A GENERALIZED PROCESSOR SHARING APPROACH TO FLOW CONTROL IN INTEGRATED SERVICES NETWORKS—THE SINGLE NODE CASE

ABHAY K. PAREKH and ROBERT G. GALLAGER

Laboratory for Information and Decision Systems
Massachusetts Institute of Technology
parekh,gallager@lids.mit.edu

Abstract The problem of allocating network resources to the users of an integrated services network is investigated in the context of rate based flow control. The network is assumed to be a virtual circuit, connection-based packet network. We propose a highly flexible and efficient multiplexing scheme called Generalized Processor Sharing (GPS) that allows the network to make worst-case performance guarantees. A practical packet-by-packet service discipline that closely approximates Generalized Processor Sharing is also presented. This allows us to relate performance results for GPS to the packet-by-packet scheme in a precise manner.

A single server GPS system is analyzed exactly, and tight bounds on worst-case packet delay, output burstiness and backlog are derived for each session, when the sources are constrained by leaky buckets. The analysis yields a simple resource assignment scheme that allows the server to make worst case delay and rate guarantees to every session in the system. Extensions of this work to arbitrary topology networks are also discussed.

1 Introduction

This paper focuses on a central problem in the control of congestion in high speed integrated services networks. Traditionally, the flexibility of data networks has been traded off with the performance guarantees given to the users. For example, the telephone network provides good performance guarantees but poor flexibility, while most packet switched networks are more flexible, but only provide marginal performance guarantees. Integrated services networks must carry a wide range of traffic types and still be able to provide performance guarantees to real-time sessions such as voice and video. We will investigate an approach to reconcile these apparently conflicting requirements when the short-term demand for link usage frequently exceeds the usable capacity.

We propose the use of a packet service discipline at the nodes of the network that is based on a multiplexing scheme called generalized processor sharing. This service discipline is combined with leaky bucket rate admission control to provide flexible, efficient and fair use of the links. A major part of our work is to analyze networks of arbitrary topology using these specialized servers, and to show how the analysis leads to implementable schemes for guaranteeing worst-case packet delay. In this paper, however, we will restrict our attention to sessions at a single node, and postpone the analysis of arbitrary topologies to [10].

The analysis will concentrate on providing guarantees on throughput and worst-case packet delay. While packet delay in the network can be expressed as the sum of the processing queueing, transmission and propagation delays, we will focus exclusively on how to limit *queueing* delay.

Our approach can be described as a strategy for rate-based flow control. Under rate-based schemes, a source's traffic is parametrized by a set of statistics such as average rate, maximum rate, burstiness etc., and is assigned a vector of values corresponding to these parameters. The user also requests a certain quality of service, that might be characterized, for example, by tolerance to worst-case or average delay. The network checks to see if a new source can be accommodated, and if so, it takes actions (such as reserving transmission links or switching capacity) to ensure the quality of service desired. Once a source begins sending traffic, the network ensures that the agreed upon values of traffic parameters are not violated.

We will assume that rate admission control is done through *leaky buckets* [12]. An important advantage of using leaky buckets is that this allows one to separate the packet delay into two components–delay in the leaky bucket and delay in the network. The first of these components is *independent* of the other active sessions and can be estimated by the user, if the statistical characterization of the incoming data is sufficiently simple (See Section 6.3 of [1] for an example).

CH3133-6/92/0000-0915 $3.00 © 1992 IEEE

The traffic entering the network has been "shaped" by the leaky bucket in a manner that can be succinctly characterized (we will do this in Section 6), and so the network can upper bound the second component of packet delay through this characterization. This upper bound is independent of the statistics of the incoming data, which is helpful in the usual case where these statistics are either complex or unknown. From this point on, we will not consider the delay in the leaky bucket.

2 An Outline

Generalized Processor Sharing (GPS) is defined and explained in Section 3. In Section 4 we present a packet-based scheme, PGPS, and show that it closely approximates GPS. Results obtained in this section allow us to translate session delay and buffer requirement bounds derived for a GPS server system to a PGPS server system. We propose a virtual clock implementation of PGPS in the next subsection. Then PGPS is compared to some other multiplexing schemes.

Having established PGPS as a desirable service discipline scheme we turn our attention to the rate enforcement function in Section 6. The leaky bucket is described and proposed as a desirable strategy for admission control. We then proceed with an analysis, in Sections 7 and Section 8, of a single GPS server system in which the sessions are constrained by leaky buckets. In Section 9 we outline an algorithm for providing performance guarantees to a new user without violating guarantees made to the existing sessions of the system. Conclusions are in Section 10.

3 GPS Multiplexing

The choice of an appropriate service discipline at the nodes of the network is key to providing effective flow control. A good scheme should allow the network to treat users differently, in accordance with their desired quality of service. However, this *flexibility* should not compromise the *fairness* of the scheme, i.e. a few classes of users should not be able to degrade service to other classes, to the extent that performance guarantees are violated. Also, if one assumes that the demand for high bandwidth services is likely to keep pace with the increase in usable link bandwidth, time and frequency multiplexing are too wasteful of the network resources to be considered as candidate multiplexing disciplines. Finally, the service discipline must be *analyzable* so that performance guarantees can be made in the first place. We now present a flow-based multiplexing discipline called Generalized Processor Sharing that is efficient, flexible, fair and analyzable, and that therefore seems very appropriate for integrated services networks. However, it has the significant drawback of not transmitting packets as entities. In Section 4 we will present a packet-based multiplexing discipline that is an excellent approximation to GPS even when the packets are of variable length.

