed as $t_{i,1} = \frac{\sigma_i}{r_i^{(1)} - \rho_i}$. The session that completes its backlog first is the one whose t_i takes the smallest possible value. That is:

$$t_{L(1)} = \min\{t_{i,1} | t_{i,1} > 0; \ \ i \in \{1, \ldots, N\}\}$$

where $L(1)$ is the index of that session which satisfies the above minimum.

The k^{th} step candidate finishing times of sessions still backlogged are:

$$t_{i,k} = \frac{\sigma_i + \sum_{j=1}^{k-1} r_i^{(j)} (t_{L(j-1)} - t_{L(j)}) + r_i^{(k)} t_{L(k-1)}}{r_i^{(k)} - \rho_i} \quad (5)$$

The k^{th} step finishing time is calculated by taking the minimum of (5) over $i \in \{1, \ldots, N\} \backslash \{\bigcup_{j=1}^{k-1} L(j)\}$ i.e.

$$t_{L(k)} = \min\{t_{i,k} | t_{i,k} > 0; \ \ i \in \{1, \ldots, N\} \backslash \bigcup_{j=1}^{k-1} L(j)\}$$

Step 2: Calculating Tighter Upper Bounds for Delay
The following Theorem [12] gives the tight bounds for the delay, denoted by d_i^*:

Theorem 1: Let $r_i(t) = r_i^{(k)}, \ \ where \ \ t \in [t_{L(k-1)}, t_{L(k)}]$
(i) If

$$r_i(\tau_i) \geq \rho_i \quad (6)$$

then

$$d_i^* = \tau_i \quad (7)$$

where τ_i is defined as:

$$W_i(0, \tau_i) = \int_0^{\tau_i} r_i(t) dt = \sigma_i \quad (8)$$

(ii) If (6) does not hold for session i then it starts with an accumulating phase, for which there exists a $j \in \{1, \ldots, N\} \backslash \{i\}$ such that

$$\forall t < t_{L(j)} \ : \ r_i(t) < \rho_i$$

and

$$\forall t \geq t_{L(j)} \ : \ r_i(t) \geq \rho_i \ ;$$

in this case

$$d_i^* = t_{L(j)} - \frac{W_i(0, t_{L(j)}) - \sigma_i}{\rho_i}$$

III. Optimal Call Admission Control Algorithm

The Connection Admission Control (CAC) for GPS scheduling used in the literature to provide deterministic delay guarantees for sessions is based on the delay bound (3). In this simple CAC, a connection i is allotted a bandwidth of $\max(\frac{\sigma_i}{d_i}, \rho_i)$ where d_i is the delay guarantee asked for. The nice part about this CAC is that it completely eliminates interference among sessions, as if there were firewalls in between those individually reserved bandwidth channels. Thus, irrespective of the arrival patterns of other sessions, connection i will be assured a bandwidth of at least $\max(\frac{\sigma_i}{d_i}, \rho_i)$. An incoming session i is refused if this bandwidth is not available at the server. However, as we had seen in Section II-C, such an allocation could result in the actual worst-case delay being far less than the tolerable delay d_i, and thus leading to a waste of the allotted bandwidth.

The basic idea of the Optimal CAC was inspired by the method to obtain tight delay bounds in Theorem 1. The algorithm takes advantage of the backlog clearing times of some sessions in allotting bandwidth for a connection i, to meet its delay

requirement, d_i. Our goal is to achieve both the constrained delay requirements of the connections and to have some extent of isolation among the sessions, while at the same time preserving statistical multiplexing advantages.

The algorithm runs iteratively where at the beginning of each iteration the rates of the sessions are calculated to satisfy their delay bounds. Then, the backlog clearing times of these sessions are calculated. The session with the minimum backlog clearing time is identified, and its rate is kept fixed for the subsequent iterations. The algorithm proceeds with the next iteration and recalculates the rates of the remaining sessions taking into account the most recent backlog clearing time (determined in the previous iteration).

In the rest of the section, we first give an exact description of the Optimal CAC algorithm and then explain its major functions.

A. The Algorithm

The CAC algorithm determines the session rates needed to support their delay guarantees in an *All-Greedy Regime*(Def. 3). It is proved in Theorem 3 of [1] that for a session i the worst-case delay, d_i^* and the worst case backlog, q_i^* are both achieved in the All-Greedy Regime. Thus, the rates calculated in this scenario will satisfy the delay bounds of all the sessions under any other arrival pattern too.

Before listing the steps of the algorithm we introduce the concept of *level*. *"level"* is an attribute that is associated with every session and it indicates when the backlog of the session may begin service. The motivation for *level* is as follows: Consider a situation where the required delay guarantee of a session i is large. In this case, it may be possible to allocate an initial rate of zero to the session i without violating its delay guarantee; this relies on the fact that after some time, session i can get a positive service rate from the work-conserving server as other sessions complete their backlogged periods. In this situation, session i is said to belong to level 2 initially, indicating that it receives a positive service rate only after all the sessions at level 1 have cleared their backlogs. In general, sessions at a level n receive positive service rates only after all the sessions at levels $1, 2, \ldots, (n-1)$ have cleared their backlogs.

