Simulation Modelling Practice and Theory 17 (2009) 513–527

Contents lists available at ScienceDirect

Simulation Modelling Practice and Theory

journal homepage: www.elsevier.com/locate/simpat

Efficient fair algorithms for message communication

Sergey Gorinsky [a,*], Eric J. Friedman [b], Shane Henderson [b], Christoph Jechlitschek [c]

[a] Washington University, Department of Computer Science and Engineering, Bryan Hall, 1 Brookings Drive, St. Louis, MO 63130, USA
[b] Cornell University, School of Operations Research and Information Engineering, Rhodes Hall, Ithaca, NY 14853, USA
[c] Intel GmbH, Dornacher Strasse 1, 85622 Munich, Germany

ARTICLE INFO

Article history:
Available online 9 September 2008

Keywords:
Capacity allocation
Efficient message communication
Minimal average delay
Fair algorithm
Slack system

ABSTRACT

A computer network serves distributed applications by communicating messages between their remote ends. Many such applications desire minimal delay for their messages. Beside this efficiency objective, allocation of the network capacity is also subject to the fairness constraint of not shutting off communication for any individual message. Processor Sharing (PS) is a de facto standard of fairness but provides significantly higher average delay than Shortest Remaining Processing Time (SRPT), which is an optimally efficient but unfair algorithm. In this paper, we explore efficient fair algorithms for message communication where fairness means that no message is delivered later than under PS. First, we introduce a slack system to characterize fair algorithms completely and develop efficient fair algorithms called Pessimistic Fair Sojourn Protocol (PFSP), Optimistic Fair Sojourn Protocol (OFSP), and Shortest Fair Sojourn (SFS). Then, we prove that a fair online algorithm does not assure minimal average delay attainable with fairness. Our analysis also reveals lower bounds on worst-case inefficiency of fair algorithms. We conduct extensive simulations for various distributions of message sizes and arrival times. During either temporary overload or steady-state operation, SFS and other newly proposed fair algorithms support SRPT-like efficiency and consistently provide much smaller average delay than PS.

1. Introduction

A message is an atomic data unit meaningful for a distributed application. In many such applications, it is desirable to deliver a message to a remote computer as soon as possible, i.e., to minimize the delay of the message. Applications communicate messages over packet-switching networks where the maximum packet size is relatively small: 1500-byte packets dominate the Internet. Hence, delivering a long message might involve thousands or even millions of packets. The ability of a network to minimize message delay is constrained chiefly by the path capacity from the source to the destination. Specifically, the capacity of a bottleneck link on the path is the main factor determining minimal achievable delay. Beside the efficiency objective of average delay minimization, allocation of the bottleneck capacity is also subject to the fairness constraint of not shutting off communication for any individual message.

Shortest Remaining Processing Time (SRPT) transmits messages preemptively in the order of remaining transmission delay and is optimally efficient [15,17]. However, minimal average delay comes at the expense of potential unfairness. Under heavy load, SRPT might starve long messages by delaying them without bound [2].

Processor Sharing (PS) is an alternative classic algorithm that has become a de facto standard of fairness in network capacity allocation [3]. PS instantaneously allocates equal shares of the bottleneck capacity to all pending messages.

* Corresponding author.
E-mail addresses: gorinsky@wustl.edu, gorinsky@arl.wustl.edu (S. Gorinsky), ejf27@cornell.edu (E.J. Friedman), sgh9@cornell.edu (S. Henderson), christoph.jechlitschek@intel.com (C. Jechlitschek).

1569-190X/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.simpat.2008.09.005

Consequently, expected delay of a message under PS is proportional to the message size. Whereas packet-switching networks do not support instantaneous sharing of the bottleneck link capacity, a great number of packet transmission algorithms were proposed to approximate the PS ideal. Packet-grained approximations of PS include Weighted Fair Queuing [5], Start-time Fair Queueing [9], Deficit Round Robin [16], and other algorithms for fair queuing at routers as well as fair end-to-end congestion control schemes exemplified by Transmission Control Protocol [10].

While SRPT is unfair, PS achieves fairness by sacrificing efficiency: average delay of messages under PS is significantly higher. Recent studies reveal remarkable existence of efficient fair algorithms that have it both ways and combine PS fairness with SRPT-like efficiency. Fair Sojourn Protocol (FSP) is a specific efficient representative of the fair algorithmic class where no message experiences longer delay than under PS [6]. FSP transmits the message with the earliest PS completion time and was independently proposed as Virtual Finish Time First (ViFi) [8]. Significant reduction in average delay under FSP versus PS is substantiated both experimentally and analytically [6,8,18].

In this paper, we shed more light on the class of fair algorithms for network capacity allocation where no individual message starves, meaning that the message is delivered no later than under PS. First, we introduce a slack system to characterize fair algorithms completely and derive necessary and sufficient conditions for an online algorithm to be fair. Using the slack system, we propose efficient fair algorithms called Pessimistic Fair Sojourn Protocol (PFSP), Optimistic Fair Sojourn Protocol (OFSP), and Shortest Fair Sojourn (SFS). Then, we prove that neither one of the above trio nor another fair online algorithm guarantees minimal average delay attainable without starvation. Our analysis also reveals lower bounds on worst-case inefficiency of fair algorithms in comparison to the optimally efficient SRPT. We conduct extensive simulations for various distributions of message sizes and arrival times, with arrival patterns ranging from smooth to bursty and different levels of variance in message sizes. The experiments show that SFS supports SRPT-like efficiency and consistently provides much smaller average delay than PS or FSP.

While our paper is written in the context of message communication over computer networks, its analytical methodology and developed algorithms constitute a more general contribution. We believe that our techniques are useful in web server scheduling and other application domains where a bottleneck resource needs to be allocated fairly and efficiently.

The rest of the paper is structured as follows. Section 2 clarifies our model, metrics, and terminology. Section 3 presents the slack system characterization of fair algorithms. Section 4 proposes PFSP, OFSP, and SFS but rules out existence of a fair online algorithm that guarantees minimal average delay achievable without starvation. Section 5 derives the lower bounds on worst-case inefficiency of fair algorithms. Section 6 reports the experimental comparison of SFS with SRPT, PS, and FSP. Finally, Section 7 sums up our findings and discusses future work.

2. Model, terminology, and metrics

We define a *message* as an atomic data unit meaningful for an application. Related studies refer to messages under other names such as jobs or requests. Messages arrive for network transfer in their entirety. *Delay* of a message is time passed from the message arrival until the whole message reaches its destination. Other common names for delay include transfer time, response time, flow time, and sojourn time. *Transmission delay* of a message represents its communication needs and equals $\frac{S}{C}$, where S is the message size, and C is the capacity of the network bottleneck link shared with all the other messages. Transmission delay is also known as processing time, e.g., as reflected in the name of SRPT. We assume that the communications utilize the entire bottleneck link capacity and experience negligible propagation, node processing, and error recovery delays. Then, besides transmission delay, the only other component of message delay is due to waiting for the bottleneck link to become available.

Arrival times and transmission delays of messages characterize network load. Network service is represented by an algorithm that allocates the bottleneck link capacity to unfinished messages. In particular, we are interested in *online algorithms* that have no information about future messages. Capacity allocation enjoys ideal flexibility that allows both instantaneous link sharing and instantaneous transmission preemption.

Since PS has become synonymous with fairness in network resource allocation, we rely on delays of individual messages under PS as a basis for defining the fair algorithmic class.

Definition 1. Starvation is a scenario where a message finishes later than under PS.

Definition 2. An algorithm for capacity allocation is fair if and only if no starvation occurs under the algorithm for any load.

To quantify fairness of an algorithm to a particular message, we introduce a metric of *starvation stretch*.

Definition 3. Starvation stretch $s_X(m)$ of message m under algorithm X is the ratio of message delay $d_X(m)$ under algorithm X to message delay $d_{PS}(m)$ under PS:

$$s_X(m) = \frac{d_X(m)}{d_{PS}(m)}. \tag{1}$$

Note that algorithm X is deemed unfair if there exists load where $s_X(m) > 1$ for at least one message m.

Also note an implicit assumption that network capacity is allocated among messages. We strongly believe that fairness of capacity allocation should be defined with respect to real-world entities, rather than messages or packet flows as in tradi-

tional networking. However, since the important "among what" aspect is orthogonal to our main contributions and requires a separate thorough treatment, we do not explore it further in this paper.

To quantify efficiency of capacity allocation under algorithm X, we measure average delay D_X for all N messages imposed load:

$$D_X = \frac{\sum_{m=1}^{N} d_X(m)}{N}. \tag{2}$$

Because SRPT is an optimally efficient algorithm if fairness concerns are put aside, we use average delay under SRPT as a baseline for assessing efficiency of fair algorithms:

Definition 4. Average letup L_X under algorithm X is the ratio of average delay D_X under algorithm X to average delay D_{SRPT} under SRPT:

$$L_X = \frac{D_X}{D_{\text{SRPT}}}. \tag{3}$$

Although a fair algorithm is not always able to match the ideal efficiency of the unfair SRPT, consistent closeness of average letup L_X to 1 is an indicator that fair algorithm X is highly efficient.

