On Fundamental Tradeoffs between Delay Bounds and Computational Complexity in Packet Scheduling Algorithms

Jun (Jim) Xu and Richard J. Lipton

***Abstract*— In this work, we clarify, extend and solve a long-standing open problem concerning the computational complexity for packet scheduling algorithms to achieve tight end-to-end delay bounds. We first focus on the difference between the time a packet finishes service in a scheduling algorithm and its virtual finish time under a GPS (General Processor Sharing) scheduler, called *GPS-relative delay*. We prove that, under a slightly restrictive but reasonable computational model, the lower bound computational complexity of any scheduling algorithm that guarantees $O(1)$ GPS-relative delay bound is $\Omega(\log n)$. We also discover that, surprisingly, the complexity lower bound remains the same even if the delay bound is relaxed to $O(n^a)$ for $0 < a < 1$. This implies that the delay-complexity tradeoff curve is flat in the "interval" $[O(1), O(n))$. We later conditionally extend both complexity results (for $O(1)$ or $O(n^a)$ delay) to a much stronger computational model, the linear decision tree. Finally, we show that the same complexity lower bounds are conditionally applicable to guaranteeing tight end-to-end delay bounds, if the delay bounds are provided through the Latency Rate (LR) framework.**

***Index Terms*— Computational complexity, packet scheduling, Quality of Service, delay bound, decision tree.**

I. Introduction

Packet scheduling is an important mechanism in providing QoS guarantees in data networks [1], [2], [3]. The fairest algorithm for packet scheduling is General Processor Sharing (GPS) [1], [4]. However, GPS is not a realistic algorithm since in a packet network, service is performed packet-by-packet, rather than bit-by-bit as in GPS. Nevertheless, GPS serves as a reference scheduler that real-world packet-by-packet scheduling algorithms (e.g., WFQ [1]) can be compared with in terms of end-to-end delay bounds and fair bandwidth allocation.

Let n be the number of active sessions in a link of rate r served by a GPS scheduler. Each session $i = 1, 2, \cdots, n$ is assigned a weight value ϕ_i. Each backlogged[1] session j at every moment t is served simultaneously at rate $r_j = r\phi_j / (\sum_{j \in B(t)} \phi_j)$, where $B(t)$ is the set of sessions that are backlogged at time t. One important property of GPS, proved in [4], is that it can guarantee tight end-to-end delay bound to traffic that is leaky-bucket [5] constrained.

It is interesting to look at the *GPS-relative delay* of a packet served by a scheduling algorithm ALG as compared to GPS. For each packet p, it is defined as $max(0, F_p^{ALG} - F_p^{GPS})$, where F_p^{ALG} and F_p^{GPS} are the times when the packet p finishes service in the ALG scheduler and in the GPS scheduler, respectively. It has been shown in [4] and [6] respectively that WFQ (Weighted Fair Queuing) and WF^2Q (Worst-case Fair Weighted Fair Queuing) schedulers both have a worst-case GPS-relative delay bound of $\frac{L_{max}}{r}$, where L_{max} is the maximum packet size in the network. That is, for each packet p,

$$F_p^{WFQ} - F_p^{GPS} \leq \frac{L_{max}}{r} \quad (1)$$

$$F_p^{WF^2Q} - F_p^{GPS} \leq \frac{L_{max}}{r} \quad (2)$$

We simply say that the delay bound is $O(1)$ since L_{max} and r can be viewed as constants independent of the number of sessions n. WFQ and WF^2Q achieve this $O(1)$ delay bound by (a) keeping perfect track of the GPS clock and (b) picking among all (in WFQ) or all eligible (in WF^2Q) head-of-line (HOL) packets, the one with the smallest GPS virtual finish time to serve next. The per-packet worse-case computational complexity of the second part ((b) part) in both WFQ and WF^2Q is $O(\log n)$. In other words, the computational cost[2] to "pay" for the $O(1)$ GPS-relative delay bound in both WFQ and WF^2Q is $O(\log n)$.

On the other hand, round-robin algorithms such as DRR (Deficit Round Robin) [7] and WRR (Weighted Round Robin) [8] have a low implementation cost of $O(1)$. However, they in general cannot provide the tight GPS-relative delay bound of $\frac{L_{max}}{r}$. In fact, the best possible delay bound they can provide is $O(n)$. This is illustrated in Fig. 1. We assume that these n sessions share the same link and are of same weight. Without loss of generality, we also assume that these sessions are served in the round-robin order $1, 2, \cdots, n$. At time 0, packets of length M have arrived at sessions $1, 2, \cdots, n-1$, and a packet of length $m < M$ has arrived at session n. Suppose M is no larger than the *service quantum* size used in round-robin algorithms so that all these packets are in the same service frame. Then clearly the short packet in session n will be served behind $n-1$ long packets. The GPS-relative delay of the short packet can be calculated as $\frac{(n-1)(M-m)}{r}$, which is $O(n)$.

Manuscript received on 12/1/2002; revised on 12/3/2003; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor I. Stavrakakis. Xu is supported in part by NSF Grant ITR/SY ANI-0113933 and NSF CAREER award ANI-0238315. Lipton is supported in part by NSF Grants ITR/SY ANI-0113933 and CCR-0002299, and by Telcordia Research. A preliminary version of the paper has been presented at ACM Sigcomm'2002.

Jun (Jim) Xu is with the College of Computing, Georgia Institute of Technology, Atlanta, GA 30332, USA (e-mail: jx@cc.gatech.edu).