The following Lemmas and Theorems form the basis of the sessions' rate calculations in the algorithm. The proofs have been omitted due to lack of space. The readers are referred to [13] for the proofs.

Lemma 1: If for a session i, $\frac{\sigma_i}{d_i} \geq \rho_i$ and $d_i^* \leq d_i$ then,
(i) The service rate of session i is $\geq \rho_i$, when the last bit of the burst is getting served.
(ii) The maximum delay is experienced by the last bit of the burst.

Lemma 2: If for a session i, $\frac{\sigma_i}{d_i} < \rho_i$ and $d_i^* \leq d_i$ then neither (i) nor (ii) of Lemma 1 is necessarily true.

The following Lemma gives a method to calculate the guaranteed rate for a session j such that the last bit of the initial burst experiences the delay d_j.

Lemma 3: Let $t_{L(1)}, t_{L(2)}, \ldots, t_{L(i-1)}$ be the sessions backlog clearing times which are $< d_j$. Then, the rate r_j needed in order for the last bit of the initial burst to experience the delay d_j is, $r_j =$

$$\frac{\sigma_j}{\sum_{k=1}^{i-1}(t_{L(k)} - t_{L(k-1)})\frac{C-\sum_{m=1}^{k-1}\rho_{L(m)}}{C-\sum_{m=1}^{k-1}r_{L(m)}} + (d_j - t_{L(i-1)})A} \tag{9}$$

where the term 'A' = $\frac{C-\sum_{m=1}^{i-1}\rho_{L(m)}}{C-\sum_{m=1}^{i-1}r_{L(m)}}$

The following Corollary follows easily from Lemma 1 and Lemma 3.

Corollary 3.1: For sessions whose $\frac{\sigma_j}{d_j} \geq \rho_j$, the allocation of the above rate (9) results in the worst case delay, $d_j^* = d_j$.

Similarly, the following Corollary follows from Lemma 2.

Corollary 3.2: If $\frac{\sigma_j}{d_j} < \rho_j$ then with the rate allocation in (9), the worst case delay d_j^* need not be equal to d_j.

Lemma 4: Let $t_{L(1)}, t_{L(2)}, \ldots, t_{L(i-1)}$ be the session backlog clearing times and $d_j \leq t_{L(i-1)}$. Then, in order for the delay d_j to occur at the break-point $t_{L(i-1)}$, the rate needed is :

$$r_j = \frac{\rho_j(t_{L(i-1)} - d_j) + \sigma_j}{\sum_{k=1}^{i-1}(t_{L(k)} - t_{L(k-1)})\left(\frac{C-\sum_{m=1}^{k-1}\rho_{L(m)}}{C-\sum_{m=1}^{k-1}r_{L(m)}}\right)} \tag{10}$$

The following Corollary follows easily from Lemma 1.

Corollary 4.1: For sessions with $\frac{\sigma_j}{d_j} \geq \rho_j$ the above allocation results in the worst-case delay $d_j^* = d_j$ only if $d_j = t_{L(i-1)}$.

The following Corollary follows easily from Theorem 1(ii):

Corollary 4.2: For sessions with $\frac{\sigma_j}{d_j} < \rho_j$ the above allocation results in the worst-case delay $d_j^* = d_j$ only if $r_j^{(i-1)} < \rho_j$ and $r_j^{(i)} \geq \rho_j$.

***Algorithm**:*

***Step 1**: Initialization*
$i = 0$ *Iteration Variable*
$level = 1$ *An attribute of a session indicating when the backlog of the session may begin service*
The following sets are defined:
$B1 = \emptyset$ *$B1$ will consist of those sessions for which the service rate is greater than their average rate when the last bit of the burst is being served*
$B2 = \emptyset$ *$B2$ will consist of those sessions for which the service rate is not necessarily greater than the average rate when the last bit of the burst is being served*
$P = \emptyset$ *P will consist of those sessions for which the rates have been determined but not the backlog finish times*
$H = \emptyset$ *H will consist of sessions at a particular level for which the rates and the backlog finish times are both determined*
$H_{main} = \emptyset$ *H_{main} will consist of the sessions included in the set H at different levels *

***Step 2**: Classification of the sessions*
$B1 = \{session\ j \mid \frac{\sigma_j}{d_j} \geq \rho_j\}$
$B2 = \{session\ j \mid \frac{\sigma_j}{d_j} < \rho_j\}$

***Step 3**: Beginning of an Iteration*

$i = i + 1$

***Step 4**: Rate Calculation for Sessions in B1 and B2*

(a) Sessions $\in B1$:

$$\forall j \in B1,\ r_j = \frac{\sigma_j}{\sum_{k=1}^{i-1}\left((t_{L(k)} - t_{L(k-1)})\frac{C - \sum_{m=1}^{k-1}\rho_{L(m)}}{C - \sum_{m=1}^{k-1} r_{L(m)}}\right) + (d_j - t_{L(i-1)})A}$$

where term 'A' = $\frac{C - \sum_{m=1}^{i-1}\rho_{L(m)}}{C - \sum_{m=1}^{i-1} r_{L(m)}}$