3. Slack system characterization of fair algorithms

In this section, we derive a complete analytic characterization for the class of fair algorithms. First, we observe that pending messages finish under PS in the same order regardless of future message arrivals. Second, we introduce the slack of a message as a key measure of flexibility that a fair algorithm has in allocating the capacity among unfinished messages. Finally, we establish necessary and sufficient conditions for an online algorithm to be fair.

Since fairness is defined in terms of message completion times under PS, we maintain a shadow PS schedule [6,8,14] when reasoning about alternative algorithm X. PS has the following remarkable property, which is proved in [6] and also known in the context of fair packet queuing [5,9].

Lemma 1. *The order of completion times for pending messages under PS is independent of future message arrivals.*

The stable ordering of pending messages under PS creates a possibility for algorithm X to deviate from the PS schedule without a risk that subsequent message arrivals cause starvation. The chief rationale for such deviations is to improve efficiency. In particular, abandoning the instantaneous sharing of PS to transmit one message at a time might reduce average delay significantly [6,8]. We denote remaining transmission delay of message m under PS and algorithm X as $\upsilon_m(t)$ and $w_m(t)$, respectively. To differentiate between the statuses of a message in the two schedules, we formalize the distinction with terms *unfinished* and *pending*.

Definition 5. Message m is unfinished at time t if and only if the message arrives by time t and $w_m(t) > 0$.

Definition 6. Message m is pending at time t if and only if the message arrives by time t and $\upsilon_m(t) > 0$.

Let n be the number of pending messages at time t. If algorithm X is fair, then the n pending messages include all unfinished messages at time t. However, $w_m(t)$ might be 0 for pending message m if algorithm X finishes this message by time t. We index pending messages in a *shadow order*.

Definition 7. The shadow order at time t is the sequence of pending messages indexed as $m = 1, \ldots, n$ in the non-decreasing order of $\upsilon_m(t)$.

To quantify the flexibility that a fair algorithm has in allocating the capacity among unfinished messages, we characterize messages with a measure called *slack*.

Definition 8. Assuming that no messages arrive in the future, consider transmitting messages one after another in the shadow order. Slack $a_m(t)$ of pending message m at time t is the difference between the completion times of message m under PS and in the above schedule.

The *slack system* refers to the trio of n-vectors $\upsilon(t)$, $w(t)$, and $a(t)$ indexed in the shadow order. Below, we derive closed-form expressions for the slack vector.

Theorem 1 (Slack). *The slack of pending message m equals*

$$a_m(t) = (n - m)\upsilon_m(t) + \sum_{i=1}^{m} (\upsilon_i(t) - w_i(t)). \tag{4}$$

Proof. If messages are transmitted one after another in the shadow order, then message m finishes in this schedule at time g:

$$g = t + \sum_{i=1}^{m} w_i(t). \tag{5}$$

PS completes pending messages also in the shadow order. Hence under PS, each message $i = 1, \ldots, m$ with remaining transmission delay $\upsilon_i(t)$ finishes by time f when PS completes message m. Also over time interval $[t; f)$, each message $m+1, \ldots, n$ reduces its remaining transmission delay by the same amount $\upsilon_m(t)$. Thus, the completion time of message m in the shadow PS schedule equals:

$$f = t + (n-m)\upsilon_m(t) + \sum_{i=1}^{m} \upsilon_i(t). \tag{6}$$

Since $a_m(t) = f - g$ by Definition 8, substituting g and f with Eqs. (5) and (6) establishes the theorem. □

While Eq. (4) captures the slack vector at any time t, the following theorem directly expresses the impact of message arrivals on the slack vector.

Theorem 2 (Arrival). *If a message with transmission delay x arrives at time t and becomes pending message m, the slack vector changes as follows:*

$$a_i(t) = \begin{cases} \tilde{a}_i(t) + \tilde{\upsilon}_i(t) & \text{for } i < m, \\ \tilde{a}_{m-1}(t) + (\tilde{n} + 1 - m)(x - \tilde{\upsilon}_{m-1}(t)) & \text{for } i = m, \\ \tilde{a}_{i-1}(t) & \text{for } i > m. \end{cases} \tag{7}$$

where tildes mark the values that the slack system would have if the message did not arrive at time t, while $\tilde{a}_0(t)$ and $\tilde{\upsilon}_0(t)$ denote 0.

Proof. The arrival of the message yields the following vectors $\upsilon(t)$ and $w(t)$:

$$(\upsilon_i(t), w_i(t)) = \begin{cases} (\tilde{\upsilon}_i(t), \tilde{w}_i(t)) & \text{for } i < m, \\ (x, x) & \text{for } i = m, \\ (\tilde{\upsilon}_{i-1}(t), \tilde{w}_{i-1}(t)) & \text{for } i > m. \end{cases} \tag{8}$$

Expressing $a_i(t)$ through Eq. (4), substituting $\upsilon_i(t)$ and $w_i(t)$ according to Eq. (8), and replacing n with $\tilde{n} + 1$ leads us to

$$a_i(t) = \begin{cases} (\tilde{n} + 1 - i)\tilde{\upsilon}_i(t) + \sum_{j=1}^{i} \left(\tilde{\upsilon}_j(t) - \tilde{w}_j(t)\right) & \text{for } i < m, \\ (\tilde{n} + 1 - m)x + \sum_{j=1}^{m-1} (\tilde{\upsilon}_j(t) - \tilde{w}_j(t)) & \text{for } i = m, \\ (\tilde{n} + 1 - i)\tilde{\upsilon}_{i-1}(t) + \sum_{j=1}^{i-1} \left(\tilde{\upsilon}_j(t) - \tilde{w}_j(t)\right) & \text{for } i > m. \end{cases} \tag{9}$$

After applying Eq. (4) again to the right sides of the above, we establish Eq. (7). □

Theorem 2 is important for two reasons. First, since a new arrival does not decrease slack for any unfinished message, new arrivals do not endanger fairness. Second, while the arrival of message m increases slack for messages $1, \ldots, m-1$, the theorem quantifies the extra leeway given to algorithm X in selecting messages for fair transmission. Now, let us examine how transmission of a message affects the slack system.

Theorem 3 (Transmission). *If algorithm X spends time interval $[t; t+\delta)$ on transmission of message m where $\delta \leqslant n \cdot \upsilon_1(t)$, and no messages arrive during this interval, then the slack vector evolves as follows:*

$$a_i(t+\delta) = \begin{cases} a_i(t) - \delta & \text{for } i < m, \\ a_i(t) & \text{for } i \geqslant m. \end{cases} \tag{10}$$

Proof. By transmitting message m over time interval $[t; t+\delta)$, algorithm X reduces remaining transmission delay for message m by δ but does not change remaining transmission delay for any of the other pending messages:

$$w_i(t+\delta) = \begin{cases} w_m(t) - \delta & \text{for } i = m, \\ w_i(t) & \text{for } i \neq m. \end{cases} \tag{11}$$

Since $\upsilon_i(t) \geqslant \upsilon_1(t) \geqslant \frac{\delta}{n}$ for each pending message $i = 1, \ldots, n$, and no messages arrive over the interval, remaining transmission delay for message i in the shadow PS schedule decreases to

$$\upsilon_i(t+\delta) = \upsilon_i(t) - \frac{\delta}{n}. \tag{12}$$

Expressing $a_i(t+\delta)$ through Eq. (4) and then applying Eqs. (11) and (12) to substitute $w_i(t+\delta)$ and $\upsilon_i(t+\delta)$ yields:

$$a_i(t+\delta) = \begin{cases} (n-i)\upsilon_i(t) - \delta + \sum_{j=1}^{i}(\upsilon_j(t) - w_j(t)) & \text{for } i < m, \\ (n-i)\upsilon_i(t) + \sum_{j=1}^{i}(\upsilon_j(t) - w_j(t)) & \text{for } i \geqslant m. \end{cases} \tag{13}$$

After applying Eq. (4) again to the right sides of the above, we establish Eq. (10). □

If no messages arrive, and algorithm X transmits message m until time $t + \min\{w_m(t), n \cdot \upsilon_1(t)\}$, then either the algorithm finishes message m, or PS completes message 1 and thereby reduces the number of pending messages. Because messages $m, \ldots, n$ preserve their slacks, Theorem 3 manifests that transmission of message m might cause starvation only for preceding messages. Hence, transmitting the first unfinished message in the shadow order (as in FSP) does not jeopardize fairness of the algorithm.

If PS completes messages $i = 1, \ldots, m$ at time t, and algorithm X is fair, then $\lim_{u \uparrow t} \upsilon_i(u) = 0$ and $\lim_{u \uparrow t} w_i(u) = 0$ for all these m messages, and Eq. (4) reveals immediately that the reduction of the pending message count from n to $n - m$ at time t does not change slacks of the remaining $n - m$ messages.

After having understood properties of the slack system, we now use the system to derive necessary and sufficient conditions for an online algorithm to be fair.

