Fairness and Efficiency in Processor Sharing Protocols to Minimize Sojourn Times

Eric J. Friedman*
Shane G. Henderson†
School of Operations Research and Industrial Engineering,
Cornell University, Ithaca, NY 14853.

March 20, 2002

Abstract

We consider the problem of designing a preemptive protocol that is both fair and efficient when one is only concerned with the sojourn time of the job and not intermediate results. Our Fair Sojourn Protocol (FSP) is both efficient, in a strong sense (similar to the shortest remaining processing time protocol – SRPT), and fair, in the sense of guaranteeing that it outperforms processor sharing (PS) for every non-terminal job in every busy period for any sample path while providing the same performance for the terminal job.

Our primary motivation is web serving in which the standard protocol is PS, while recent work proposes using SRPT or variants. Our work suggests both a framework in which to evaluate proposed protocols and an attractive new protocol, FSP.

*Work supported in part by National Science Foundation Grant No. ANI–9730162.
Email:friedman@orie.cornell.edu, www:http://www.orie.cornell.edu/∼friedman
†Work supported in part by National Science Foundation Grant No. DMI–0085165.
Email: shane@orie.cornell.edu, www:http://www.orie.cornell.edu/∼shane

1 Motivation

Many queuing problems arising on computers and the Internet allow processor time to be shared among multiple jobs simultaneously. These include data transmission, batch job processing and web serving. In all three of these examples, some version of processor sharing is standard. For example, data transmission with FIFO queuing in which file transfers are broken into multiple packets, essentially shares a link among multiple processes simultaneously, while both time-sharing on computers and processor sharing on web servers are the norm. Aside from the simplicity of these processor sharing (PS) protocols, the typical justification for PS is "fairness": all jobs receive similar processing resources.[1]

Recently, non-sharing protocols, such as shortest-remaining-processing-time (SRPT) have been recommended for both data transmission [Mod97b, Mod97a] and web serving [BHB01].[2] While such protocols obviously improve the sojourn times (times from arrival of the job to completion of its service) for small jobs, [BHB01] present intriguing results that show that, in many cases, even the sojourn times for large jobs are improved under SRPT in comparison with PS, where these improvements can be dramatic. In fact, understanding these fascinating results is what motivated this paper.

However, improvement is not guaranteed, and in some settings, large

[1] Technically, almost all of these examples do not truly exhibit processor sharing, since they are typically implemented via time division multiplexing, but as long as the time slices are sufficiently small it is reasonable to treat this as true processor sharing.

[2] Various heuristic protocols that are combinations of PS and SRPT have also been studied. In particular [Mod97b, Mod97a, Che98] propose families of protocols which are meant to bridge the gap between SRPT and "fair" protocols. Simulation analysis suggest that these protocols are promising, but unfortunately, as yet there are no formal guarantees of their performance.

jobs can do much worse under SRPT compared to PS. For example, it is well known that under SRPT, long jobs can be starved [Tan92, Sta95, BCM98], where one can interpret starvation strictly in the sense of never receiving service, or more informally as in the sense where a large job has an extremely long (but finite) sojourn time. We believe that potential starvation provides an impediment to the wide scale implementation of protocols such as SRPT.

In this paper, we provide a framework for both understanding these "counter-intuitive" results and for designing protocols that are both "fair" and "efficient". For example, while PS is fair, it is extremely inefficient, and so its performance may be significantly worse than an efficient protocol such as SRPT, for *all* classes of jobs. SRPT, while efficient, is not necessarily fair. We propose a new protocol, the "fair sojourn protocol" (FSP) which is both fair and efficient and thus provides significant performance improvements over PS while guaranteeing fairness. In fact, our analysis provides some basic tools for analyzing and constructing other fair and efficient protocols.[3]

Our main insight applies to problems for which one only cares about the sojourn times of jobs and intermediate processing stages are irrelevant. For example, in data transmission, our analysis applies to bulk file transfer, such as in web serving, but not to telnet sessions, in which intermediate processing is important.

[3]Fair and efficient mechanisms are the focus of much of cooperative game theory and cost sharing research, see e.g., [Mou91, Mou02] for introductions and overviews. For a specific application of these ideas in a related model, see Cres and Moulin [CM01], who have studied a scheduling model in which all jobs arrive at the same time.

2 Model

We consider a single server queue with an arbitrary arrival process and arbitrary distribution of job sizes. We allow these to be correlated in any manner. In particular, any of the standard queuing models are allowed in our model as is any arbitrary deterministic sequence of arrivals and service times. Our key assumption is that when a job arrives, its processing time is known with certainty. We define a sample path as the sequence of arrival times and job lengths, i.e., $((t_1, l_1), (t_2, l_2), \ldots, (t_n, l_n))$. (The reason that we consider such a general model is that our main results involve sample-path arguments, and as such do not rely on distributional assumptions.)