A Generalized Processor Sharing (GPS) server is work conserving and operates at a fixed rate r. It is characterized by positive real numbers $\phi_1, \phi_2, ..., \phi_N$. Let $S_i(\tau, t)$ be the amount of session i traffic served in an interval $[\tau, t]$. Then a GPS server is defined as one for which

$$\frac{S_i(\tau,t)}{S_j(\tau,t)} \geq \frac{\phi_i}{\phi_j}, \quad j = 1, 2, ..., N \tag{1}$$

for any session i that is backlogged in the interval $[\tau, t]$.

Summing over all sessions j:

$$S_i(\tau,t)\sum_j \phi_j \geq (t-\tau) r \phi_i$$

and session i is guaranteed a rate of

$$g_i = \frac{\phi_i}{\sum_j \phi_j} r. \tag{2}$$

GPS is an attractive multiplexing scheme for a number of reasons:

- Define ρ_i to be the session i average rate. Then as long as $\rho_i \leq g_i$, the session can be guaranteed a throughput of ρ_i, independent of the demands of the other sessions.
- The delay of an arriving session i bit can be bounded as a function of the session i queue length, independent of the queues and arrivals of the other sessions. Schemes such as FCFS, LCFS, and Strict Priority do not have this property.
- By varying the ϕ_i's we have the flexibility of treating the sessions in a variety of different ways. For example, when all the ϕ_i's are equal the system reduces to uniform processor sharing. As long as the combined average rate of the sessions is less than r, any assignment of positive ϕ_i's yields a stable system.

- It is possible to make worst-case network queueing delay *guarantees* when the sources are constrained by leaky buckets. We will present our results on this later. Thus GPS is particularly attractive for sessions sending real-time traffic such as voice and video.

Figure 3.1 illustrates generalized processor sharing. Variable length packets arrive from both sessions on infinite capacity links and appear as impulses to the system. For $i = 1, 2$, let $A_i(0,t)$ be the amount of session i traffic that arrives at the system in the interval $(0,t]$, and similarly, let $S_i(0,t)$ be the amount of session i traffic that is served in the interval $(0,t]$. We assume that the server works at rate 1.

When $\phi_1 = \phi_2$, and both sessions are backlogged,

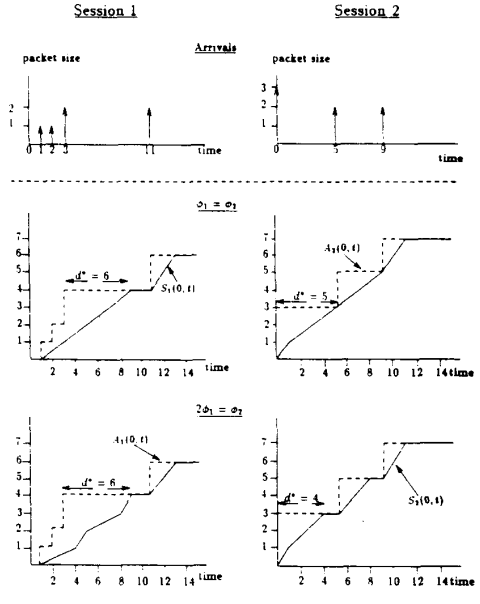

Figure 3.1: An example of generalized processor sharing.

they are each served at rate $\frac{1}{2}$ (eg. the interval $[1,6]$). When $2\phi_1 = \phi_2$, and both sessions are backlogged, session 1 is served at rate $\frac{1}{3}$ and session 2 at rate $\frac{2}{3}$. Notice how increasing the relative weight of ϕ_2 leads to better treatment of that session in terms of both backlog and delay. Also, notice that under both choices of ϕ_i, the system is empty at time 13 since the server is work conserving under GPS.

4 Packet-by-Packet GPS

A problem with GPS is that it is an idealized discipline that does not transmit packets as entities. It assumes that the server can serve multiple sessions simultaneously and that the traffic is infinitely divisible. In this section we propose a simple packet-by-packet transmission scheme that is an excellent approximation to GPS even when the packets are of variable length. Our idea is similar to the one used in [4] to simulate uniform processor sharing. *We will adopt the convention that a packet has arrived only after its last bit has arrived.*

Let F_p be the time at which packet p will depart (finish service) under generalized processor sharing. Then a very good approximation of GPS would be a work conserving scheme that serves packets in increasing order of F_p. (By work conserving we mean that the server is always busy when there are backlogged packets in the system.) Now suppose that the server is becomes free at time τ. The next packet to depart under GPS *may not have arrived* at time τ, and since the server has no knowledge of when this packet will arrive, there is no way for the server to be *both* work conserving and to serve the packets in increasing order of F_p. Now suppose the server becomes free at time τ, i.e. it has just finished transmitting a packet at time τ. The server picks the first packet that would complete service in the GPS simulation if no additional packets were to arrive after time τ. Let us call this scheme PGPS for *packet-by-packet* generalized processor sharing.