(b) Sessions $\in B2$:

$if\ (\ i = 1\ )$

$r_j = \rho_j$

$else\ if\ (\ d_j \leq t_{L(i-1)}\ )$

{

$r_j = \frac{\rho_j(t_{L(i-1)} - d_j) + \sigma_j}{\sum_{k=1}^{i-1}(t_{L(k)} - t_{L(k-1)})\left(\frac{C - \sum_{m=1}^{k-1}\rho_{L(m)}}{C - \sum_{m=1}^{k-1} r_{L(m)}}\right)}$

$if\ (r_j^{(i-1)} < \rho_j\ and\ r_j^{(i)} \geq \rho_j)$

{

$P = P \cup \{j\}$

$B2 = B2 - \{j\}$

}

$else\ if\ (r_j^{(i-1)} \geq \rho_j)$

{

$r_j = \frac{\rho_j}{\left(\frac{C - \sum_{k=1}^{i-2}\rho_{L(k)}}{C - \sum_{k=1}^{i-2} r_{L(k)}}\right)}$

$P = P \cup \{j\}$;

$B2 = B2 - \{j\}$;

}

$else\ if\ (r_j^{(i)} < \rho_j)$

$r_j = \frac{\rho_j}{\left(\frac{C - \sum_{k=1}^{i-1}\rho_{L(k)}}{C - \sum_{k=1}^{i-1} r_{L(k)}}\right)}$

}

$else\ if\ (\ d_j > t_{L(i-1)}\ )$

{

$$r_j = \frac{\sigma_j}{\sum_{k=1}^{i-1}\left((t_{L(k)} - t_{L(k-1)})\frac{C - \sum_{m=1}^{k-1}\rho_{L(m)}}{C - \sum_{m=1}^{k-1} r_{L(m)}}\right) + (d_j - t_{L(i-1)})\frac{C - \sum_{m=1}^{i-1}\rho_{L(m)}}{C - \sum_{m=1}^{i-1} r_{L(m)}}}$$

$if\ (r_j^{(i)} \geq \rho_j)$

{

$B1 = B1 \cup \{j\}$

$B2 = B2 - \{j\}$

}

$else$

$r_j = \frac{\rho_j}{\left(\frac{C - \sum_{k=1}^{i-1}\rho_{L(k)}}{C - \sum_{k=1}^{i-1} r_{L(k)}}\right)}$

}

***Step 5**: Calculation of the backlog clearing times for sessions $\in B1,\ B2\ and\ P$*

$t_{j,i} = \frac{S_{j,i}}{r_j^{(i)} - \rho_j}\ \forall\ j \in B1,\ B2\ and\ P$

$where\ S_{j,i} = \sigma_j + \sum_{m=1}^{i-1}[r_j^{(m)}(t_{L(m-1)} - t_{L(m)})] + r_j^{(i)}(t_{L(i-1)})$

$and\ r_j^{(m)} = r_j\ (\frac{C - \sum_{k=1}^{m-1}\rho_{L(k)}}{C - \sum_{k=1}^{m-1} r_{L(k)}})$

***Step 6**: Choose the session with the minimum backlog time and include it in H*

$L(i) = session\ j\ such\ that:$

$t_{L(i)} = min\{t_{j,i} \mid 0 < t_{j,i} < \infty;\ j \in B1,\ B2,\ P\}$

$if\ (\ no\ such\ t_{L(i)}\ exists)$

$then\ L(i) = 0;$

$else$

{

$H = H \cup \{L(i)\}$

$Delete\ L(i)\ from\ the\ set\ (\ B1,\ B2,\ or\ P\ )$

}

***Step 7**: Identify sessions $\in$ B1 whose delay is less than $t_{L(i)}$*

$P = P \cup \{\ session\ j \mid j \in B1\ and\ d_j \leq t_{L(i)}\ \}$

$B1 = B1 - \{session\ j \mid j \in B1\ and\ d_j \leq t_{L(i)}\}$

***Step 8**: Check Schedulability Condition*

$(a)\quad if\ (\ \sum_{j \in H} r_j > C\ )$

{

$The\ sessions\ are\ not\ schedulable$

$STOP$

}

$(b)\quad else\ if\ (\ \sum_{j \in H} r_j = C\ )$

{

$(i)\ if\ (\ P \neq \emptyset\ OR\ B2\ has\ any\ session\ d_j \leq t_{L(i)}\ )$

{

$The\ sessions\ are\ not\ schedulable$

$STOP$

}

$(ii)\quad else$

{

$C = C - \sum_{j \in H} \rho_j$

$Include\ all\ sessions\ that \in H\ into\ H_{main}$

$level++$

$*\ Reinitialize\ H\ and\ i\ *$

$i = 0$

$H = \emptyset$

$Update\ the\ level\ of\ the\ remaining\ sessions\ in\ B1\ and\ B2\ to\ 'level'$

}

}

***Step 9**: Check for the Termination Conditions of the Loop begun in Step 3*

if ((Number of sessions in H_{main} < Total number of sessions) AND ($L(i) \neq 0$))