Theorem 4 (Fairness). *An online algorithm for capacity allocation is fair if and only if $a_m(t) \geqslant 0$ for all messages m at any time t under arbitrary load.*

Proof. ($\Rightarrow$) Suppose $a_m(t) < 0$ for message m at time t under load l. Construct load $\tilde{l}$ from load l by removing all messages that arrive after time t. Since algorithm X is online and operates identically under both loads up to time t, $a_m(t) < 0$ under load $\tilde{l}$ as well. As per Theorem 1, we have

$$t + \sum_{i=1}^{m} w_i(t) > t + (n-m)\upsilon_m(t) + \sum_{i=1}^{m} \upsilon_i(t). \tag{14}$$

While the left side of Ineq. (14) specifies the earliest finish time for all m messages $1, \ldots, m$ under algorithm X, the right side shows the time when all these m messages finish in the shadow PS schedule for load $\tilde{l}$. Hence, at least one of messages $1, \ldots, m$ finishes under algorithm X later than in the shadow PS schedule for load $\tilde{l}$. By Definition 2, algorithm X is unfair. By transposition, fairness of algorithm X implies $a_m(t) \geqslant 0$ for all messages m at any time t under arbitrary load.

($\Leftarrow$) Suppose $a_m(t) \geqslant 0$ for all t and m under arbitrary load. Consider the general scenario of message completion in the shadow PS schedule where m messages $1, \ldots, m$ finish at time t. Let time u be such that no messages arrive or finish under PS during time interval $(u; t)$. Since $a_m(u) \geqslant 0$, Theorem 1 implies

$$\sum_{i=1}^{m} w_i(u) \leqslant n \cdot \upsilon_m(u) + \sum_{i=1}^{m} \upsilon_i(u), \tag{15}$$

where n is the number of pending messages at time t, and

$$\lim_{u \uparrow t} \sum_{i=1}^{m} w_i(u) \leqslant \lim_{u \uparrow t} \left(n \cdot \upsilon_m(u) + \sum_{i=1}^{m} \upsilon_i(u) \right). \tag{16}$$

Since $w_i(u) \geqslant 0$ and $\lim_{u \uparrow t} \upsilon_i(u) = 0$ for all $i = 1, \ldots, m$, we establish

$$\lim_{u \uparrow t} \sum_{i=1}^{m} w_i(u) = 0. \tag{17}$$

Hence, the m messages $1, \ldots, m$ finish under algorithm X also by time t. By Definition 2, algorithm X is fair. □

4. Efficient fair algorithms

The slack system constitutes not only a comprehensive framework for analyzing fairness of capacity allocation algorithms but also a powerful practical tool for design and implementation of efficient fair algorithms. In particular, the algorithm that transmits the first unfinished message in the shadow order is nothing else but FSP. Since FSP never reduces slack for unfinished messages, fairness of FSP is an immediate result of the slack system characterization.

While FSP always follows the shadow order, transmitting another message with shorter remaining transmission delay might decrease average delay. However, the threat of starvation limits the choice of messages available to a fair algorithm. We identify two types of such eligible messages.

Definition 9. An unfinished message is startable at time t if and only if a fair online algorithm can start transmitting the message at time t.

Definition 10. An unfinished message is finishable at time t if and only if starting at time t a fair online algorithm can uninterruptedly transmit the message until its completion regardless of future message arrivals.

The following two theorems characterize startable and finishable messages within the slack system.

Theorem 5 (Startability). *Message m is startable at time t if and only if $a_i(t) > 0$ for all $i < m$ and $a_i(t) \geqslant 0$ for all $i \geqslant m$.*

Proof. Theorem 4 links fairness of an online algorithm to $a_i(t) \geqslant 0$ for each pending message i. By Theorem 3, transmission of message m reduces slack for all $i < m$ but preserves slack for all $i \geqslant m$. Hence, the ability of a fair online algorithm to start transmitting message m at time t is equivalent to $a_i(t) > 0$ for all $i < m$ and $a_i(t) \geqslant 0$ for all $i \geqslant m$. □

Theorem 6 (Finishability). *Message m is finishable at time t if and only if $a_i(t) > w_m(t)$ for all $i < m$ and $a_i(t) \geqslant 0$ for all $i \geqslant m$.*

Proof. Theorem 4 links fairness of an online algorithm to $a_i(t) \geqslant 0$ for each pending message i. By Theorem 3, uninterrupted completion of message m preserves slack for all $i \geqslant m$ but decreases slack by $w_m(t)$ for all $i < m$. Hence, the ability of a fair online algorithm to finish message m uninterruptedly regardless of future message arrivals is equivalent to $a_i(t) > w_m(t)$ for all $i < m$ and $a_i(t) \geqslant 0$ for all $i \geqslant m$. □

While each finishable message is also startable, some startable message m might not be finishable despite its attractiveness due to shorter remaining transmission delay than for any of the finishable messages. A fair online algorithm might still be able to finish this message m uninterruptedly if new messages arrive after time t and raise the slacks of the preceding messages sufficiently high. However, if new arrivals do not fulfil such hope, and the slack of a preceding message hits 0 before message m finishes, the fair algorithm will need to suspend message m, and average delay will likely increase due to the excessive optimism of the algorithm at time t. Below, we define three algorithms SFS, OFSP, and PFSP (introduced as SFS+ in [7]) that take different approaches in pursuing the SRPT-inspired desire to favor messages with shorter remaining transmission delay. All three algorithms index messages in the shadow order.

Definition 11. Pessimistic Fair Sojourn Protocol (PFSP) is the algorithm that transmits the first finishable message with the shortest remaining transmission delay.

Definition 12. Optimistic Fair Sojourn Protocol (OFSP) is the algorithm that transmits the first startable message with the shortest remaining transmission delay.

Definition 13. Shortest Fair Sojourn (SFS) is the algorithm that transmits the first message with the shortest remaining transmission delay if this message is finishable; otherwise, SFS transmits the first unfinished message.

The following theorem establishes fairness of these three algorithms.

Theorem 7. *PFSP, OFSP, and SFS are fair algorithms.*

Proof. Since each of the algorithms transmits a startable message whenever there are unfinished messages, Theorems 4 and 5 imply fairness of the algorithms. □

Whereas FSP routinely provides lower average delay than PS [6,8], a natural question is whether FSP, PFSP, OFSP, SFS, or another fair online algorithm always yields minimal average delay achievable without starvation. The following theorem rules out such possibility.

Theorem 8 (Optimally efficient fairness). *A fair online algorithm does not guarantee minimal average delay attainable without starvation.*

Proof. (1) Consider load l of 12 messages presented in Fig. 1. Since SRPT causes no starvation for this load, it is optimal to transmit the messages in an SRPT order: any permutation of messages 1 through 9 followed by messages 11, 10, and 12. Fig. 2a depicts such an optimal schedule with minimal average delay $\frac{618}{12}$. Note that average delay under an algorithm that transmits message 10 before messages 11 and 12 (as in FSP) is higher.

(2) Consider load $\tilde{l}$ that expands load l with message 13 as shown in Fig. 1. At time 105 when message 13 arrives, messages $1, \ldots, 9$ and 11 are already finished in the optimal schedule for load l, and the slack system with pending messages 10, 13, 11, 12 becomes as follows: $\upsilon(105) = (3, 4, 8, 18)$, $w(105) = (9, 4, 0, 20)$, and $a(105) = (3, 2, 10, 0)$. All three unfinished messages 10, 13, and 12 are startable at time 105. Although message 13 has the shortest remaining transmission delay among the unfinished messages, message 13 is not finishable at time 105. An algorithm that transmits message 13 anyway (as in OFSP) has to suspend message 13 by time 108 to avoid starvation of message 10 because the latter finishes in the shadow PS schedule at time 117, as shown in Fig. 2b. The excessively optimistic decision to transmit message 13 at time 105 yields suboptimal average delay. The best strategy for a fair algorithm under the current circumstances at time 105 is to

Messages of load l	Messages of load $\tilde{l}$	Arrival time	Transmission delay
$1,\ldots,9$	$1,\ldots,9$	0	10 each
10	10	0	14
11	11	90	10
12	12	90	20
	13	105	4

Fig. 1. Two loads in the proof of Theorem 8.