Notice that when $\phi_1 = \phi_2$ in the example of Figure 3.1, the first packet to complete service under GPS is the session 1 packet that arrives at time 1. However, the PGPS server is forced to begin serving the long session 2 packet at time 0, since there are no other packets in the system at that time. Thus the session 1 packet arriving at time 1 departs the system at time 4, i.e. 1 time unit later than it would depart under GPS.

A natural issue to examine at this point is how much later packets may depart the system under PGPS relative to GPS. First we present a useful property of GPS systems.

Lemma 1 *Let p and p' be packets in a GPS system at time τ and suppose that packet p completes service before packet p' if there are no arrivals after time τ. Then packet p will also complete service before packet p' for any pattern of arrivals after time τ.*

$\phi_i \frac{\partial V(t_j+\tau)}{\partial \tau}$. Thus, V can be interpreted as increasing at the marginal rate at which backlogged sessions receive service.

Now suppose that the k^{th} session i packet arrives at time a_i^k and has length L_i^k. Then denote the virtual times at which this packet begins and completes service as S_i^k and F_i^k respectively. Defining $F_i^0 = 0$ for all i, we have

$$\begin{aligned} S_i^k &= \max\{F_i^{k-1}, V(a_i^k)\} \\ F_i^k &= S_i^k + \frac{L_i^k}{\phi_i}. \end{aligned} \tag{6}$$

There are three attractive properties of the virtual time interpretation from the standpoint of implementation. First, the virtual time finishing times can be determined at the packet arrival time. Second, the packets are served in order of virtual time finishing time. Finally, we need only update virtual time when there are events in the GPS system. However, the price to paid for these advantages is some overhead in keeping track of the sets B_j, which is essential in the updating of virtual time:

Define Next(t) to be the *real* time at which the next packet will depart the GPS system after time t if there are no more arrivals after time t. Thus the next virtual time update after t will be performed at $Next(t)$ if there are no arrivals in the interval $[t, \text{Next}(t)]$. Now suppose a packet arrives at some time, t, and that the time of the event just prior to t is τ (if there is no prior event, i.e. if the packet is the first arrival in a busy period, then set $\tau = 0$). Then, since the set of busy sessions is fixed between events, $V(t)$ may be computed from (5), and the packet stamped with its virtual time finishing time. Next(t) and the new value of B are also computed.

Given this mechanism for updating virtual time, PGPS is defined as follows: When a packet arrives, virtual time is updated and the packet is stamped with its virtual time finishing time. The server is work conserving and serves packets in increasing order of time-stamp.

5 Comparing PGPS to other schemes

Under weighted round robin, every session i, has an integer weight, w_i associated with it. The server polls the sessions according a *precomputed sequence* in an attempt to serve session i at a rate of $\frac{w_i}{\sum_j w_j}$. If an empty buffer is encountered, the server moves to the next session in the order instantaneously. When an arriving session i packet just misses its slot in a frame it cannot be transmitted before the next session i slot. If the system is heavily loaded in the sense that almost every slot is utilized, the packet may have to wait almost N slot times to be served, where N is the number of sessions sharing the server. Since PGPS approximates GPS to within one packet transmission time regardless of the arrival patterns, it is immune to such effects. PGPS also handles variable length packets in a much more systematic fashion than does weighted round robin. However, if N or the packets sizes are small then it is possible to approximate GPS well by weighted round robin.

Zhang proposes an interesting scheme called virtual clock multiplexing [13]. Virtual clock multiplexing allows guaranteed rate and (average) delay for sessions independent of the behavior of other sessions. However, if a session produces a large burst of data, even while the system is lightly loaded, that session can be "punished" much later when the other sessions become active. Under PGPS the delay of a session i packet can be bounded in terms of the session i queue size seen by that packet upon arrival, whereas no such bound is possible under virtual clock multiplexing because of the punishment feature. Thus, good *worst-case* performance can only be guaranteed under virtual clock multiplexing under stringent access control. Also, the additional flexibility of PGPS may be useful in an integrated services network.

Stop-and-Go Queueing is proposed by Golestani in [5, 6, 7], and is based on a network-wide time slot structure. A finite number of connection types are defined, where a type g connection is characterized by a fixed frame size of T_g. Each session i is assigned a connection type g. The admission policy under which delay and buffer size guarantees can be made is that no more than $r_i T_g$ bits may be submitted during any type g frame. Thus bandwidth is allocated by peak rates rather than average rates. While this is a more restrictive admission policy than leaky bucket (as we shall see in Section 6), it allows for tight control of jitter in the network. The service discipline is not work-conserving, but is designed to preserve the smoothness properties of the admitted traffic. It has the advantage of being very amenable to analysis. PGPS uses the links more efficiently and flexibly and can provide comparable worst-case end-to-end delay bounds. Since it is work-conserving, PGPS will also provide better average delay than stop-and-go for a

given access control scheme. However, stop-and-go queueing may provide significantly better bounds on jitter.