GOTO Step 3

***Step 10**: Check for the Final Schedulability Conditions*

if $(\sum_j r_j > C)\ j \in (B1 \cup B2 \cup P \cup H)$
{
The sessions are not schedulable
$STOP$
}

***Step 11**: Termination Condition of the Algorithm*
$if(C - \sum_j r_j < \epsilon)$
$STOP$
$else$
{
C = sum of rates of sessions
GOTO Step 1 and re-execute the algorithm with the reduced C
}

Before explaining the steps of the algorithm we state the following Lemma, the proof of which is in [13].

Lemma 5: For a fixed value of C, the rates of sessions in a particular iteration $i+1$ $(i \geq 1)$, are less than or equal to their corresponding rates in iteration i.

The basic idea of the algorithm is to allocate rates to sessions such that they do not over-achieve their delays. It does so by taking into account the backlog clearing times of the other sessions. The steps in the algorithm are elucidated below:

The N sessions at the GPS Server are assumed to satisfy the stability criterion, that is $\sum_{j=1}^{N} \rho_j < C$, where C is the server capacity. Every session j is characterized by six attributes: They are:
(1) Burstiness (σ_j)
(2) Average Rate (ρ_j)
(3) Delay Guarantee (d_j)
(4) Allotted Rate (r_j)
(5) Backlogged Time and
(6) level
In *Step 2*, a session j is classified as belonging to set $B1$ or $B2$ depending on whether $\frac{\sigma_j}{d_j} \geq \rho_j$ or $\frac{\sigma_j}{d_j} < \rho_j$, respectively. As we had seen in Lemma 1, the sessions in B1 necessarily have their service rates $\geq$ average rates when the last bit of the burst is being served. By Lemma 2 this is not necessarily true for the sessions in $B2$. As the algorithm proceeds the sessions in $B2$ which need a service rate $\geq$ average rate during the service of the last bit of the burst, are separated out and put into $B1$. This classification is necessary since the method of rate calculation in order not to over-achieve the worst case delay, is different for a session $\in B1$ and for that $\in B2$.

Step 4(a) calculates the rates of the sessions $\in B1$ using Lemma 3. The sessions in B1 must have their service rates $>$ their average rates when the last bit of the burst is being served (Lemma 1). Thus by Corollary 3.1 we know that for such a session j the rate allocation calculated in this step results in the worst case delay, d_j^* = required delay, d_j.

Step 4(b) calculates the rates of the sessions $\in B2$ in three different cases.

The first case is executed only for the first iteration $(i = 1)$. In this case a session $\in B2$ is allotted a rate equal to its average rate. At this stage the backlog finish time of no session $(\in (B1 \cup B2))$ has been fixed, and hence such an allocation is necessary in order for a session $\in B2$, not to violate its delay bound.

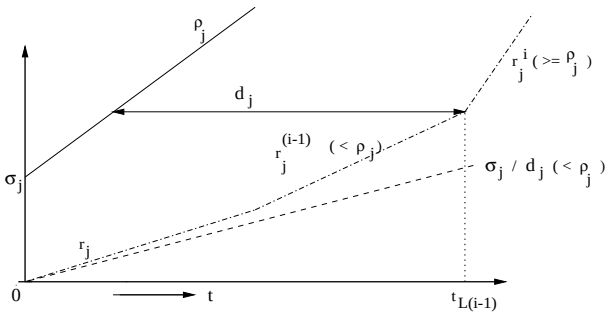

Fig.2(a) The case where the worst case delay is experienced at the break-point, $t_{L(i-1)}$.

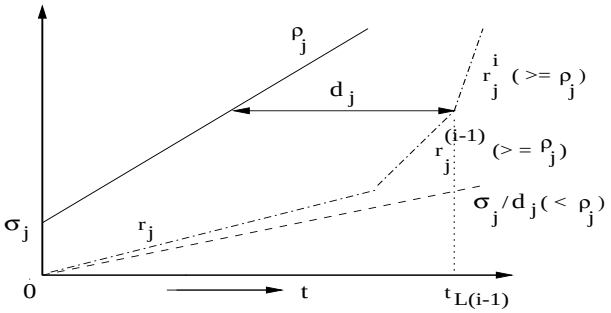

Fig.2(b) A case where the delay d_j at the break-point, $t_{L(i-1)}$ is not the worst-case delay.

The second and the third cases are valid only if, at least one session's backlog finish time is fixed $(i.e.\ i \geq 2)$.