We will consider preemptive scheduling protocols under which the processor may be shared. Formally, such a protocol can be defined as a function ω where $\omega(i;t)$ is the instantaneous work rate on job i at time t. We assume that $\omega(i;t)$ is nonnegative and piecewise continuous. Furthermore, we assume that at all times t at which there is work present in the system, $\sum_i \omega(i;t) = \mu$ where μ is the processing rate, i.e., the server allocates its maximum possible effort. (For simplicity, and without loss of generality, in the following we will set $\mu = 1$.) Of course, jobs can not be processed before they arrive; thus, if $t < t_i$ then $\omega(i;t) = 0$. The completion time of job i is then given by the smallest value of t such that $\int_{t_i}^{t} \omega(i;s)ds = l_i$. Lastly, we are interested in online protocols, where $\omega(i;t)$ cannot depend on arrivals after time t.[4] The protocol is the mapping from arrivals to a function ω and we will typically denote a protocol as $p \in P$ where P is the set of all allowed protocols.

Given a scheduling protocol p let $so(p;l)$ be the expected steady-state

[4]Essentially, Cres and Moulin [CM01] study an offline version of this problem.

sojourn time of jobs of length l under protocol p (assuming it exists). One reasonable goal is to minimize some weighted expectation of the sojourn times, such as the average sojourn time, $E_l[so(p;l)]$. It is well known that the protocol SRPT minimizes this function (over the set of all protocols). Another important measure is the slowdown ratio $sl(p;l) = so(p;l)/l$. For example, HB01 analyze starvation (or fairness) in terms of the slowdown ratio.

As we will demonstrate later, FSP is strictly preferred over PS for all jobs except those that terminate a busy period, and thus is guaranteed to have a better average sojourn and slowdown than PS. As is well known, SRPT is always better than PS in terms of average sojourn times, since it is optimal for that metric [Sch68, Smi76]. As shown in BH01 for specific examples, it may even be better in terms of average slowdown; in fact, they show that in some cases, every class receives a smaller expected slowdown under SRPT than PS. However, in many cases, under SRPT large jobs may occasionally experience extremely large delays, leading to larger average slowdowns for large jobs under SRPT than under PS. In addition, as our simulation results for a specific example show, FSP can have a smaller slowdown ratio than SRPT for large jobs, and have better worst case slowdown behavior for all job sizes. Thus, if slowdown and starvation are considered important criteria then FSP may be quite attractive.

3 Efficiency and Fairness

The key assumption in our analysis is that only the sojourn time of a job is relevant for performance evaluation. Thus, we will consider the values of

sojourn times in developing notions of efficiency and fairness.

First, we present a simple baseline definition of efficiency based on the notion of dominance.

Definition 1 *A protocol p' dominates protocol p if no job completes later under p' than under p on any sample path, and there is at least one job on at least one sample path that completes earlier under p' than under p.*

Definition 2 *A protocol p is efficient if there is no other protocol p' that dominates it.*

Inefficient protocols are clearly suboptimal for any reasonable loss measure that only depends on the sojourn times of jobs. Similarly, if some protocol is dominated by another protocol, then the dominated protocol will be no better under any reasonable loss measure.

Definition 3 *A reasonable loss measure is a mapping $\pi : P \rightarrow \Re$ such that if p' dominates p then $\pi(p') \leq \pi(p)$.*

For example, expected sojourn time and expected slowdown are reasonable performance measures, as is any weighted average of sojourn times.

One can develop strict versions of these notions. First we define a busy period as a sequence of arrivals that all overlap, i.e., a busy period ends whenever a job completes and there are no jobs waiting for service.[5] Note that busy periods consist of the same jobs for any nonstalling protocol (i.e., for any protocol that works at the maximum possible rate whenever work is present).

[5]For example, if there are no waiting jobs and the current job completes at time t, then we say that the busy period is over, even if a new job arrives at time t.

Definition 4 *A protocol p' strictly dominates protocol p if, for every busy period on every sample path, every job except the terminal one completes strictly earlier under p' than under p.*

Definition 5 *A strict loss measure is a mapping $\pi : P \to \Re$ such that if p' strictly dominates p then $\pi(p') < \pi(p)$.*

Define an arrival process to be trivial (steady-state trivial) if the probability (probability in steady-state) of a busy period consisting of more than one job occurring is 0. For example, if the arrival process is Poisson, then it is neither trivial nor steady-state trivial. Over finite time intervals, weighted averages of sojourn times in which every job is given nonzero weight are strict loss measures, such as average sojourn time or average slowdown. If we consider limiting averages, then these same measures will be strict loss measures for any nontrivial steady state arrival process.

Next we consider fairness. Following the literature we will define PS as the standard of fairness.