6 Leaky Bucket

Consider the leaky bucket scheme [12] of Figure 6.1. Tokens or permits are generated at a fixed rate, ρ, and packets can be released into the network only after removing the required number of tokens from the token bucket. There is no bound on the number of packets that can be buffered, but the *token* bucket contains at most σ bits worth of tokens. In addition to securing the required number of tokens, the traffic is further constrained to leave the bucket at a maximum rate of $C > \rho$. The constraint imposed by the

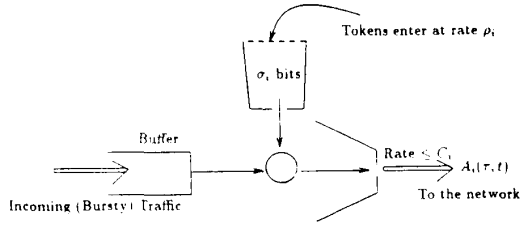

Figure 6.1: The Leaky Bucket

leaky bucket is as follows: If $A_i(\tau, t)$ is the amount of session i flow that leaves the leaky bucket and enters the network in the time interval $(\tau, t]$, then

$$A_i(\tau, t) \leq \min\{(t-\tau)C_i, \sigma_i + \rho_i(t-\tau)\}, \quad \forall t \geq \tau \geq 0, \tag{7}$$

for every session i. We say that session i conforms to (σ_i, ρ_i, C_i), or $A_i \sim (\sigma_i, \rho_i, C_i)$.

This model for incoming traffic is essentially identical to the one recently proposed by Cruz [2], [3], and it has also been used in various forms to represent the inflow of parts into manufacturing systems by Kumar [9]. The arrival constraint is attractive since it restricts the traffic in terms of average rate (ρ), peak rate (C), and burstiness (σ and C).

Figure 6.2 shows how a fairly bursty source might be characterized using the constraints. Represent $A_i(0,t)$ as in Figure 6.2. Let there be $l_i^A(t)$ bits worth of tokens in the session i token bucket at time t. We assume that the session starts out with a full bucket of tokens. If $K_i(t)$ is the total number of tokens arriving at the session i bucket in $(0,t]$ (it does not include the full bucket of tokens that session i starts out with), then

$$K_i(t) = \min_{0 \leq \tau \leq t} \{A_i(0,\tau) + \rho_i(t-\tau)\}.$$

We now have the following expression for $l_i^A(t)$:

$$l_i^A(t) = \sigma_i + K_i(t) - A_i(0,t). \tag{8}$$

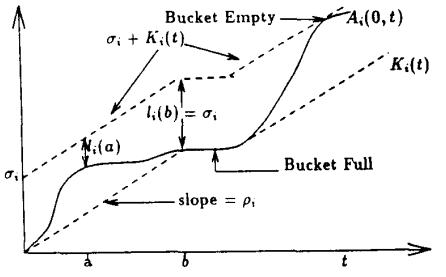

Figure 6.2: $A_i(t)$ and $l_i(t)$.

7 Analysis

In this section we analyze the worst-case performance of single node GPS systems for sessions that operate under leaky bucket constraints i.e., the session traffic is constrained as in (7).

There are N sessions, and the only assumptions we make about the incoming traffic are that $A_i \sim (\sigma_i, \rho_i, C_i)$ for $i = 1, 2, ..., N$, and that the system is empty before time zero. The server is work conserving (i.e. it is never idle if there is work in the system), and operates at the fixed rate of 1.

Let $S_i(\tau, t)$ be the amount of session i traffic served in the interval $(\tau, t]$. Note that $S_i(0,t)$ is continuous and non-decreasing for all t (see Figure 7.1). The session i backlog at time τ is defined to be

$$Q_i(\tau) = A_i(0,\tau) - S_i(0,\tau).$$

The session i delay at time τ is denoted by $D_i(\tau)$, and is the amount of time that session i flow arriving at time τ spends in the system before departing. Thus

$$D_i(\tau) = \inf\{t \geq \tau \ : \ S_i(0,t) = A_i(0,\tau)\} - \tau. \tag{9}$$

From Figure 7.1 we see that $D_i(\tau)$ is the horizontal distance between the curves $A_i(0,t)$ and $S_i(0,t)$ at the

$\phi_i \frac{\partial V(t_j+\tau)}{\partial \tau}$. Thus, V can be interpreted as increasing at the marginal rate at which backlogged sessions receive service.

Now suppose that the k^{th} session i packet arrives at time a_i^k and has length L_i^k. Then denote the virtual times at which this packet begins and completes service as S_i^k and F_i^k respectively. Defining $F_i^0 = 0$ for all i, we have

$$\begin{aligned} S_i^k &= \max\{F_i^{k-1}, V(a_i^k)\} \\ F_i^k &= S_i^k + \frac{L_i^k}{\phi_i}. \end{aligned} \tag{6}$$

There are three attractive properties of the virtual time interpretation from the standpoint of implementation. First, the virtual time finishing times can be determined at the packet arrival time. Second, the packets are served in order of virtual time finishing time. Finally, we need only update virtual time when there are events in the GPS system. However, the price to paid for these advantages is some overhead in keeping track of the sets B_j, which is essential in the updating of virtual time:

Define Next(t) to be the *real* time at which the next packet will depart the GPS system after time t if there are no more arrivals after time t. Thus the next virtual time update after t will be performed at $Next(t)$ if there are no arrivals in the interval $[t, \text{Next}(t)]$. Now suppose a packet arrives at some time, t, and that the time of the event just prior to t is τ (if there is no prior event, i.e. if the packet is the first arrival in a busy period, then set $\tau = 0$). Then, since the set of busy sessions is fixed between events, $V(t)$ may be computed from (5), and the packet stamped with its virtual time finishing time. Next(t) and the new value of B are also computed.