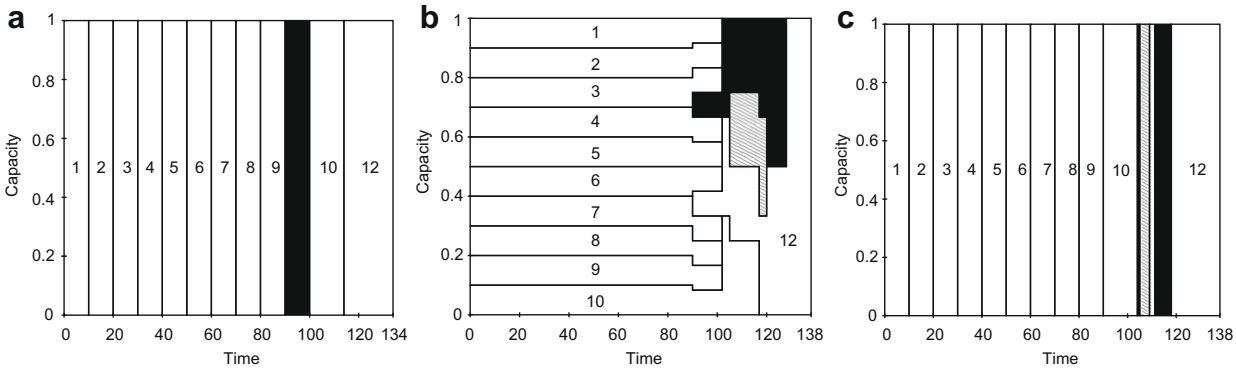

Fig. 2. None of fair online algorithms always provides minimal average delay attainable without starvation; messages 11 and 13 are denoted with solid black and stripes, respectively: (a) optimal schedule for load l; (b) shadow PS schedule for load $\tilde{l}$; (c) optimal schedule for load $\tilde{l}$.

complete transmission of message 10 by time 114 and then transmit messages 13 and 12 (as in PFSP and SFS). The resulting schedule provides average delay $\frac{635}{13}$. Now, return to time 0 and consider a fair algorithm that transmits all the 13 messages of load $\tilde{l}$ one after another in the shadow order (as in FSP): any permutation of messages 1 through 9 followed by messages 10, 11 (suspended at time 105 to transfer message 13), 13, 11 (the rest of it), and 12. Fig. 2c depicts such an schedule. This schedule provides smaller average delay $\frac{634}{13}$ and is optimal for load $\tilde{l}$.

(3) While loads l and $\tilde{l}$ are identical until time 105, the optimal choice of the message (message 10 versus 11) to transmit at time 90 depends on whether message 13 arrives at future time 105. Hence, a fair online algorithm does not guarantee minimal average delay attainable without starvation. □

5. Analysis of worst-case inefficiency

While Section 4 reveals that none of fair online algorithms assures minimal average delay achievable with knowledge of future message arrivals, this section analyzes the worst-case inefficiency of fair online algorithms in comparison to the optimally efficient but unfair SRPT when load contains N messages. First, we report a prior result on PS inefficiency [13] because its proof serves as a foundation for our subsequent reasoning about FSP and other fair online algorithms.

Theorem 9 (PS inefficiency by Motwani et al.). *Worst-case average letup under PS is* $\Omega(\frac{N}{\log N})$.

Proof. Let H_m denote the mth harmonic number $\sum_{i=1}^{m}\frac{1}{i}$. Consider load l where messages 1 and 2 arrive at time 0, and each of them has transmission delay 1. The other $N-2$ messages of load l are such that message $m=3,\ldots,N$ arrives at time H_{m-2} and has transmission delay $\frac{1}{m-1}$. The SRPT schedule that finishes message $m=2,\ldots,N$ at time H_{m-1} and message 1 at time $H_{N-1}+1$ achieves minimal average delay $D_{\mathrm{SRPT}}=\frac{2H_{N-1}+1}{N}$.

Upon the arrival of message $m=3,\ldots,N$ at time H_{m-2}, the PS schedule has m pending messages with remaining transmission delay $\frac{1}{m-1}$. Hence, PS completes all N messages at time $H_{N-1}+1$ and provides average delay $D_{\mathrm{PS}}=\frac{2N+H_{N-1}-1}{N}$. By Definition 4, average letup under PS is

$$L_{\mathrm{PS}}=\frac{2N+H_{N-1}-1}{2H_{N-1}+1}. \tag{18}$$

Since H_N is $\Theta(\log N)$, worst-case average letup under PS is $\Omega\left(\frac{N}{\log N}\right)$. □

Theorem 9 suggests that avoidance of starvation is a potent constraint because PS can be highly inefficient. As we show below, efficiency loss can be substantial even for fair algorithms that strive to improve upon PS by transmitting one message at a time.

Theorem 10 (FSP inefficiency). *Worst-case average letup under FSP is* $\Omega(\frac{N}{\log N})$.

Proof. Consider load $\tilde{l}$ that is identical to load l in the proof of Theorem 9 except for transmission delay of messages 1 and 2. Each of these two messages has transmission delay $1-\frac{1}{3N}$ in load $\tilde{l}$. The SRPT schedule that finishes message 2 at time $1-\frac{1}{3N}$, message $m=3,\ldots,N$ at time H_{m-1}, and message 1 at time $H_{N-1}+1-\frac{2}{3N}$ achieves minimal average delay $D_{\mathrm{SRPT}}=\frac{2H_{N-1}+1-\frac{1}{N}}{N}$.

Upon the arrival of message $m=3,\ldots,N$ at time H_{m-2}, the shadow PS schedule has m pending messages: each of messages 1 and 2 has remaining transmission delay $\frac{1}{m-1}-\frac{1}{3N}$, and every message $3,\ldots,m$ has remaining transmission delay $\frac{1}{m-1}$. Hence, PS completes both messages 1 and 2 at time $H_{N-1}+\frac{2}{3}$ and all the other $N-2$ messages later at time $H_{N-1}+1-\frac{2}{3N}$. The FSP schedule that transmits the N messages one after another as $1,\ldots,N$ finishes message 1 at time $1-\frac{1}{3N}$, message 2 at time $2-\frac{2}{3N}$, message $m=3,\ldots,N$ at time $H_{m-1}+1-\frac{2}{3N}$, and provides average delay $D_{\mathrm{FSP}}=\frac{N+H_{N-1}-\frac{2}{3}+\frac{1}{3N}}{N}$. By Definition 4, average letup under FSP is

$$L_{\mathrm{FSP}}=\frac{N+H_{N-1}-\frac{2}{3}+\frac{1}{3N}}{2H_{N-1}+1-\frac{1}{N}}. \tag{19}$$

Since H_N is $\Theta(\log N)$, worst-case average letup under FSP is $\Omega(\frac{N}{\log N})$. □

Whereas our lower bound on worst-case average letup under FSP is the same as for PS, the following theorem indicates that other fair online algorithms might do better. Nevertheless, fairness still takes a heavy unavoidable toll on their efficiency in the worst case.

Theorem 11 (Fair online inefficiency). *Worst-case average letup under a fair online algorithm is* $\Omega(\sqrt{N})$.

Proof. Consider load of N messages where N is large, $N-1$ short messages are such that message $m=1,\ldots,N-1$ arrives at time $m-1$ and has transmission delay 1, and long message N arrives at time 0 and has transmission delay w such that PS completes message N at time $\frac{N}{2}$.

(1) To estimate w, let $n(t)$ denote the number of pending messages in the shadow PS schedule at time t. Also, let time $k(t)$ be such that for a message pending throughout time interval $[k(t),t)$, remaining transmission delay of the message decreases by 1 over the interval:

$$\int_{k(t)}^{t}\frac{\mathrm{d}x}{n(x)}=1. \tag{20}$$

In particular, $k(\frac{8}{3})=0$. Let p be the number of short messages that PS completes during time interval $[t;t+\delta)$ where $t\geqslant\frac{8}{3}$, δ is a positive integer, and $t+\delta<\frac{N}{2}$. Then, the number of pending messages changes over this time interval as follows:

$$n(\tilde{t}+\delta)-n(t)=\delta-p \tag{21}$$

where $\tilde{t}=\lim_{u\uparrow t}u$. The number of short messages that PS completes during interval $[t;t+\delta)$ is equal to the number of short messages that arrive during interval $[k(t);k(\tilde{t}+\delta))$. Since short messages arrive with period 1, we approximate Eq. (21) with

$$n(\tilde{t}+\delta)-n(t)=\delta-(k(\tilde{t}+\delta)-k(t)). \tag{22}$$

Dividing both sides of Eq. (22) by δ leads us to the difference equation which we approximate with the following differential equation:

$$\dot{n}(t)=1-\dot{k}(t) \tag{23}$$

where the dot represents the time derivative. Differentiating Eq. (20), we receive

$$\frac{1}{n(t)}=\frac{\dot{k}(t)}{n(k(t))}. \tag{24}$$

Taking into account that

$$\int_{0}^{\frac{N}{2}}\frac{\mathrm{d}x}{n(x)}=w, \tag{25}$$

we solve the series of Eqs. (23)–(25) to establish that w is $\Theta(\sqrt{N})$.

(2) The SRPT schedule that transmits the messages one after another as $1,\ldots,N$ achieves minimal average delay $D_{\mathrm{SRPT}}=\frac{2N+w-2}{N}$. Now, consider a fair online algorithm X. Being fair, algorithm X finishes message N by time $\frac{N}{2}$. Average delay under algorithm X is at least average delay in the schedule where message N is transmitted as under algorithm X, and all the short messages are transmitted one after another in the order of their arrival. Whereas each of the last $\lceil\frac{N}{2}+w-1\rceil$ short messages in such schedule experiences delay w, average delay under algorithm X is $D_X\geqslant\frac{(\frac{N}{2}+w-1)w}{N}$. By Definition 4, average letup under X is

$$L_X \geqslant \frac{(\frac{N}{2}+w-1)w}{2N+w-2}. \tag{26}$$

Because w is $\Theta(\sqrt{N})$ as per our derivations above, worst-case average letup under algorithm X is $\Omega(\sqrt{N})$. □

6. Simulations

The worst-case inefficiency bounds reported in Section 5 are high and do not represent typical behavior which we study below. Our experiments reveal that SFS, PFSP, and OFSP yield similar average delays. While each of the three algorithms slightly outperforms the other in some settings, the similar performance suggests that all these algorithms gain their main strength just from considering a message with the shortest remaining transmission delay. Hence, this section compares PS, SRPT, and FSP with SFS only because SFS is computationally simpler.