Definition 6 *A protocol p is fair if it dominates PS.*

This definition differs significantly from those normally used in cooperative game theory and cost allocation [Mou91, Mou02]; however, it seems appropriate in this context.

4 FSP

In this section we present the Fair Scheduling Protocol (FSP), providing an intuitive introduction, the formal protocol and a simple implementation.

4.1 Intuition

Consider a sample path in which n jobs of length 1 arrive at the same time. Under PS the sojourn time for every job is n. However, if we simply order the jobs randomly and then schedule them in order, we get an average sojourn time of $(n+1)/2$ and a longest sojourn time of n. Thus, all jobs receive better service under this ordering. Thus, we see that PS can be very inefficient. In fact, in this example, PS is the worst nonstalling protocol possible.

More generally, for any sample path under PS, consider two concurrent jobs a and b where a will complete first at time t. If we were to trade some of b's processing time before t to a for the same amount of a's processing time after t (and before the completion time of b) then a would complete earlier and b's completion time would be unchanged. Thus it is easy to see that since we only care about sojourn times, it is always possible to improve on PS. In fact, for any busy period, we can strictly improve the service on every single job except for the terminal one, i.e., the job that completes last under PS. This is the motivation for FSP, which guarantees that all jobs except the terminal one in any busy period are completed earlier than they would have under PS. Note that there are many other protocols that also achieve this goal. We plan to examine this class of protocols in the future.

Thus, sharing processor time is always suboptimal. In general, once a job has started processing, that job should continue to completion. Perhaps the only exception to this rule is when a new job arrives. If the new job is small enough, it might be reasonable to switch to processing the new job.[6]

[6]There may be situations with correlated arrival times where the non-arrival of a job might provide information that could be used to change the job in service. However, we expect situations of this type to be rare, and do not consider them here.

4.2 Explaining FSP

FSP attempts to mimic PS as much as possible while obeying the following rule. In FSP we compute the time at which jobs would complete under PS and then order the jobs in terms of earliest (PS) completion times. FSP then devotes its full attention to the (uncompleted) job with the earliest (PS) completion time. Such a definition is, in fact, well defined as the following lemma and discussion show.[7]

Lemma 1 *Consider any two sample paths of arrivals that are identical up to time t and two jobs a, b that both arrive before t. Then under PS they complete in the same order on both sample paths.*

Proof: If one or both jobs complete by time t then there is nothing to prove. If not, then both jobs are still being processed at time t. The result follows immediately, since under PS, jobs complete in the order of shortest remaining processing time. (Note however, that at any point in time the remaining processing times under PS can differ from those under SRPT.) ■

The importance of this lemma is that given that several jobs remain to be completed at some time t, they will complete in the same order under PS *regardless* of the arrival times and sizes of jobs that arrive after time t. Therefore, the ordering of processing under FSP is well-defined.

Thus, there are no reversals under PS, where job A begins processing and then we switch to job B even though job B had entered the system before

[7] We have not specified the protocol's behavior in the event of ties. Although any tie breaking rule would work, for concreteness we assume that in the case of a tie we choose the job with the earliest arrival time.

we started job A.[8]

There are many nontrivial sample paths on which FSP and SRPT are identical. The simplest example of this occurs when all jobs arrive at the same time, in which case FSP schedules identically to SRPT. See also Examples I and III in the next section. The main difference between FSP and PS is that FSP protects long jobs that have been in the system for a long time against the possibility of starvation, perhaps due to a string of arriving short jobs.

4.3 Examples

We now present several detailed examples of the FSP protocol in order to explain its behavior and provide counter-examples for our later analysis.

4.3.1 Example I

Our first example demonstrates the common case when FSP behaves identically to SRPT. Consider the following example: A job of length 3 arrives at $t = 0$ then a job of length 1 arrives at $t = 1$ and then a job of length 1 arrives at $t = 2$.

Under SRPT we see that job 1 begins in service, then when job 2 arrives it preempts job 1 and goes into service. It then completes at the same time that job 3 arrives and begins service. Then job 3 completes and job 1 returns to service. The following table summarizes this behavior, where a $*$ indicates that the specified job is in service from the current time to the next indicated one.

[8] In the common case where processor sharing is actually implemented via time division multiplexing then one can view the inefficiency of PS as arising from its numerous alterations between jobs being processed.

SRPT and FSP: Example I					
Time	0	1	2	3	5
Remaining work (1)	*3	2	2	*2	0
Remaining work (2)	x	*1	0	0	0
Remaining work (3)	x	x	*1	0	0

Thus, we see that first job 2 completes at $t = 2$, then job 3 completes at $t = 3$ and finally job 1 completes at $t = 5$. In this case FSP is identical to SRPT.