Given this mechanism for updating virtual time, PGPS is defined as follows: When a packet arrives, virtual time is updated and the packet is stamped with its virtual time finishing time. The server is work conserving and serves packets in increasing order of time-stamp.

5 Comparing PGPS to other schemes

Under weighted round robin, every session i, has an integer weight, w_i associated with it. The server polls the sessions according a *precomputed sequence* in an attempt to serve session i at a rate of $\frac{w_i}{\sum_j w_j}$. If an empty buffer is encountered, the server moves to the next session in the order instantaneously. When an arriving session i packet just misses its slot in a frame it cannot be transmitted before the next session i slot. If the system is heavily loaded in the sense that almost every slot is utilized, the packet may have to wait an $O(N)$ slot times to be served, where N is the number of sessions sharing the server. Since PGPS approximates GPS to within one packet transmission time regardless of the arrival patterns, it is immune to such effects. PGPS also handles variable length packets in a much more systematic fashion than does weighted round robin. However, if N or the packets sizes are small then it is possible to approximate GPS well by weighted round robin.

Zhang proposes an interesting scheme called virtual clock multiplexing [13]. Virtual clock multiplexing does not allow bursty users to increase delay to packets from other "better behaved" sessions, and is also able to treat sessions differently according to their average rates. It also provides good *average* packet delay. Occasionally, packets are "punished " for using the server at a rate higher than the guaranteed rate even when the additional service was given under lightly loaded conditions i.e. it was not at the expense of packets from any other sessions. Under PGPS no such punishment occurs, which why rate guarantees can be seen by *every* arriving packet, even in the absence of rate admission control. Also, the additional flexibility of PGPS may be useful in an integrated services network.

Stop-and-Go Queueing is proposed by Golestani in [5, 6, 7], and is based on a network-wide time slot structure. A finite number of connection types are defined, where a type g connection is characterized by a fixed frame size of T_g. The admission policy under which delay and buffer size guarantees can be made is that no more than $r_k T_g$ bits are admitted into the network in any type g frame, where r_k is the transmission rate assigned to session i. Thus bandwidth is allocated by peak rates rather than average rates. While this is a more restrictive admission policy than leaky bucket (as we shall see in Section 6), it allows for tight control of jitter in the network. The service discipline is not work-conserving, but is designed to preserve the smoothness properties of the admitted traffic. It has the advantage of being very amenable to analysis. PGPS uses the links more efficiently and flexibly and can provide comparable worst-case end-to-end delay bounds. Since it is work-conserving, PGPS will also provide better average delay than stop-and-go for a

ordinate value of $A_i(0,\tau)$. Clearly, $D_i(\tau)$ depends on the arrival functions $A_1, ..., A_N$. We are interested in computing the maximum delay over all time, and over all arrival functions that are consistent with (7). Let D_i^* be the maximum delay for session i. Then

$$D_i^* = \max_{(A_1,...,A_N)} \max_{\tau \geq 0} D_i(\tau).$$

Similarly, we define the maximum backlog for session i, Q_i^*:

$$Q_i^* = \max_{(A_1,...,A_N)} \max_{\tau \geq 0} Q_i(\tau).$$

We can characterize the session i traffic that leaves

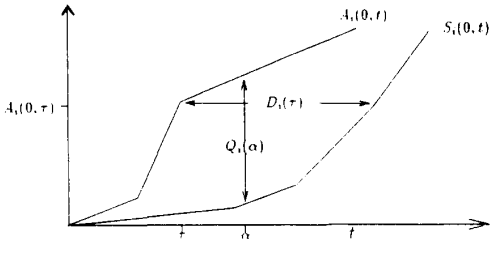

Figure 7.1: $A_i(0,t)$, $S_i(0,t)$, $Q_i(t)$ and $D_i(t)$

the server in terms of additional parameters so that $S_i \sim (\sigma_i^{\text{out}}, \rho_i^{\text{out}}, C_i^{\text{out}})$. The problem we will solve in the following sections is:

Given $\phi_1, ..., \phi_N$ for a GPS server of rate 1 and given (σ_j, ρ_j, C_j), $j = 1, ..., N$, what are D_i^*, Q_i^* and the parameters σ_i^{out}, ρ_i^{out} and C_i^{out} for every session i?

7.1 Preliminaries

Given $A_1, ..., A_N$, let σ_i^τ be defined for each session i and time $\tau \geq 0$ as

$$\sigma_i^\tau = Q_i(\tau) + l_i(\tau) \tag{10}$$

where $l_i(\tau)$ is defined in (8). Thus σ_i^τ is the sum of the number of tokens left in the bucket and the session i backlog at the server at time τ. If $C_i = \infty$ we can think of σ_i^τ as the maximum amount of session i backlog at time τ^+, over all arrival functions that are identical to $A_1, ..., A_N$ up to time τ. Observe that $\sigma_i^0 = \sigma_i$.