6.1. Experimental methodology

We simulate transmission of 3000 messages over a link with capacity $C = 10$ Tbps under full link utilization and no data loss. Our choices for the link capacity and traffic are dictated by our desire to model high-speed networks of the future. To characterize intensity of the traffic, we quantify the notion of *load* l as

$$l = \frac{M}{C \cdot T}, \tag{27}$$

where M denotes the average message size, and T is the average message interarrival time. Since the number of messages is finite, all message delays remain finite even with $l > 100\%$. This feature of our experimental setup enables us to compare the evaluated algorithms under long-term overload conditions, which is impossible with analytical techniques that target only steady-state algorithmic behaviors.

We conduct experiments for diverse arrival patterns ranging from smooth to bursty and different levels of variance in message sizes. With respect to message sizes, we report results for uniform, exponential, Lomax, and Pareto distributions. Our experimental arrangements for message interarrival times include exponential and Pareto distributions. The average rate of arrivals is chosen to yield a desired value of load l. The code and running instructions for all the reported simulations are available at our web site [11].

6.2. Fairness: lower delays for all

First, we simulate settings where messages arrive according to a Poisson process, and message sizes are uniformly distributed between 100 GB and 100 TB. Fig. 3a plots the starvation stretch for every seventh of all 3000 (ordered by size) messages in a single experiment at 95% load. Under either of SRPT, FSP or SFS, a small fraction of messages across the whole spectrum of message sizes has starvation stretch 1. These messages finish at exactly the same times as under PS because they conclude a traffic burst by emptying the queue upon both their completion under PS and their finish under SRPT, FSP or SFS. Small and even midsize messages benefit significantly from SRPT, which delivers them up to 50 times faster than PS. However, some large messages starve under SRPT. For example, delay for the least lucky message under the unfair SRPT is about 50 times larger than under PS.

For the fair FSP, Fig. 3a shows that 800 smallest messages enjoy similarly low starvation stretches as under SRPT. To explain the similarity, we observe that a small message is likely to possess both the shortest remaining transmission delay and earliest PS completion time among unfinished messages. For larger messages, the FSP profile becomes different. Starvation stretches of midsize messages rise significantly closer to 1 than under SRPT. On the other hand, the increase enables FSP to finish all large messages by their PS completion times.

SFS also schedules small messages similarly to FSP: respective points in Fig. 3a often coincide. The reason for the similarity is the same as for SRPT versus FSP. Again, SFS and FSP differ in their treatment of midsize and large message. A dense cluster of points around starvation stretch 1 for large messages under SFS indicates that SFS reduces delays for midsize messages by postponing large messages almost as long as possible without causing starvation. In addition to the across-the-spectrum line at starvation stretch 1, Fig. 3a also reveals sparser but still discernible rows of points with starvation stretches $\frac{1}{2}$, $\frac{1}{3}$, and $\frac{1}{4}$. The rows correspond to messages that arrive and finish while 1, 2, or 3 other messages remain both pending under PS and unfinished under SRPT, FSP or SFS.

To expose the discussed trends more clearly, we repeat the experiment 1000 times and average the 1000 obtained sets of starvation stretches sorted in the increasing order of message sizes. Fig. 3b shows that SRPT substantially decreases delays of small and midsize messages but the largest messages typically starve. Under FSP, not only small messages (the rich) benefit from abandoning PS but also the largest messages (the poor) have average starvation stretch about 0.7. Hence, FSP improves upon PS across the board by reducing delays for all classes of messages: rich, middle, and poor! Fig. 3b also illustrates strategic differences between SFS and FSP. By keeping starvation stretches of large messages closer to 1, SFS helps the middle class of midsize messages to enjoy significantly lower delay than under FSP.

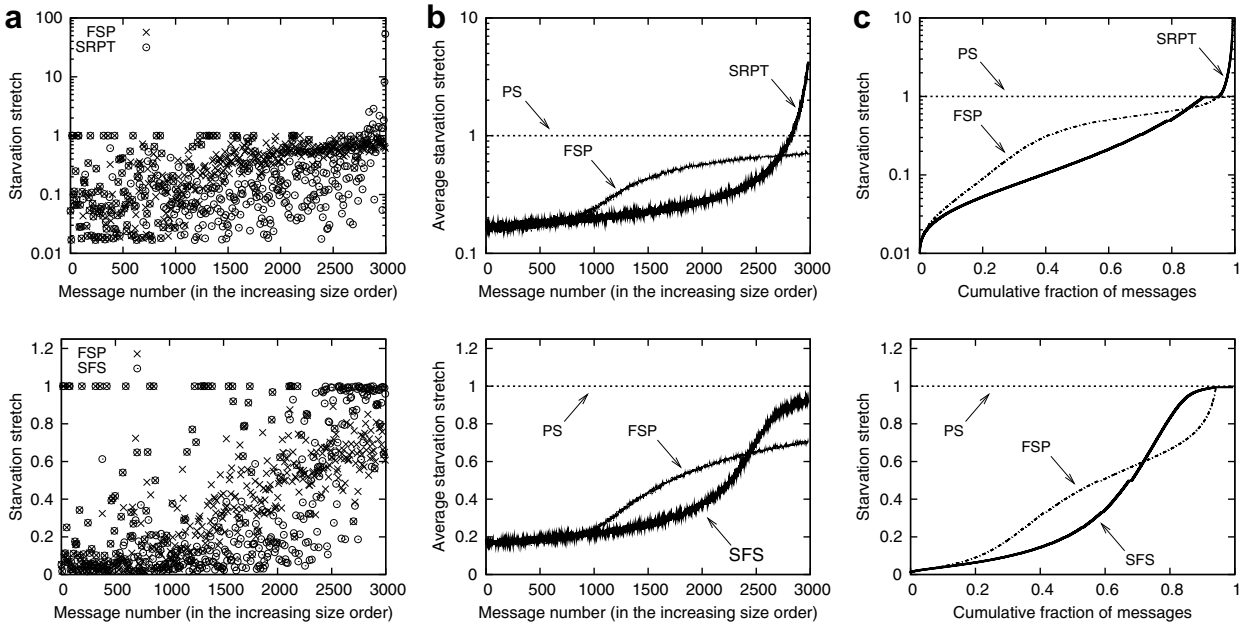

Fig. 3. Fairness of FSP and SFS versus unfairness of SRPT for uniformly distributed message sizes and exponentially distributed interarrival times with 95% load: (a) starvation stretch for every seventh message in a single experiment; (b) average of starvation stretches over 1000 experiments; (c) cumulative distribution of individual starvation stretches from 100 experiments.

The average starvation stretches reported in Fig. 3b blur fates of individual messages. Hence, Fig. 3c plots cumulative distributions of all 300,000 individual starvation stretches in 100 instances of our experiment. Comparison of FSP with SRPT shows that while starvation stretches up to the 85th percentile are higher under the fair FSP, the top 5% of starvation stretches under the unfair SRPT exceed 1, i.e., belong to starved messages. Comparison of SFS with FSP reveals a main divide around 73%. Up to the 73rd percentile, starvation stretches are lower under SFS. Under either SFS or FSP, the top 5% of starvation stretches equal 1. Between the 73rd and 95th percentiles, FSP yields smaller starvation stretches. Similarly to Fig. 3a, lines in Fig. 3c contain horizontal segments at starvation stretches 1, $\frac{1}{2}$, $\frac{1}{3}$, $\frac{1}{4}$, and $\frac{1}{5}$. These flat segments reflect messages that arrive, are completed by PS, and finish under SRPT, FSP or SFS while 0, 1, 2, 3, or 4 other messages remain unfinished.