The second case is when $d_j \leq t_{L(i-1)}$. Recall that $t_{L(i-1)}$ is the time instant when session $L(i-1)$ finishes its backlog (it is $(i-1)^{th}$ session to do so). In this case the rates are calculated using Lemma 4. However such a rate allocation need not always result in the worst case delay d_j^* being equal to the required delay, d_j. d_j^* is equal to d_j only if a certain condition is satisfied by the rates $r_j^{(i-1)}$ and $r_j^{(i)}$. There are three possible conditions that arise and are checked in the subsequent statements of $Step\ 4(b)$.

The three conditions are illustrated in Fig.2. Fig.2(a) illustrates the first condition when the session service rate before the break-point $(t_{L(i-1)})$, $r_j^{(i-1)} < \rho_j$ while the service rate after the break-point, $r_j^{(i)} \geq \rho_j$. By Corollary 4.2, we know that the worst-case delay d_j^* acheived in such a scenario is equal to the required delay d_j. In this case the rate r_j of session j cannot be further reduced in the subsequent iterations without violating the delay bound. Hence this session is deleted from $B2$ and included in the set P (P is the set of sessions whose rate allocations are frozen but their backlog clearing times are not).

Fig.2(b) illustrates the second condition i.e. when the rate before the breakpoint, $r_j^{(i-1)} \geq \rho_j$. In this case the worst case delay is not d_j and intuitively, the session j does not benefit from the backlog clearing time of session $L(i-1)$, in terms of reducing its rate allocation. The rate of such a session is updated to $\frac{\rho_j}{\left(\frac{C - \sum_{k=1}^{i-2} \rho_{L(k)}}{C - \sum_{k=1}^{i-2} r_{L(k)}}\right)}$ and included in the set P.

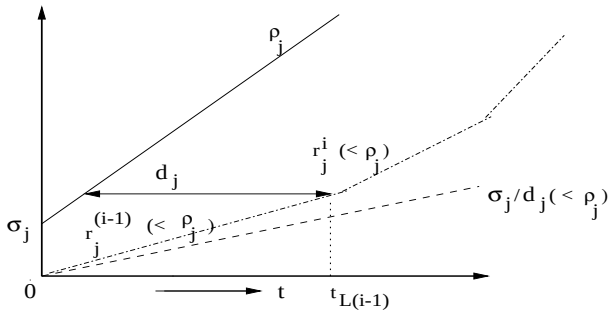

Fig.2(c) A case where the delay d_j at the break-point, $t_{L(i-1)}$ is not the worst-case delay.

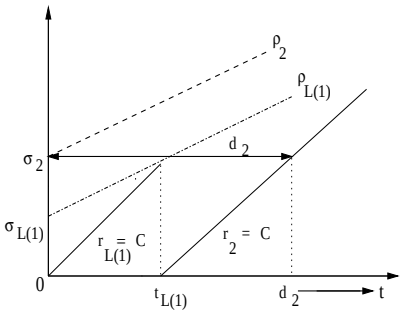

Fig.3 Delay guarantees are satisfied with sessions at different levels.

The third condition illustrated in Fig.2(c) is when the session service rate after the break-point, $r_j^{(i)} < \rho_j$. In this case the worst-case delay is achieved beyond $t_{L(i-1)}$, so the rate of the session is updated to $\frac{\rho_j}{\left(\frac{C-\sum_{k=1}^{i-1}\rho_{L(k)}}{C-\sum_{k=1}^{i-1}r_{L(k)}}\right)}$. In such a case the session over-achieves its delay, so it is allowed to remain in $B2$ for a possible reduction in the rate r_j in the subsequent iterations.

The third case is when $d_j > t_{L(i-1)}$. In this case the rates are calculated using Lemma 3. By Corollary 3.2 we know that the worst case delay d_j^* is not necessarily equal to d_j. d_j^* is equal to d_j only if the session j service rate when the last bit of the burst is being served is $\geq \rho_j$, in which case this session is put into the set $B1$; otherwise the rate of the session is updated such that it attains its average rate after $t_{L(i-1)}$ and continues to remain in $B2$.

As proved in Lemma 5, the sessions which remain in $B2$ at the end of *Step 4(b)* in a particular iteration would possibly be allotted lesser rates in the subsequent iterations.

Step 5 computes the backlog finish times for the sessions $\in B1,\ B2$ and P using (5) of Section II.

Step 6 chooses the session with the minimum backlog time and includes it in set H (H is the set of sessions whose allotted rates and the backlog finsh times are frozen). Note that it is possible that the backlog clearing times of all the sessions in $(B1 \cup B2 \cup P)$ are equal to ∞. In such a scenario $L(i) = 0$. If in a particular iteration i, $L(i) = 0$, subsequent iterations of the algorithm will not yield better rate allocations (i.e. lesser rate allocations) for the remaining sessions $\in (B1 \cup B2)$. Hence $L(i) = 0$ is a loop terminating condition that is checked further ahead.