Next consider the behavior of PS as tabulated here:

PS: Example I						
Time	0	1	2	3.5	4.5	5
Remaining work (1)	*3	*2	*1.5	*1	*0.5	0
Remaining work (2)	x	*1	*0.5	0	0	0
Remaining work (3)	x	x	*1	*0.5	0	0

In this case, first job 2 completes at $t = 3.5$, then job 3 completes at $t = 4.5$ and finally job 1 completes at $t = 5$.

Thus we see that, in this case, both FSP and SRPT strictly dominate PS as jobs 2 and 3 have significantly shorter sojourn times under either of these than under PS while job 1, the terminal job, completes at the same time for both.

4.3.2 Example II

In this example we show that FSP may differ from SRPT and while FSP path dominates PS, SRPT does not. Consider the modification of the above example where job 3 is of length 1.9. In this case we have the following:

For SRPT:

SRPT: Example II					
Time	0	1	2	3.9	5.9
Remaining work (1)	*3	2	2	*2	0
Remaining work (2)	x	*1	0	0	0
Remaining work (3)	x	x	*1.9	0	0

Here, first job 2 completes at $t = 2$, then job 3 completes at $t = 3.9$ and finally job 1 completes at $t = 5.9$.

Next consider the behavior of PS:

PS: Example II						
Time	0	1	2	3.5	5.5	5.9
Remaining work (1)	*3	*2	*1.5	*1	0	0
Remaining work (2)	x	*1	*0.5	0	0	0
Remaining work (3)	x	x	*1.9	*1.4	*0.4	0

We see that first job 2 completes at $t = 3.5$, then job 1 completes at $t = 5.5$ and finally job 3 completes at $t = 5.9$. Thus, job 1, the largest job, completes earlier under PS than under SRPT.

Lastly, consider the behavior of FSP:

FSP: Example II					
Time	0	1	2	4	5.9
Remaining work (1)	*3	2	*2	0	0
Remaining work (2)	x	*1	0	0	0
Remaining work (3)	x	x	1.9	*1.9	0

In this case first job 2 completes at $t = 2$, then job 1 completes at $t = 4$ and finally job 3 completes at $t = 5.9$. Thus, job 1, the largest job, completes much earlier under FSP than under either PS or SRPT while job 3 completes later under FSP than under SRPT.

This example is easily modified to show that FSP can obtain arbitrarily large improvements (for large jobs) over SRPT and PS.

4.3.3 Example III

In this example we show a path on which jobs complete under FSP in a different order than they do under PS, something that might appear at first to contradict the definition of FSP. Consider the following example: Jobs of length 3 and 10 arrive at $t = 0$ and then a job of length 0.9 arrives at $t = 4$.

First consider the behavior of PS:

PS: Example III					
Time	0	4	6.7	6.9	13.9
Remaining work (1)	*3	*1	*0.1	0	0
Remaining work (2)	*10	*8	*7.1	*7	0
Remaining work (3)	x	*0.9	0	0	0

Under PS, first job 3 completes at $t = 6.7$, then job 1 completes at $t = 6.9$ and finally job 2 completes at $t = 13.9$.

Next, consider the behavior of FSP (which coincides with that of SRPT in this case):

FSP: Example II					
Time	0	3	4	4.9	13.9
Remaining work (1)	*3	0	0	0	0
Remaining work (2)	10	*10	9	*9	0
Remaining work (3)	x	x	*0.9	*0	0

Under FSP, first job 1 completes at $t = 3$, then job 3 completes at $t = 4.9$ and finally job 2 completes at $t = 13.9$.

Thus, in this example jobs complete in a different order under FSP (1,3,2) than under PS (3,1,2). This is because under FSP, job 1 completes before job 3 arrives, so even though job 3 would get higher priority than job 1 under FSP, this is irrelevant. Finally, note that both jobs 1 and 3 complete earlier under FSP than under PS while job 2 completes at the same time for both.

4.3.4 Example IV

Now we present a simple example demonstrating starvation. Consider a system in which a new job of length 1 arrives at unit intervals, $t = 0, 1, \ldots, 99$ and a single job of length 1.1 also arrives at $t = 0$. Under SRPT the "long" job completes at $t = 101.1$ while it completes at 2.1 under FSP. (It completes at (approximately) $t = 2.97$ under PS.)

Thus, as is well known, when there is a steady stream of short jobs, a single long job can receive extremely poor service under SRPT.

4.4 Implementation

Although FSP appears to be significantly more complex to implement than either PS or SRPT we believe the additional overhead is not significant. Below we present a simple implementation that is not computationally intensive, which will useful in our formal analysis. For example, compared to the computation performed by a typical webserver to serve a webpage, the computation of the protocol seems insignificant.

The underlying notion of this implementation is that we track the performance of PS as if it were applied "in the background". This allows us to determine the order in which jobs should be served under FSP.