6.3. Efficiency: decrease of average delay

To evaluate efficiency of the algorithms, we conduct our experiment for various values of load l. We repeat the experiment 1000 times for each examined load $l \leqslant 120\%$, i.e., including all examined instances of underload, but generally less for overloads of $l > 120\%$. Fig. 4a illustrates an intuitive expectation that average delays under SRPT, PS, FSP, and SFS grow as load increases. After load hits and surpasses 100%, the delays remain finite and even decelerate their growth because the number of messages in every experiment is finite. For the extreme of "infinite" load when all 3,000 messages arrive simultaneously, the average delays are analytically expressed in [8]. In particular, PS yields the following average delay in a single experiment with simultaneous message arrivals:

$$D^{\infty}_{\mathrm{PS}} = \frac{\sum_{k=1}^{N}(2(N-k)+1)m_k}{N \cdot C}, \tag{28}$$

where m_k is the size of the kth smallest message, $N = 3000$ is the number of messages, and $C = 10$ Tbps is the link capacity. When the messages arrive simultaneously, SRPT, SFS, and FSP produce an identical transmission schedule for the experiment and achieve the same average delay [8]:

$$D^{\infty}_{\mathrm{SRPT}} = D^{\infty}_{\mathrm{SFS}} = D^{\infty}_{\mathrm{FSP}} = \frac{\sum_{k=1}^{N}(N-k+1)m_k}{N \cdot C}. \tag{29}$$

For the considered uniform distribution of message sizes, we derive the expected average delay under PS as

$$D^{\infty}_{\mathrm{PS}} = \frac{(4N+1)m_{\min} + (2N-1)m_{\max}}{6C} \approx 88,118 \text{ s},$$

where $m_{\min} = 100$ GB and $m_{\max} = 100$ TB are, respectively, minimum and maximum message sizes in the distribution. The expected average delay under SRPT, SFS, and FSP becomes

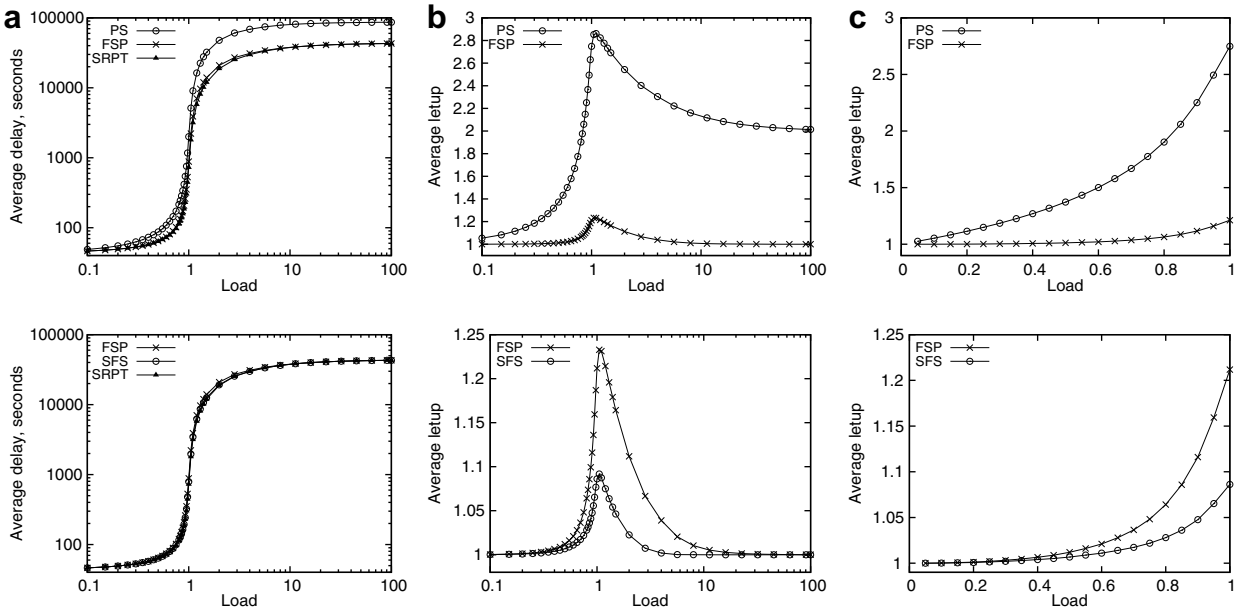

Fig. 4. Efficiency of SRPT, PS, FSP, and SFS with uniformly distributed message sizes and exponentially distributed interarrival times: (a) average delay, (b) average letup, and (c) average letup during underload.

$$D^{\infty}_{\mathrm{SRPT}} = D^{\infty}_{\mathrm{SFS}} = D^{\infty}_{\mathrm{FSP}} = \frac{(N+1)(2m_{\min}+m_{\max})}{6C} \approx 44,081 \text{ s}.$$

Fig. 4a confirms that experimental average delays converge asymptotically to the above analytical predictions.

Fig. 4b plots average letups under PS, FSP, and SFS. All three letups peak around $l = 100\%$. At this load where the arrival rate matches the link capacity, PS, FSP, and SFS have, respectively, 2.8 times, 22%, and 9% larger average delays than SRPT. Asymptotically, the average letup under PS converges to

$$L^{\infty}_{\mathrm{PS}} = \frac{(4N+1)m_{\min}+(2N-1)m_{\max}}{(N+1)(2m_{\min}+m_{\max})} \approx 2,$$

while SFS and FSP converge to the optimal efficiency:

$$L^{\infty}_{\mathrm{SFS}} = L^{\infty}_{\mathrm{FSP}} = 1.$$

In general, SFS provides SRPT-like efficiency with consistently lower average delay than FSP.

While our results for $l > 100\%$ offer interesting insights into behavior of the algorithms in long-term overload conditions, Fig. 4c focuses on underload scenarios $l < 100\%$ which are the most relevant for steady-state operation. Again, SFS consistently outperforms FSP. For example, when load equals 80%, average delays under PS, FSP, and SFS are, respectively, 2 times, 7%, and only 3% worse then the minimum attained under the unfair SRPT.

6.4. Sensitivity to traffic patterns

To examine how the distribution of messages sizes affects performance of fair algorithms, we conduct extra simulations where messages still arrive according to a Poisson process but message sizes follow an exponential, Lomax, or Pareto distribution. As in the uniform distribution above, the average message size is kept 51,250 GB in each of the three new distributions.

The exponential distribution is another representative of scenarios where message sizes are large and have relatively small variance. Fig. 5 reveals that increased diversity of message sizes reduces efficiency of fair algorithms. With the exponentially distributed message sizes, average letup under PS peaks around 4.5. FSP decreases the worst observed average letup to 1.5, which is brought further down to 1.3 by SFS. The efficiency profiles of the three algorithms remain qualitatively the same: the algorithms are most inefficient when load is around 100%, and SFS consistently provides lowest average delay among the fair trio.

The next examined distribution of message sizes is Lomax, also known as Pareto of the second kind. We set the scale and index of the Lomax distribution to 25,625 GB and 1.5, respectively. The distribution represents high-variability settings where the average message size is large, and the number of short messages is significant. Fig. 6 shows that inefficiency of PS is greater than with the exponentially distributed message sizes: worst observed average delay is almost 6 times higher

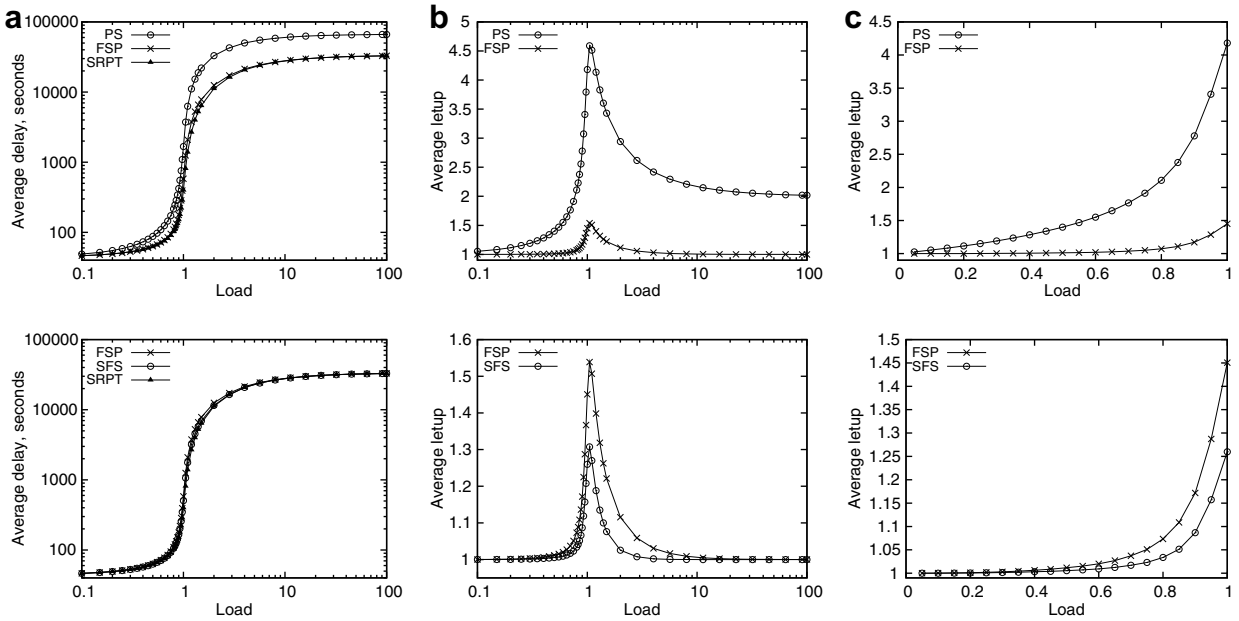

Fig. 5. Efficiency of SRPT, PS, FSP, and SFS with exponential distributions for both message sizes and interarrival times: (a) average delay, (b) average letup, and (c) average letup during underload.