Step 7 identifies the sessions $\in B1$ which have their delay requirements $\leq t_{L(i)}$. The rates r_j of such sessions have to be frozen since they do not benefit from the backlog clearing time of $L(i)$. For the remaining sessions in $B1$, subsequent iterations will yield a lesser rate allocation as proved in Lemma 5.

Step 8 checks the schedulability conditions at the end of one iteration. Note that at this step, even if $\sum_{j \in H} r_j = C$, it could still be possible to accommodate the remaining sessions depending on their delay requirements. As mentioned before, these sessions would belong to the next higher level, and their service is begun only after the sessions at the preceding level complete their backlogs (see Fig.3). In the example in Fig.3 though the sum of the rates at the end of the first iteration is $= C$, session 2 can still be supported. This is because session 2 can begin clearing its backlog at $t_{L(1)}$ and still satisfy its delay bound d_2. Thus session 2 has a level = 2.
The values of $C,\ i$ and H are re-initialized and the algorithm is continued.

Step 9 checks the termination conditions of the loop. The loop terminates either if all the sessions have been included in H_{main} or if there is no session $\in (B1 \cup B2 \cup P)$ which has a finite backlog finish time $(i.e.\ L(i) = 0)$.

Step 11 checks for the termination condition of the algorithm. The algorithm initially begins with C = server capacity. At the end of the first iteration if the sum of session rates, $\sum_{j \in N} r_j < C$ then the remaining bandwidth $C - \sum_{j \in N} r_j$ is assumed to be given to a "dummy" session, with parameters: $\sigma_d = \infty, \rho_d = 0$. This essentially means that the "dummy" session also gets a share of the excess bandwidth released when any session j $(j \in N)$ completes its backlog period. However, our aim is to find the minimum rates of sessions $1, \ldots, N$ without any assumption of the "dummy" session. In other words, we have to find the minimum capacity C that is required to support the N sessions , when the excess released bandwidth in any iteration i of sessions $L(1), \ldots, L(i-1)$ is distributed among the remaining sessions, $N - \{\cup_{k=1}^{i-1} L(k)\}$ only.

Thus at the end of execution of the algorithm for the first time, if the difference between the server capacity, C and the sum of session rates is $> \epsilon$ (ϵ is a small chosen quantity), the capacity C is updated to the sum of the session rates. In general, let C_n $(n \geq 1)$ denote the minimum capacity that the n^{th} execution of the algorithm started with. Then, if at the end of the n^{th} execution of the algorithm, $C_n - \sum_{j \in N} r_j > \epsilon$, the capacity that the $(n+1)^{th}$ execution starts with is C_{n+1} = sum of session rates at the end of the n^{th} execution. The following Lemma proves the correctness of the above method, the proof of which is in [13].

Lemma 6: If at the beginning of the n^{th} $(n \geq 2)$ execution of the algorithm, the server capacity C_n = sum of session rates at the end of $(n-1)^{th}$ execution of the algorithm, then:
(i) The sum of session rates at the end of the n^{th} execution of the algorithm is $\leq C_n$.
(ii) The algorithm terminates after a finite number of execu-

tions.

B. Properties

B.1 Optimal Bandwidth Allocation

The algorithm is claimed to be optimal on the basis of the following:
(i) Optimality means that the given set of connections cannot be supported by a smaller bandwidth allocation if their delay guarantees are to be met,
(ii) In the algorithm, the initial rates of each session are decreased to such a point that further decreases would violate the delay guarantees.

B.2 Bandwidth and Delay De-coupled System

Rate-Proportional Servers (RPS) form a large sub-class of GPS-based schedulers. They have a rate-proportional allocation strategy in which the guaranteed rate of a connection needs to be no less than the connection's average rate and the delay bound for the connection is inversely proportional to the guaranteed rate. In other words the guaranteed rate for a connection i is $\max(\frac{\sigma_i}{d_i}, \rho_i)$. This introduces a coupling between the end-to-end delay bound and the bandwidth provided to each connection. Thus, in order for a connection to get a low delay bound, a high bandwidth channel needs to be allotted. This results in a waste of resources when the low delay connection also has a low throughput.

GPS-based schedulers with a *General Resource Assignment* or a *Non-Rate Proportional Resource Assignments* do not have such a restriction [14]. The CAC in this paper does such a Non-Rate Proportional Resource Assignment which results in efficient network utilization.

B.3 Statistical Multiplexing Effect

Notice that, the fewer the number of sessions at the node, the greater is the rate needed to be allotted for a session i to satisfy its delay requirement. For example, if session i is the only session at the server then it has to be allotted a rate of $\max(\frac{\sigma_i}{d_i}, \rho_i)$. This rate will possibly reduce as the number of sessions at the node increases and session i gets a share of the released excess bandwidth after some sessions clear their backlogs.

B.4 Supports Best-Effort Traffic

In case there is Best-Effort traffic that has to be supported along with N guaranteed-delay sessions, then the algorithm is executed only once. At the end of the first execution, if $\sum_{j\in N} r_j < C$, the remaining capacity $C - \sum_{j\in N} r_j$ can be allotted to the Best-Effort traffic without violating the QoS requirements of the guaranteed-delay sessions.