The basis for our implementation is the ordered linked list v, which has elements $v_i = (j_i, r_i, c_i)$. The job j_i has remaining processing time r_i (under the virtual PS), $c_i = 0$ implies that the job has completed under FSP (although it may not yet have completed under PS which is why it remains in the list), and $c_i = 1$ means it has not. The list is ordered in increasing order of r_i. Associated with v is the time t at which it was last updated, which for

simplicity is considered an external variable. Let $|v|$ denote the number of elements in v and if $|v| > 0$ then v_1 is the first element in the list.

Our implementation will rely upon 5 main routines: NextVirtualCompletionTime, ProcessJob, VirtualJobCompletion, RealJobCompletion, and JobArrival. The routines VirtualJobCompletion and JobArrival are used to track the progress that PS would make were it applied to the observed jobs. These 5 routines may be defined as follows.

NextVirtualCompletionTime(v,t): `[Returns the next ‘‘virtual’’ completion time]`

`If` $|v| = 0$ `then return` $\emptyset$

`else return` $t + r_1 * |v|$

ProcessJob(v): `[Returns the job number to process]`

`If` $|v| = 0$ `then return` $\emptyset$

`else [`$k = \{\min i \mid c_i = 1\}$

`return` j_k`]`.

VirtualJobCompletion(v,t,s): `[Update when a job completes]`

`For all elements of` v `set` $r_i = r_i - (s - t)/|v|$.

`Remove` v_1 `from` v

`Set` $t = s$.

RealJobCompletion(v,t,s,j): `[Update when a job completes]`

`Find` i `s.t.` $j_i = j$.

`Set` $c_i = 0$.

JobArrival(v,t,s,l,j): `[Update when a new job arrives]`

`For all elements of` v `set` $r_i = r_i - (s - t)/|v|$

`Insert` $(j, l, 1)$ `into` v. `(maintaining the sort`[9]`)`

`Set` $t = s$.

Now, the implementation of FSP is straightforward. At $t = 0$ let $v = \emptyset$ and $t = 0$. During the running of the algorithm, we only need to do computation when a job arrives, virtually completes or completes. When a job j of length l arrives at time s call JobArrival(v, t, s, l, j), call ProcessJob(v) to find the next job to go into service and reset the virtual completion time by calling NextVirtualCompletionTime(v, t). When we reach a virtual completion time at time s call VirtualJobCompletion(v, t, s). When the current job j completes, call RealJobCompletion(v, t, s, j) and then call ProcessJob(v) to find the next job to go into service.

The computation required of this algorithm upon an arrival or a completion (virtual or real) is $O(|v|)$ and that $|v|$ is less than or equal to the number of jobs that would have been in service at this time under PS for this sample path. This does not seem to be significantly computationally burdensome compared to the typical complexity of the jobs being serviced. Nonetheless the precise computational costs of FSP deserve further study.

¿From a more practical point of view, when applied to webserving, Harchol-Balter et. al. [HBBSA01] point out that implementation of web serving protocols is difficult in current webserving software. In particular, it is difficult to achieve precise scheduling of jobs. Nonetheless, they provide an approximate implementation of SRPT which shows significant improvements of the default implementation over PS. We expect that similar implementations of FSP should also be possible. More importantly, given the inefficiency of PS,

[9] In case of ties place the new job last among equals.

we believe that web serving software needs to be developed that allows for better job scheduling.

5 Evaluating Efficiency and Fairness

First we show that both FSP and SRPT are efficient.

Theorem 2 *Both FSP and SRPT are efficient.*

Proof: The proof for SRPT follows easily from the fact that it minimizes expected sojourn time. For FSP we use the following argument.

Consider an arbitrary protocol p' and assume that there is a sample path under which p' improves upon FSP, so that all jobs complete either at the same time as in FSP, or earlier. We will show that no job can complete strictly earlier under p' than under FSP.

Before proceeding, we note the following. Consider the k'th job to complete under FSP. Then during that job's time in the system, from its arrival to its completion, no job that completes later than the k'th job can receive service since this would contradict Lemma 1. From this it is easy to see that the first job to complete under FSP does so at the earliest possible time, since it is uninterrupted.

We proceed inductively. Consider the first job to complete under FSP. By the discussion in the previous paragraph it cannot complete sooner under any protocol. Thus this job completes at the same time under both p' and FSP. Assume that the first n jobs that complete under p' and FSP do so at the same time. Now consider the $(n+1)$st job to complete under FSP. By the discussion in the previous paragraph the $(n+1)$st job to complete must

do so as early as possible, conditional upon the servicing of the n previously completed jobs. These n jobs completed at the same times under both p' and FSP, so that under FSP, the time at which the $(n+1)$st job completes is at least as early as the time at which the $(n+1)$st job completes under p'. Therefore, the $(n+1)$st job completes at the same time under both p' and FSP, completing the proof. ■

Although FSP and SRPT are both efficient, PS is not. This follows from our next result which shows that FSP strongly dominates PS. Thus, PS is wasteful in a very strong sense.