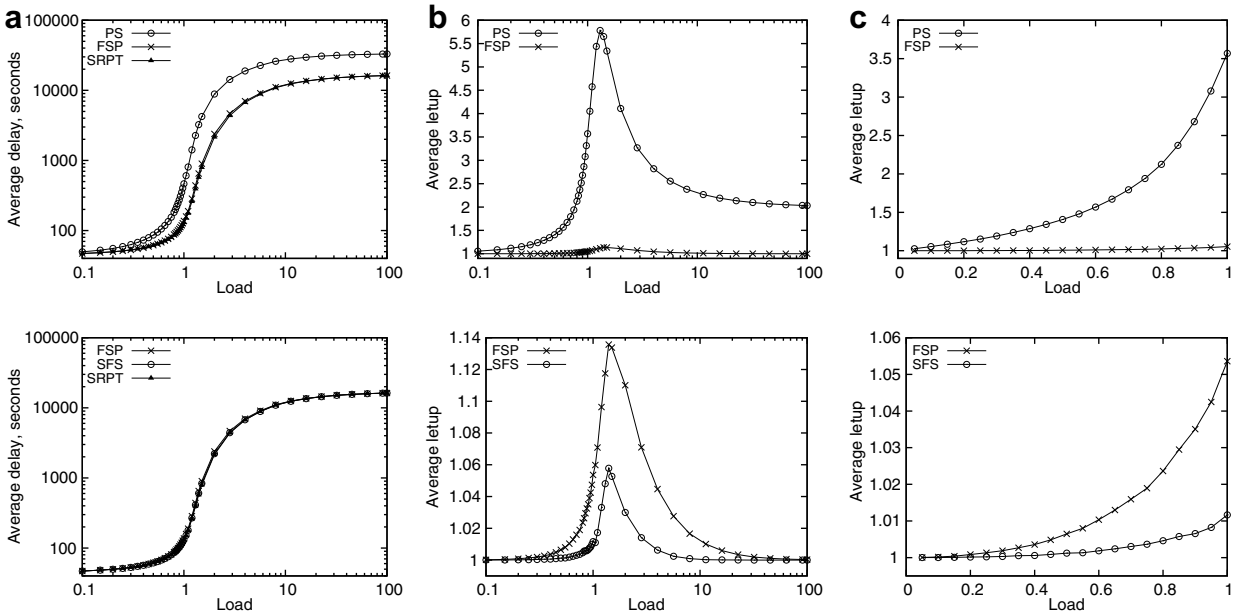

Fig. 6. Efficiency of SRPT, PS, FSP, and SFS with Lomax-distributed message sizes and exponentially distributed interarrival times: (a) average delay, (b) average letup, and (c) average letup during underload.

than under SRPT. On the other hand, FSP and SFS benefit from the significant presence of short messages. Average delay under FSP and SFS is only 14% and 6% higher, respectively, than the SRPT minimum.

Our second source of highly variable message sizes is the Pareto distribution with the scale of 17,083 GB and index of 1.5. With the fraction of short messages being less prominent than in the Lomax distribution, the Pareto pattern is the most cruel to PS. Fig. 7 demonstrates that average letup under PS peaks around 7.5. Interestingly, these are also the settings where unfairness of SRPT is less of a problem. Our experiments confirm the surprising finding by Bansal and Harchol-Balter [1] that even under heavy load with Pareto-distributed message sizes, SRPT starves only few messages and does not inflate starvation

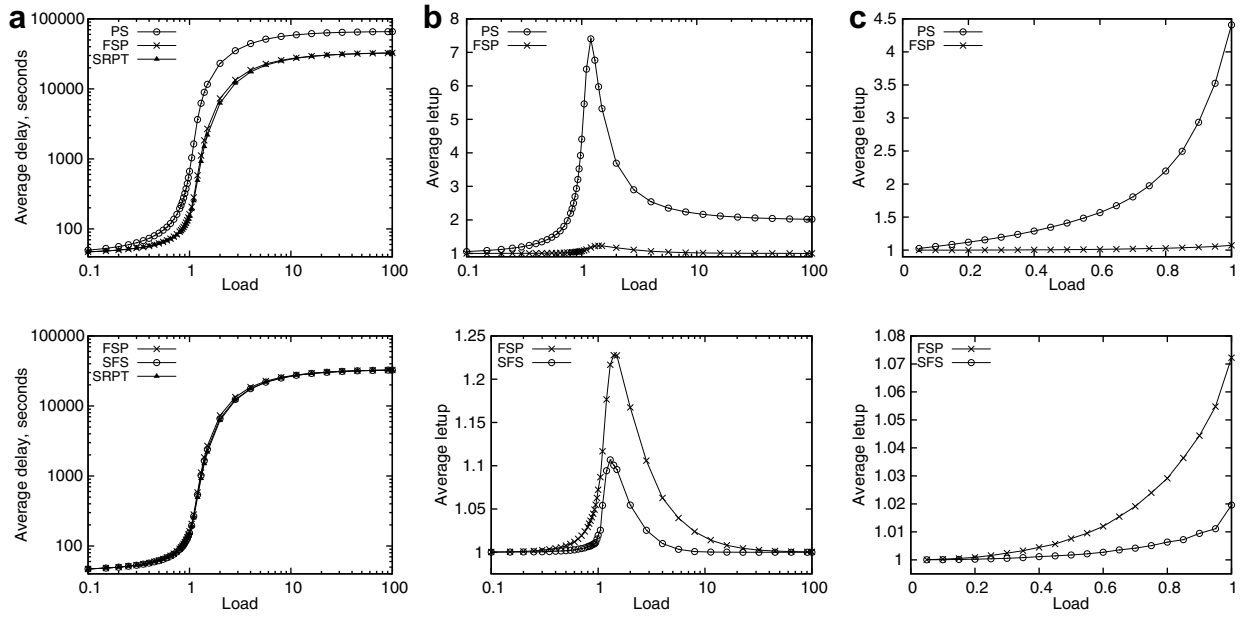

Fig. 7. Efficiency of SRPT, PS, FSP, and SFS with Pareto-distributed message sizes and exponentially distributed interarrival times: (a) average delay, (b) average letup, and (c) average letup during underload.

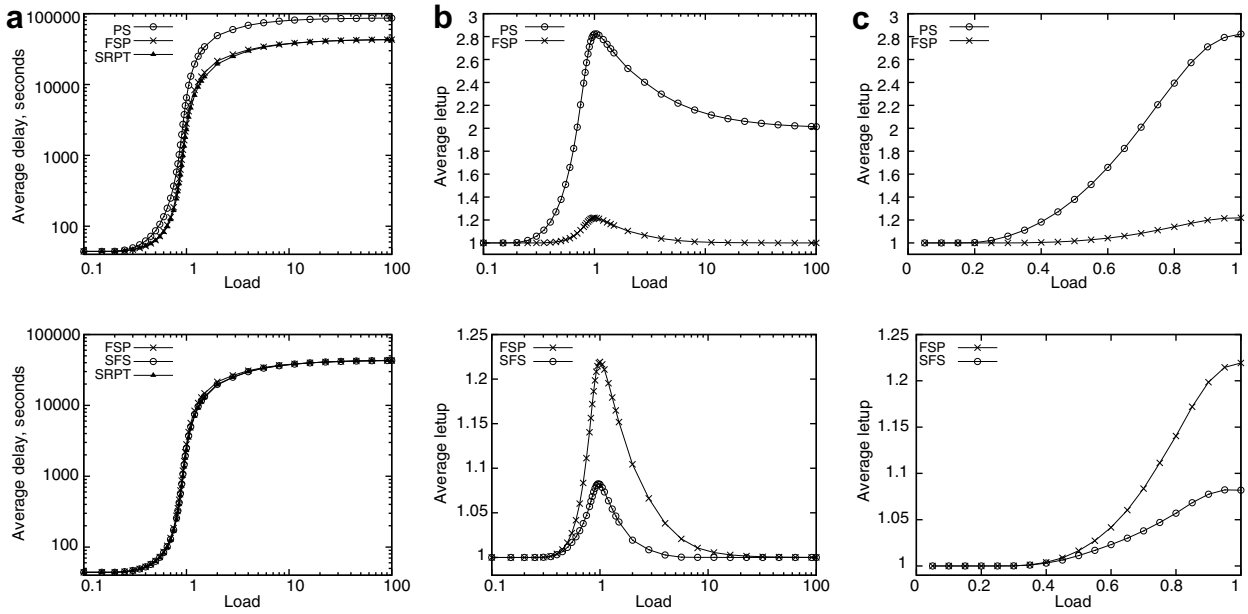

Fig. 8. Efficiency of SRPT, PS, FSP, and SFS with uniformly distributed message sizes and Pareto-distributed interarrival times: (a) average delay, (b) average letup, and (c) average letup during underload.

stretch much beyond 1. In comparison to PS, efficiency of the fair FSP and SFS is far superior: average delay is at most 23% and 11% worse, respectively, than the minimum achieved by SRPT.