IV. Numerical Results

The numerical results are presented in two parts. The first part illustrates how the algorithm allots bandwidth optimally to guarantee the delay requirements of the connections, in different scenarios. The second part demonstrates the *Non-Rate Proportional* bandwidth allocation strategy of the Optimal CAC and its advantage over the *Rate Proportional* bandwidth allocation. All the numerical results could not be included due to lack of space.

Table I

Session	σ	ρ	d	d^*	Backlog Time	Allotted Rate	level
1	100	20	2	2	3.33	50	1
2	150	10	3	3	3.57	50	1
3	100	10	7	7	8.788	29.167	2
4	100	5	30	27.62	∞	3.04	2

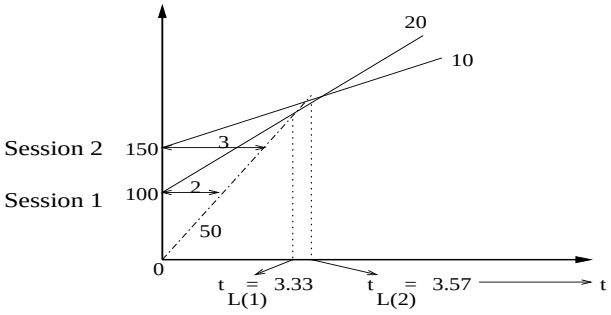

Fig.4(a) Sessions 1 and 2 at level 1.

A. Computing the Optimal Bandwidth to support the Connections' Delay Guarantees

Fig.4 illustrates a case where the sessions belong to different $levels$ and satisfy their delay requirements. The server capacity is 100. Sessions 1 and 2 belong to level 1 while sessions 3 and 4 belong to level 2. The sessions' parameters are given in Table I. These sessions are not schedulable under the Rate-Proportional CAC. The reader is referred to [13] for the remaining results of this section.

B. General Resource Allocation vs. Rate Proportional Resource Allocation

In this Section we demonstrate the greater efficiency of the *General Resource Allocation* in the GPS scheduler as compared to the *Rate Proportional Allocation*. The server capacity C is fixed at 1000. As mentioned before, we consider leaky-bucket constrained sources.

The comparison is carried out as follows. There are three types of traffic (Type 1, 2 and 3) whose input parameters and the delay requirements are fixed as shown in Table II.

In Fig.5(a) the number of Type-1 connections is fixed while we compute the maximum number of Type-2 and Type-3 connections that can be admitted at the server, using the *Rate Proportional CAC* and *Optimal Non-rate Proportional CAC*. In

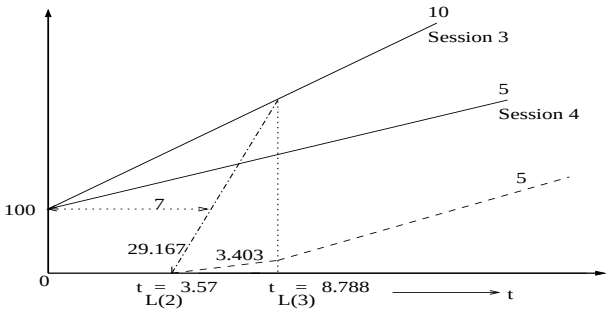

Fig.4(b) Sessions 3 and 4 at level 2.

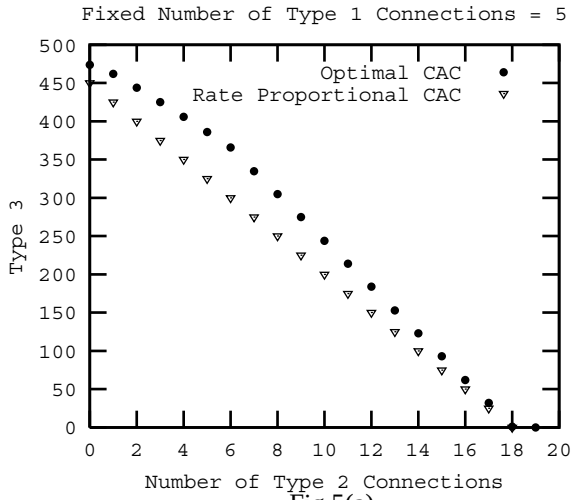

Fig.5(a)

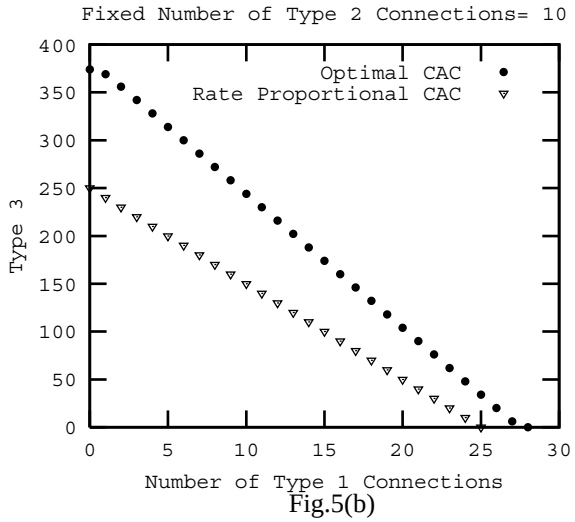

Fig.5(b)

Fig.5(b), the number of Type 2 connections is fixed.