Theorem 3 *FSP strongly dominates PS.*

Proof: To prove the theorem we follow the working of the algorithm we presented in the previous section. Let v and w be vectors indicating the remaining work for each job in the system under both PS and FSP respectively. The vectors are ordered in exactly the same way, and so that $v_1 \leq v_2 \leq \ldots \leq v_n$. (Hence elements i in both vectors correspond to the remaining work for the *same* job under both disciplines.) It may be the case that $w_i = 0$ while $v_i > 0$ for several values of i.

In the following discussion, an event is either a new job arriving, a job completing under PS (a virtual service completion), or a job completing under FSP. We will prove by induction on the sequence of events in a busy period that for all i, $\sum_{j=1}^{i} w_j \leq \sum_{j=1}^{i} v_j$ at (immediately after) all event times. (We will write this as $w \ll v$.) This will prove the result, since the next job to complete under PS is always job 1, and at that job completion time (which is an event), $0 = v_1 \geq w_1$, so that $w_1 = 0$, i.e., the job in position 1 completed either at the same instant, or earlier, under FSP.

Suppose that the first job to arrive in a busy period is of size y. At the time of this arrival, $|v| = 1$ and $v_1 = w_1 = y$. Thus, the base case is true. Suppose the result is true at the time of event m, where $m \geq 1$. Let v and w be the vectors at the time of this event.

Let x be the time between the occurrence of event m and event $m+1$. Let v' and w' be the vectors v and w updated for the period between event m and event $m+1$ (but not for event $m+1$). Let i be the index of the first nonzero value in w. Then

$$\begin{aligned} w'_j &= w_j = 0, \quad j = 1, \ldots, i-1, \\ w'_i &= w_i - x, \\ w'_j &= w_j, \quad j = i+1, \ldots, n, \text{ and} \\ v'_j &= v_i - x/|v| \quad \forall j. \end{aligned}$$

It is then easy to see, using the fact that $w \ll v$, that immediately before the occurrence of event $m+1$, the ordering $w' \ll v'$ holds and that all $v'_j \geq 0$ (otherwise a virtual completion would have occurred in the interval). In fact, we get strict inequality for all components from i onwards except for the last component.

Now consider the different types of events that may occur, and their impacts on the vectors v and w.

Case 1: Virtual service completion (completion of job under PS). Since v is maintained in increasing order, we may take the completing job to correspond to v_1. We have that $0 = v'_1 \geq w'_1$, so $w'_1 = 0$, and it follows that this job completes under FSP either at the same time, or earlier, than it would under PS. Set v and w equal to v' and w' respectively, with the first

component deleted. Then $w \ll v$, and the inductive step for this case is complete.

Case 2: Real service completion. In this case, one of the jobs completes under FSP. The vectors v' and w' remain unchanged. We set $v = v'$, $w = w'$, and so $w \ll v$. The inductive step for this case is complete.

Case 3: A new job, of size y, arrives. Insert the value y into the vector v' maintaining the sort, to give v. In case of ties, place y at the last position possible. Insert the value y into the vector w' at the same position to give w. Then it is easy to see that since $w' \ll v'$, we also have that $w \ll v$. The inductive step for this case is complete.

In all 3 cases, the relation $w \ll v$ is preserved, and this completes the proof. ■

Thus, as discussed earlier, this implies that for any strict loss measure, FSP will be strictly better than PS. For example, for any nontrivial arrival process[10], both average sojourn time and average slowdown will be strictly less for FSP than for PS.

Note that neither SRPT nor FSP dominates the other, as can be seen from the examples discussed earlier. Thus, depending on the arrival process either SRPT or FSP (or any other efficient protocol) can do better on different loss measures. For example, SRPT is optimal with respect to average sojourn time, while it is possible, and probably common (see the next section) that FSP will be superior to SRPT under average slowdown. In addition there is no guarantee that SRPT will be fair.

[10]Recall that an arrival process is trivial if the probability of two jobs overlapping is 0.

6 Simulations

The path-by-path results given earlier establish that FSP strongly dominates PS, but that neither FSP nor SRPT dominate the other. Thus, if one restricts attention to these three protocols, it is clear that one should implement either FSP or SRPT. We have shown through examples of sample paths that FSP can provide significantly shorter sojourn times for large jobs as compared to SRPT, since FSP is guaranteed to be fair. But how significant is this? This question is important to consider, given that FSP demands more background computation than SRPT.

In this section we present a small simulation experiment that helps to shed light on the potential tradeoffs between FSP and SRPT. Our goal is not to present a comprehensive simulation study that explores all the intricacies of the tradeoffs, but rather to show that there is indeed value in exploring FSP as an alternative to SRPT.