Finally, we experiment with different patterns of message interarrival times. Unlike with message sizes, the distribution of interarrival times does not affect efficiency of fair algorithms substantially. Fig. 8 plots average delay and letup under PS, FSP, and SFS for uniformly distributed message sizes and Pareto-distributed interarrival times with Pareto index 1.5. The results strongly resemble the average delay and letup reported for exponentially distributed interarrival times in Fig. 4. The main deviation seems to lie in reaction to load changes. With Pareto-distributed interarrival times, average letup grows

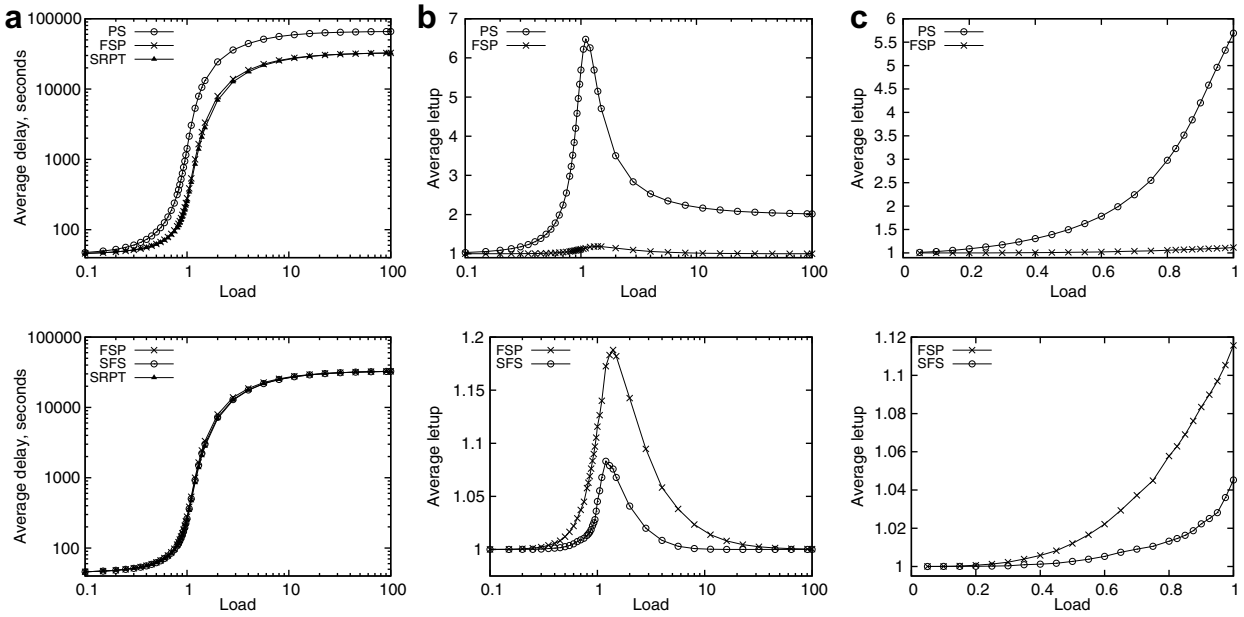

Fig. 9. Efficiency of SRPT, PS, FSP, and SFS with Pareto distributions for both message sizes and interarrival times: (a) average delay, (b) average letup, and (c) average letup during underload.

and declines slower while approaching and passing a comparable peak value around the 100% load. For Pareto-distributed message sizes, comparison of Figs. 7 and 9 reveals that switching from the exponential to Pareto distribution of interarrival times has a similar impact of widening and slightly shortening the peaks of average letup under PS, FSP, and SFS.

Regardless of the distributions chosen for message sizes and interarrival times, the efficiency profiles of the fair algorithms are qualitatively the same. The ability of PS, FSP, and SFS to compete with SRPT is the weakest when load is about 100%. Under large overload conditions, average letup under PS converges to around 2 while FSP and SFS operate like SRPT and support SRPT-like efficiency. Our simulations show consistently that PS and SFS are, respectively, the least and most efficient among the evaluated fair algorithms.

7. Conclusions

In this paper, we explored the class of fair algorithms for message communication where fairness means that no message is delivered later than under PS. In addition to PS, the fair class includes FSP and newly proposed PFSP, OFSP, and SFS. We introduced a slack system to characterize fair algorithms completely and proved that a fair online algorithm does not guarantee minimal average delay achievable with fairness. Our analysis also revealed lower bounds on worst-case inefficiency of fair algorithms. Our extensive simulations for various distributions of message sizes and arrival times showed that SFS supports SRPT-like efficiency and consistently provides much smaller average delay than PS and FSP. The simulations demonstrated that FSP and SFS gain their efficiency improvement over PS across the whole spectrum of message sizes, including long messages but primarily due to dramatic delay reduction for short messages. SFS outperforms FSP by decreasing delay for midsize messages.

Our experimental study focused on SFS. A further study needs to compare efficiency of SFS, PFSP, and OFSP with the optimal efficiency achievable by a fair offline algorithm. If the efficiency gap is significant, an interesting approach to narrowing the gap is to predict properties of future messages from statistical information about past messages.

While link scheduling can affect location of bottlenecks in an arbitrary network topology, our focus on one link was clearly an excessive simplification. Future studies should tackle the harder general problem and also address diversity of applications. First, our analysis ignored propagation, node processing, error recovery and other delays dominated by bottleneck transmission delays for long messages. For shorter messages, the extra delays contribute more to overall delay and thereby reduce the relative gains from the efficient utilization of the bottleneck link. To handle a more complex model, we will learn from prior research on message-grained transmission over packet-switching networks [4,12]. Second, some applications are interested in other network performance metrics than minimal delay achievable under current load. We are designing an integrated allocation framework where a fair service minimizes average message delay, and the other services support applications with different performance objectives.

Acknowledgements

We are grateful to the US National Science Foundation for supporting this work financially through grants CNS-0626661, ANI-9730162, ITR-0325453, and DMI-0400287. Gavin Hurley played an important role in early stages of the research and contributed greatly to development of the presented ideas. We also thank Walter Willinger for his useful suggestions on conducting the experimental studies.

References

[1] N. Bansal, M. Harchol-Balter, Analysis of SRPT scheduling: investigating unfairness, in: Proceedings ACM SIGMETRICS, 2001.
[2] M.A. Bender, S. Chakrabarti, S. Muthukrishnan, Flow and stretch metrics for scheduling continuous job streams, in: Proceedings ACM-SIAM SODA, 1998.
[3] D. Bertsekas, R. Gallager, Data Networks, Prentice-Hall, 1987.
[4] L. Cherkasova, Scheduling strategy to improve response time for web applications, in: Proceedings High-performance Computing and Networking (HPCN Europe), 1998.
[5] A. Demers, S. Keshav, S. Shenker, Analysis and simulation of a fair queueing algorithm, in: Proceedings ACM SIGCOMM, 1989.
[6] E.J. Friedman, S.G. Henderson, Fairness and efficiency in web server protocols, in: Proceedings ACM SIGMETRICS, 2003.
[7] S. Gorinsky, C. Jechlitschek, Fair efficiency, or low average delay without starvation, in: Proceedings International Conference on Computer Communications and Networks (ICCCN), 2007.
[8] S. Gorinsky, N.S.V. Rao, Dedicated channels as an optimal network support for effective transfer of massive data, in: Proceedings IEEE INFOCOM, 2006.
[9] P. Goyal, H. Vin, H. Cheng, Start-time fair queueing: a scheduling algorithm for integrated services packet switching networks, in: Proceedings ACM SIGCOMM, 1996.
[10] V. Jacobson, Congestion avoidance and control, in: Proceedings ACM SIGCOMM, 1988.
[11] C. Jechlitschek, S. Gorinsky, Simulation Suite for PS, SRPT, FSP, and SFS, December 2007, <http://www.arl.wustl.edu/~gorinsky/efficient_fair_algorithms>.
[12] E. Modiano, Scheduling packet transmissions in a multi-hop packet switched network based on message length, in: Proceedings International Conference on Computer Communications and Networks (ICCCN), 1997.
[13] R. Motwani, S. Phillips, E. Torng, Nonclairvoyant scheduling, Theoretical Computer Science 130 (1) (1994) 17–47.
[14] C. Phillips, C. Stein, J. Wein, Scheduling jobs that arrive over time, in: Proceedings 4th International Workshop on Algorithms and Data Structures (WADS), LNCS, Springer-Verlag, 1995.
[15] L.E. Schrage, A proof of the optimality of the shortest remaining processing time discipline, Operations Research 16 (3) (1968) 687–690.
[16] M. Shreedhar, G. Varghese, Efficient fair queueing using deficit round robin, in: Proceedings ACM SIGCOMM, 1995.
[17] D.R. Smith, A new proof of the optimality of the shortest remaining processing time discipline, Operations Research 26 (1) (1978) 197–199.
[18] A. Wierman, M. Harchol-Balter, Bounds on a fair policy with near optimal performance, Tech. Rep. CMU-CS-03-198, Carnegie Mellon University, November 2003.