In all the three cases it is clear that the General Resource Allocation results in a higher network utilization than the Rate-Proportional Allocation. This difference will be even more pronounced as the diversity of the traffic mix increases.

V. Summary and Future Work

In this paper we have presented an Optimal Call Admission Control algorithm for Generalized Processor Sharing schedulers. The CACs presented in the literature allot bandwidth according to the *Rate Proportional* resource allocation and are based on loose delay bounds. These CACs, though simple, would result in a gross over-allocation of bandwidth. There have been efforts in the literature for a Non-Rate Proportional CAC [10]; however, these do not consider an optimal allocation and could still result in an over-allocation. The Optimal CAC in this paper does a *Non-Rate Proportional Resource* allocation and overcomes the above limitations. Its fundamental merit arises from the fact that it takes into account the bandwidth released by the sessions which have completed their backlogs, while calculating the sessions' rates. The algorithm allots minimum rates to the sessions such that their delay bounds are not violated. It is shown in the Numerical Results Section that the Optimal CAC can accept significantly more number of connections than the CAC based on *Rate Proportional Allocation*, and hence results in better network utilization.

Table II

Session	σ	ρ	Required Delay (d)
1	100	10	5
2	100	25	2
3	9	2	7.6

In this paper we have provided an important insight into *Optimal Resource Allocation* in a GPS scheduler. Our current research focuses on an *Optimal Non-rate Proportional* resource allocation in a *network* of GPS schedulers to meet a connection's end-to-end delay bound. General resource assignment in other scheduling disciplines like the Virtual Clock can be addressed in future research.

Finally, it would be interesting to compare the schedulable regions of GPS under General Resource assignment and that of Earliest Deadline First, which is known to be the optimal scheduling discipline in literature.

References

[1] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The single-node case," *IEEE/ACM Trans. on Networking*, vol. 1, no. 3, pp. 344–357, Jun. 1993.

[2] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The multiple-node case," *IEEE/ACM Trans. on Networking*, vol. 2, no. 2, pp. 137–150, Apr. 1994.

[3] A. Demers, S. Keshav, and S. Shenkar, "Analysis and simulation of a fair queueing algorithm," *Internetworking Research and Experience*, vol. 1, 1990.

[4] Z.L. Zhang, Z. Liu, and D. Towsley, "Closed-form deterministic end-to-end performance bounds for the generalized processor sharing scheduling discipline," *Journal of Combinatorial Optimization (Special Issue on Scheduling)*, In Print.

[5] O. Yaron and M. Sidi, "Generalized processor sharing networks with exponentially bounded burstiness arrivals," in *Proceedings of IEEE INFOCOM'94*, Jun. 1994, pp. 628–634.

[6] Z.L. Zhang, D. Towsley, and J. Kurose, "Statistical analysis of the generalized processor sharing scheduling discipline," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 6, pp. 1071–1080, Aug. 1995.

[7] G. de Veciana and G. Kesidis, "Bandwidth allocation for multiple qualities of service using generalized processor sharing," *IEEE Trans. on Information Theory*, vol. 42, no. 1, 1996.

[8] J. Kurose Z.L. Zhang, Z. Liu and D. Towsley, "Call admission control schemes under generalized processor sharing scheduling," *Telecommunication Systems*, In Print.

[9] K. Kumaran *et al*, "Novel techniques for the design and control of generalized processor sharing schedulers for multiple qos classes," in *Proceedings of IEEE INFOCOM*, 2000.

[10] R. Szabó *et al*, "Call admission control in generalized processor sharing schedulers using non-rate proportional weighting of sessions," in *Proceedings of IEEE INFOCOM'99*, 1999.

[11] R.Cruz, "A calculus for network delay, part i: Network elements in isolation," *IEEE Trans. on Information Theory*, vol. 37, no. 1, pp. 114–131, Jan. 1991.

[12] R. Szabó *et al*, "Non-rate proportional weighting of generalized processor sharing schedulers," in *Proceedings of IEEE GLOBECOM'99*, 1999.

[13] D. Nandita, J. Kuri, and H.S. Jamadagni, "Optimal call admission control in generalized processor sharing schedulers," Tech. Rep. TR-00-02, Center for Electronics Design and Technology, Indian Institute of Science, http://shravana.cedt.iisc.ernet.in/∼dnandita/, Jun. 2000.

[14] A. K. Parekh, *A Generalized processor sharing approach to flow control in integrated services networks*, Ph.D. thesis, Dep. Elec. Eng. Comput. Sci., M.I.T., 1992.