Consider a single-server system where jobs arrive according to a Poisson process at rate λ. Successive job sizes are i.i.d. with mean 1. (This choice of mean job size is arbitrary, as one can always choose time units to ensure that this holds.) Following BH01, job sizes are chosen to have a truncated Pareto distribution. In particular, the distribution function of job sizes is

$$F(x) = \frac{b^{-\alpha} - x^{-\alpha}}{b^{-\alpha} - c^{-\alpha}}$$

for $x \in [b, c]$, where $c > b > 0$. We chose $c = 10^7, \alpha = 2$, and $b = (2 - c^{-1})^{-1}$, which ensures that the mean job size is 1. The constant $\alpha = 2$ differs from that chosen in BH01 ($\alpha = 1.1$) to simplify some of the analysis[11]. (Choosing

[11] Choosing $\alpha = 1.1$ would require us to determine b above numerically

Protocol	$(0, 2]$	$(2, 4]$	$(4, 7]$	$(7, 10]$	$(10, 20]$	$(20, \infty)$
SRPT	1.84 ± 0.01	6.1 ± 0.2	10.4 ± 0.7	14 ± 1	16 ± 2	17 ± 3
FSP	2.3 ± 0.4	8.2 ± 0.6	11 ± 1	13 ± 1	14 ± 2	15 ± 2
PS	18 ± 2	18 ± 2	18 ± 2	18 ± 2	18 ± 2	17 ± 3
SRPT - FSP	-0.43 ± 0.04	-2.1 ± 0.5	-1.0 ± 0.8	0.6 ± 0.9	2.1 ± 0.9	1.9 ± 0.7

Table 1: The first three rows give 95% confidence intervals for the expected mean slowdown over the first 10,000 jobs broken down by bin. The final row gives a 95% confidence interval for the expected difference between SRPT and FSP mean slowdown.

$\alpha \approx 1$ apparently reflects current opinion regarding the distribution of file sizes transmitted over the Internet.) Given the modest goals of this section, $\alpha = 2$ seems like a reasonable selection. We chose $\lambda = 0.95$, so that the system is very heavily loaded.

We simulated the completion of 10,000 jobs under PS, SRPT and FSP. Common random numbers were used, so that each protocol saw the same sample path of jobs. For reporting purposes, we grouped the jobs into 6 intervals (or bins) according to job size. The bins were $(0, 2]$, $(2, 4]$, $(4, 7]$, $(7, 10]$, $(10, 20]$ and $(20, \infty)$. For each bin, we recorded the mean of the slowdown ratios and the maximum of the slowdown ratios over all jobs that fell in that bin. This calculation was repeated 30 independent times to enable the construction of confidence intervals.

Table 1 gives the results for the expected *mean* slowdown over 10,000 jobs for SRPT, FSP and PS broken down by bin so that the effect of job size becomes apparent. It also gives a confidence interval for the difference in performance (in each bin) between SRPT and FSP. Note that in all cases except one, the confidence intervals for the difference do not contain zero, so that the differences are statistically significant.

Protocol	$(0, 2]$	$(2, 4]$	$(4, 7]$	$(7, 10]$	$(10, 20]$	$(20, \infty)$
SRPT	31 ± 3	48 ± 4	58 ± 9	54 ± 8	52 ± 10	35 ± 7
FSP	29 ± 2	37 ± 4	40 ± 5	38 ± 6	36 ± 6	28 ± 5
PS	69 ± 8	61 ± 7	55 ± 7	48 ± 7	42 ± 7	32 ± 5
SRPT - FSP	2 ± 3	12 ± 4	18 ± 7	16 ± 5	16 ± 6	7 ± 4

Table 2: The first three rows give 95% confidence intervals for the expected maximum slowdown over the first 10,000 jobs broken down by bin. The final row gives a 95% confidence interval for the expected difference between SRPT and FSP maximum slowdown.

We see from these results that, as expected, PS is not competitive with either SRPT and FSP. The well-known result that PS gives a uniform slowdown over all job sizes is also apparent. The difference between FSP and SRPT is harder to discern. SRPT performs better than FSP on small jobs, while the reverse is true for large jobs. However, the two methods perform quite similarly in terms of expected mean slowdown.

Table 2 gives the results for the expected *maximum* slowdown within each bin over 10,000 jobs for SRPT, FSP and PS. It also gives a confidence interval for the difference in performance (in each bin) between SRPT and FSP. Note that in all cases except one, the confidence intervals for the difference do not contain zero, so that the differences are statistically significant.

Again we see the poor performance of PS relative to both FSP and SRPT. We also see that FSP outperforms SRPT in terms of the expected maximum slowdown of jobs. For example, for job sizes in the interval $(7, 10]$, SRPT gives an expected maximum slowdown (over the first 10,000 jobs) of approximately 54, while the corresponding value for FSP is approximately 38. The confidence interval for the difference in performance shows that with 95%

confidence the difference between these 2 values is at least 11, which is a significant fraction of the absolute maximums (54 and 38) observed. This marked improvement in expected maximum slowdown is apparent in almost all bins, clearly showcasing the value of FSP in this example.