Define a **system busy period** to be a maximal interval B such that for any $\tau, t \in B$, $\tau \leq t$:

$$\sum_{i=1}^{N} S_i(\tau, t) = t - \tau.$$

Since the system is work conserving, if $B = [t_1, t_2]$, then $\sum_{i=1}^N Q_i(t_1) = \sum_{i=1}^N Q_i(t_2) = 0$.

Lemma 2 *When $\sum_j \rho_j < 1$, the length of a system busy period is at most*

$$\frac{\sum_{i=1}^N \sigma_i}{1 - \sum_{i=1}^N \rho_i}.$$

A simple consequence of this Lemma is that all system busy periods are bounded when $\sum_i \rho_i < 1$. Since session delay is bounded by the length of the largest possible system busy period, the session delays are bounded as well. Since the system is stable, $\rho_i^{\text{out}} = \rho_i$, and σ_i^{out} is bounded for each session i. Also, $C_i^{\text{out}} = 1$, since the maximum rate at which the session can send traffic on the output link for session i is the rate at which it operates. Finally, $\sigma_i^{\text{out}} \geq \sigma_i$ for each i. To see this, suppose that
$A_i(0,\tau) = \min\{C_i\tau, \sigma_i + \rho_i\tau\}$, $\tau \geq 0$. Then for sufficiently large τ, we have $A_i(0,\tau) = \sigma_i + \rho_i\tau$. Since the system is stable, any session i backlog must be cleared. Then the amount served at this time must be $\sigma_i + \rho_i\tau$, implying that $\sigma_i^{\text{out}} \geq \sigma_i$.

Let a **session i busy period** be a maximal interval B_i contained in a single system busy period, such that for all $\tau, t \in B_i$:

$$\frac{S_i(\tau,t)}{S_j(\tau,t)} \geq \frac{\phi_i}{\phi_j} \quad j = 1, 2, .., N. \tag{11}$$

Notice that it is possible for a session to have zero backlog during its busy period. However, if $Q_i(\tau) > 0$ then τ *must* be in a session i busy period at time τ. We have already shown that

Lemma 3 : For every interval $[\tau, t]$ that is in a session i busy period

$$S_i(\tau,t) \geq (t-\tau)\frac{\phi_i}{\sum_{j=1}^N \phi_j}.$$

Notice that when $\phi = \phi_i$ for all i, the service guarantee reduces to

$$S_i(\tau,t) \geq \frac{t-\tau}{N}.$$

7.2 Greedy Sessions

Session i is defined to be greedy starting at time τ if

$$A_i(\tau,t) = \min\{C_i(t-\tau), l_i(\tau)+(t-\tau)\rho_i\}, \text{ for all } t \geq \tau. \tag{12}$$

In terms of the leaky bucket, this means that it uses as many tokens as possible (i.e. sends at maximum

possible rate) for all times $\geq \tau$. At time τ, session i has $l_i(\tau)$ tokens left in the bucket, but it is constrained to send traffic at a maximum rate of C_i. Thus it takes $\frac{l_i^\tau}{C_i - \rho_i}$ time units to deplete the tokens in the bucket. After this, the rate will be limited by the token arrival rate, ρ_i.

Figure 7.2 depicts the arrival function A_i^τ which is greedy starting at time τ. Inspection of the figure

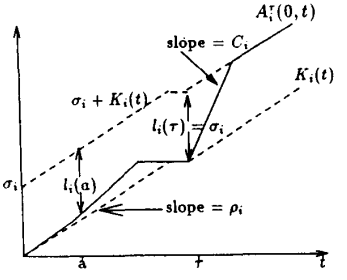

Figure 7.2: A session i arrival function that is greedy from time τ.

(and from (12)), we see that if a system busy period starts at time zero, then

$$A_i^0(0,t) \geq A(0,t), \quad \forall A \sim (\sigma_i, \rho_i, C_i), \quad t \geq 0.$$

We have shown the following [11]:

Theorem 3 *Under generalized processor sharing, for every session i: D_i^*, Q_i^* and σ_i^{out} are achieved (not necessarily at the same time) when every session is greedy starting at time zero.*

This is an intuitively pleasing and satisfying result. It seems reasonable that if a session sends as much traffic as possible at all times, it is going to impede the progress of packets arriving from the other sessions. But notice that we are claiming a worst case result, which implies that it is never more harmful for a subset of the sessions to "save up" their bursts, and to transmit them at a time greater than zero. While there are many examples of service disciplines for which which this "all-greedy regime" does not maximize delay, the amount of work required to establish Theorem 3 (see [11]) is still somewhat surprising.

7.3 An All-greedy GPS system

Theorem 3 suggests that in order to compute D_i^*, Q_i^*, and σ_i^{out}, we should examine the dynamics of a system in which all the sessions are greedy starting at time 0, the beginning of a system busy period. Since the system busy period is finite we can label the sessions in the order in which their first individual busy periods are terminated. To simplify the presentation, we will assume that $C_i \geq 1$ for all i—the general case is dealt with in [11].