Nonetheless, the key difference between FSP and SRPT is the tradeoffs between fairness and expected sojourn time and more analysis is needed to understand these.

7 Concluding Comments

We believe that the formal analysis developed in this paper provides a useful framework for analyzing protocols with processor sharing. However, it is merely a first cut. There are many other notions of efficiency and fairness that could be used. Our analysis was motivated by the general theory of cost allocation [You85, Mou02] and its recent application to scheduling protocols [CM01]. We believe that the combination of ideas from this literature with those from scheduling and queuing theory is an important part of understanding these issues.

There remain many interesting open problems relating to other efficient protocols, of which there are many. Additionally, one important direction for further research is the extension of our analysis and protocols to cases where job times are not completely known upon arrival but can only be estimated (perhaps with improving accuracy) during processing. This would seem to be important for dynamic web pages, such as those created by a database query in which the processing time is unknown, but can often be estimated with some degree of accuracy. In this setting one could modify

FSP to use expected processing times. However, we have no formal results in that setting.

Also, we believe that sample path arguments, as used in this paper, are important in the analysis and that standard queuing models may be inadequate. For example, in web serving the arrival processes may be extremely complex and simple arrival processes are likely to be insufficient models of reality. For example, one might even expect arrivals to be highly correlated or perhaps arrival rates might even be endogenous, since faster serving encourages users to increase their request rate.

The key to our analysis is the assumption that performance of a protocol only depends on sojourn times and not intermediate results. If one accepts this then perhaps the only argument for PS over FSP is based on the complexity of FSP. However, given the amount of computing power available in most applications, we believe that the overhead from FSP is negligible compared to the processing requirements of typical jobs, such as those in webserving. The simulation results reported in the previous section suggest that the protective aspects of FSP compared to SRPT are also significant, although the comparison here is not as clearcut.

Although our analysis of FSP compared to PS is clearcut, and we believe compelling, the comparison of FSP to SRPT is not. Since neither path dominates the other this is not surprising. Additionally, the comparison between FSP and other heuristic protocols [Mod97b, Mod97a, Che98] designed for these environments remains wide open. Since there are few analytic results, clearly a careful simulation-based study is called for. Nonetheless, we believe that the analytic (sample path) aspects of our analysis might be applicable

to understanding and refining these protocols.

References

[BCM98] M. Bender, S. Chakrabarti, and S. Muthukrishnan. Flow and stretch metrics for scheduling continous job streams. In *Proc. 9th ACM-SIAM Symposium on Discrete Algorithms*, 1998.

[BHB01] Nikhil Bansal and Mor Harchol-Balter. Analysis of srpt scheduling: Investigating unfairness. To appear in Proceedings of ACM Sigmetrics 2001 Conference on Measurement and Modeling of Computer Systems, 2001.

[Che98] L. Cherkasova. Scheduling strategies to improve response time for web applications. In *High-performance computing and network-ing: international conference and exhibition*, page 305314, 1998.

[CM01] H. Cres and H. Moulin. Scheduling with opting out: Improving upon random priority. *Operations Research*, 49(4):565–577, 2001.

[HBBSA01] Mor Harchol-Balter, Nikhil Bansal, Bianca Schroeder, and Mukesh Agrawal. Size-based scheduling to improve web performance. mimeo, CMU, 2001.

[Mod97a] E. Modiano. Scheduling algorithms for message transmission over a satellite broadcast system. In *Proceedings of IEEE MILCOM97*, pages 628–634, 1997.

[Mod97b] E. Modiano. Scheduling packet transmissions in a multi-hop packet switched network based on message length. In *International Conference on Computer Communications and Networks*, 1997.

[Mou91] H. Moulin. *Axioms of cooperative decision making*. Cambridge University Press, New York, 1991.

[Mou02] H. Moulin. Axiomatic cost and surplus-sharing. In Arrow, Sen, and Suzumura, editors, *Handbook of Social Choice and Welfare*. North-Holland, New York, 2002.

[Sch68] L.E. Schrage. A proof of the optimality of the shortest processing remaining time discipline. *Operations Research*, 16:678690, 1968.

[Smi76] D.R. Smith. A new proof of the optimality of the shortest remaining processing time discipline. *Operations Research*, 26:197199, 1976.

[Sta95] W. Stallings. *Operating Systems*. Prentice Hall, 2nd edition, 1995.

[Tan92] A.S. Tanenbaum. *Modern Operating Systems*. Prentice Hall, 1992.

[You85] H. P. Young. *Cost Allocation: Methods, principles and applications*. Elsevier Science, New York, 1985.