From (7) we know that

$$A_i^0(0,\tau) = \min\{C_i\tau, \sigma_i + \rho_i\tau\}, \quad \tau \geq 0,$$

and let us assume for clarity of exposition, that $\sigma_i > 0$ for all i. Both $Q_i(\tau)$ and $D_i(\tau)$ can be found from Figure 7.3. As we explained earlier, $Q_i(\tau)$ is just the

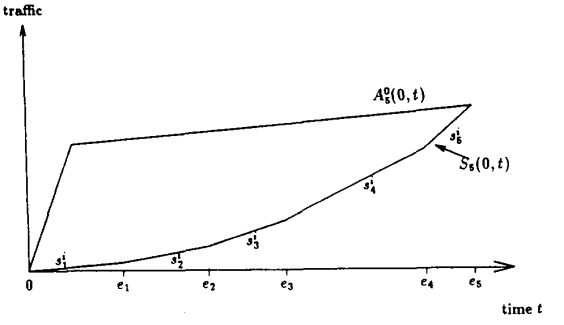

Figure 7.3: Session 5 arrivals and departures after 0, the beginning of a system busy period.

vertical distance between the two curves, and $D_i(\tau)$ is the horizontal distance between the curves $A_i^0(0,t)$ and $S_i(0,t)$ at the ordinate value of $A_i^0(0,\tau)$.

Session i will arrive at rate C_i for time

$$b_i = \frac{\sigma_i}{C_i - \rho_i}, \tag{13}$$

after which it arrives at rate ρ_i. Let e_i be the time (≥ 0) when the session i busy period is terminated for the first time. Then note that $b_i \leq e_i$ for every i (since $C_i \geq 1$).

In the interval $[0, e_1]$, each session i is in a busy period (since we assumed that $\sigma_i > 0$ for all i), and is served at rate $\phi_i/(\sum_{k=1}^N \phi_k)$. At e_1 one of the sessions, which we label session 1 by convention, ends its busy period. Since session 1 is greedy after 0, it follows that $\rho_1 < \phi_1/(\sum_{k=1}^N \phi_k)$. (It is easy to show that such a session must exist when $\sum_i \rho_i < 1$.) Now each session j, still in a busy period will be served at rate $(1 - \rho_1)\phi_j/(\sum_{k=2}^N \phi_k)$ until a time e_2, when another session, 2, ends its busy period. Similarly, we can can show that

$$\rho_k < \frac{(1 - \sum_{j=1}^{k-1} \rho_j)\phi_k}{\sum_{j=k}^N \phi_j}, \quad k = 1, 2, ..., N. \tag{14}$$

As shown in Figure 7.3, the slopes of the various segments that comprise $S_i(0,t)$ are $s_1^i, s_2^i, ..., s_i^i$. From (14) we deduce that:

$$s_k^i = \frac{(1 - \sum_{j=1}^{k-1} \rho_j)\phi_i}{\sum_{j=k}^{N} \phi_j}, \quad k = 1, 2, ..., i.$$

It can be seen that $\{s_k^i\}$ $k = 1, 2, ..., i$ forms an increasing sequence.

For any two sessions i, j indexed greater than k we can define a "univeral slope" s_k, by:

$$s_k = \frac{s_k^i}{\phi_i} = \frac{s_k^j}{\phi_i} = \frac{1 - \sum_{j=1}^{k-1} \rho_j}{\sum_{j=k}^{N} \phi_k}, \quad i, j > k, \; k = 1, 2, ..., N.$$

This allows us to describe the behavior of **all** the sessions in a single figure as is depicted in Figure 7.4. It

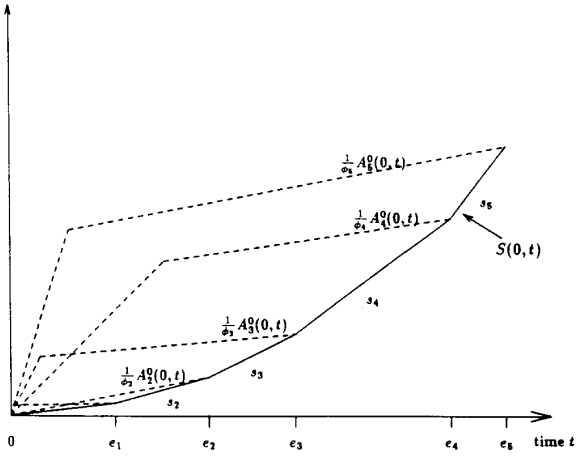

Figure 7.4: The dynamics of an all-greedy GPS system.

is interesting to note that the universal service curve $S(0,t)$ is identical to the Virtual Clock function, $V(t)$, defined in (5).

8 Calculating D_i^*, Q_i^* and σ_i^{out}

The universal service curve shown in Figure 7.4 can be efficiently computed. Thus, in order to compute worst case performance measures we assume that all the sessions will be greedy starting at some time zero, and that they all have full buckets of tokens at this time.

Again we assume that $C_i \geq 1$ for all i. For the more general case we refer the reader to [11]. Find the largest indexed slope j such that $s_j < \frac{\rho_i}{\phi_j}$. If there is no such slope, set $j = 0$. Let

$$t_i = \max\{e_j, b_i\},$$

where b_i was defined in (13).

Now we have:

$$Q_i^* = A_i^0(0, t_i) - \phi_i S(0, t_i).$$

To compute D_i^* when $C_i < \infty$ we first find t^* the smallest time such that

$$A_i^0(0, t^*) = \phi_i S(0, t_i).$$

Then

$$D_i^* = t_i - t^*.$$

Now if $C_i = \infty$ we first check if $\sigma_i \leq \phi_i S(0, t_i)$. If it is, then compute t^* and D_i^* as above. Otherwise find e^* the smallest time such that

$$S(0, e^*) = \frac{\sigma_i}{\phi_i}$$

and set

$$D_i^* = e^*.$$

Finally, we can evaluate σ_i^{out} through the following result:

Lemma 4 *For every session i:*

$$\sigma_i^{\text{out}} = \max\{\sigma_i, Q_i^*\}.$$

In principle it is possible to give closed form expressions for these quantities once the order in which the sessions end their individual busy periods has been determined. However, such expressions are messy and do not yield much insight.

9 Picking the ϕ's

Suppose a new session wants to use a server that is currently being shared by sessions $\{1, ..., N\}$. Every session i is characterized by $\sigma_i, \rho_i, C_i, d_i$ where d_i is the worst case packet delay that can be tolerated by session i. For the active sessions we have $\phi_1, ..., \phi_N$, and wish to assign ϕ_{N+1} so that the new session can be accommodated without violating any of the existing guarantees on throughput and delay. The following is one possible approach that could be used.

1. If $\sum_{i=1}^{N+1} \rho_i \geq 1$ then reject session $N+1$. Otherwise proceed to step 2.

2. Find ϕ_{N+1}^{min}, the smallest value of ϕ_{N+1} that would ensure session $N+1$ a worst case delay of d_{N+1}.

3. For every session $i = 1, 2, ..., N$ find ϕ_{N+1}^{i}, the largest value of ϕ_{N+1} that would still ensure session i a worst case delay of d_i.

4. Compute $\phi_{N+1}^{max} = \min_{i=1,...,N} \phi_{N+1}^{i}$. If $\phi_{N+1}^{min} > \phi_{N+1}^{max}$ then reject session $N+1$. Otherwise

$$\phi_{N+1} = \frac{\phi_{N+1}^{max} - \phi_{N+1}^{min}}{2}.$$

Note that any choice of $\phi_{N+1} \in [\phi_{N+1}^{min}, \phi_{N+1}^{max}]$ will meet worst case delay guarantees. However, picking the extreme points is not advisable since otherwise no more sessions could be accepted after session $N+1$. In step four we pick the midpoint of the interval.

It is also worth noting, that the assignment scheme presented here assumes the existence of a "fairness checker" that prevents a few sessions from monopolizing the server at the expense of later arriving sessions whose requirements cannot be met.

10 Conclusions

We presented a fair, flexible and efficient multiplexing scheme called Generalized Processor Sharing that appears to be appropriate for integrated services networks. We also proposed a packet based scheme that approximates GPS closely. We analyzed the GPS multiplexer when the sources are constrained by leaky buckets, and presented an efficient algorithm to determine worst case delays for a given single server GPS system. A method to add new users to the system was also discussed.

Elsewhere [10], we have extended this work to PGPS networks of arbitrary topologies. We provide an efficient method for computing bounds on session delay and backlog for a broad class of GPS networks. It is hoped that our results in this paper and in the sequel can form the basis for a highly flexible and efficient rate-based flow control scheme for integrated services networks.

References

[1] D. Bertsekas and R. Gallager, *Data Networks*, Prentice Hall, Englewood Cliffs, NJ, 1991.

[2] R. L. Cruz, *A calculus for network delay, Part I: Network elements in isolation*, IEEE Transactions on Information Theory, 37 (1991), pp. 114–131.

[3] ——, *A calculus for network delay, Part II: Network analysis*, IEEE Transactions on Information Theory, 37 (1991), pp. 132–141.

[4] A. Demers, S. Keshav, and S. Shenkar, *Analysis and simulation of a fair queueing algorithm*, Proceedings of SIGCOMM '89, (1989), pp. 1–12.

[5] S. J. Golestani, *Congestion-free transmission of real-time traffic in packet networks*, in Proceedings of IEEE INFOCOM '90, San Fransisco, CA, 1990, pp. 527–536.

[6] ——, *A framing strategy for connection managment*, in Proceedings of SIGCOMM '90, 1990.

[7] ——, *Duration-limited statistical multiplexing of delay sensitive traffic in packet networks*, in Proceedings of IEEE INFOCOM '91, 1991.

[8] A. C. Greenberg and N. Madras, *Comparison of a fair queueing discipline to processor sharing*, in Performance '90, Edinborough, Scotland, 1990, pp. 239–254.

[9] C. Lu and P. R. Kumar, *Distributed scheduling based on due dates and buffer prioritization*, tech. rep., University of Illinois Technical Report, 1990.

[10] A. K. Parekh and R. G. Gallager, *A Generalized Processor Sharing approach to flow control—The Multiple Node Case*, Tech. Rep. 2076, Laboratory for Information and Decision Systems, MIT, 1991.

[11] ——, *A Generalized Processor Sharing approach to flow control—The Single Node Case*, Tech. Rep. 2040, Laboratory for Information and Decision Systems, MIT, 1991.

[12] J. Turner, *New directions in commnunications (or Which way to the information age)*, IEEE Communications Magazine, 24 (1986), pp. 8–15.

[13] L. Zhang, *A New Architecture for Packet Switching Network Protocols*, PhD thesis, Department of Electrical Engineering and Computer Science, MIT, August 1989.