Richard J. Lipton is with the College of Computing, Georgia Institute of Technology, Atlanta, GA 30332, USA (e-mail: rjl@cc.gatech.edu).

[1] We say that a flow f is backlogged at time t (under a scheduler) if there is a positive amount of flow f traffic in the service queue at time t.

[2] Here the cost of the GPS clock tracking ((a) part) is not included.

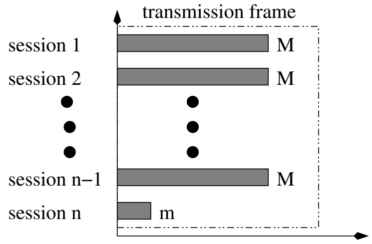

Fig. 1. How round robin algorithms incur $O(n)$ GPS-relative delay

We have just shown that algorithms with $O(\log n)$ complexity (GPS time tracking overhead excluded) such as WFQ and WF^2Q can provide $O(1)$ GPS-relative delay bound, while $O(1)$ round-robin algorithms such as DRR and WRR can only guarantee a delay bound of $O(n)$. A long-standing open problem in the QoS community is whether this represents indeed the fundamental tradeoff between computational complexity of the scheduling algorithms and the GPS-relative delay bound they can achieve. This problem was been posed again in Sigcomm'01 by Guo (author of [9]). Our work formally defines and clarifies this open problem, and solves it in a comprehensive way.

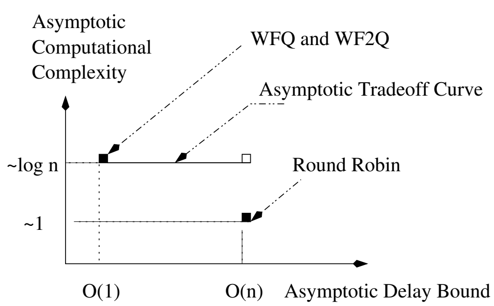

Fig. 2. The asymptotic tradeoff curve between delay bound and computational complexity

The first major result of this paper is to show that $\Omega(\log n)$ is indeed the complexity lower bound to guarantee $O(1)$ GPS-relative delay[3], excluding the cost of tracking GPS time. This bound is established under the decision tree computation model that allows direct comparisons between its inputs, which, in our context, is equivalent to allowing comparisons between GPS finish times of the packets. This model seems slightly restrictive but is reasonable for our context, since such comparisons are indeed sufficient for assuring $O(1)$ GPS-relative delay bound in WFQ and WF^2Q [4], [6]. This result granted for the moment, we now have two points on the complexity-delay tradeoff curve, as shown in Fig. 2. One is the $O(n)$ delay at the complexity of $\Omega(1)$ and the other is the $O(1)$ delay at the complexity of $\Omega(\log n)$. One interesting question to ask is how other parts of the "tradeoff curve" look. More specifically, to guarantee a delay bound that is asymptotically between $O(1)$ and $O(n)$, say $O(\sqrt{n})$, can the complexity of packet scheduling be asymptotically lower than $\Omega(\log n)$, say $\Omega(\sqrt{log_2 n})$? The result we discover and prove is surprising: for any fixed $0 < a < 1$, the asymptotic complexity for achieving $O(n^a)$ delay is always $\Omega(log_2 n)$. As shown in Fig. 2, this basically says that the asymptotic tradeoff curve is flat and has a jump at $O(n)$.

The second major result of this paper is to strengthen the aforementioned lower bounds by extending them to a much stronger computational model: decision tree that allows *linear comparisons*. However, under this computational model, we are able to prove the same complexity lower bounds of $\Omega(\log n)$ only when the scheduling algorithm guarantees $O(1)$ or $O(n^a)$ $(0 < a < 1)$ *disadvantage delay* bound. *Disadvantage delay* is a slightly stronger type of delay than the GPS-relative delay, since for each packet, its *disadvantage delay* is no smaller than its GPS-relative delay. Nevertheless, the second result is provably stronger than our first result for both $O(1)$ and $O(n^a)$ cases.

Our third and final result is to show that the same complexity lower bounds can be extended to guaranteeing tight end-to-end delay bounds, if the delay bounds are provided through the Latency Rate (LR) framework (introduced in [11]). In particular we show that, the minimum complexity for an LR scheduler to provide a tight latency of $O(n^a)\frac{L_{max}}{r} + \frac{L_{max,i}}{r_i}$ $(0 \le a < 1)$ is $\Omega(\log n)$ per packet, where $L_{max,i}$ is the maximum size of a packet in session i and r_i is the guaranteed rate of session i. This result is important since most of existing scheduling algorithms that provide tight end-to-end delay bounds are LR schedulers with latency $O(1)\frac{L_{max}}{r} + \frac{L_{max,i}}{r_i}$.

Though it is widely believed as a "folklore theorem" that scheduling algorithms which can provide tight end-to-end delay bounds require $\Omega(\log n)$ complexity (typically used for maintaining a priority queue), it has never been carefully formulated and proved. To the best of our knowledge, our work is the first major and successful step in establishing such complexity lower bounds. Our initial goal was to show that the $\Omega(\log n)$ delay bounds hold under the decision tree model that allows linear comparisons. Though we are not able to prove this result in full generality, our rigorous formulation of the problem and techniques introduced in proving slightly weaker results serve as the basis for further exploration of this problem.

The rest of the paper is organized as follows. In Section II, we introduce the computational models and assumptions we will use in proving our results. The aforementioned three major results are established in Sections III, IV, and V respectively. Section VI discusses related work, including our recent progress on establishing the complexity lower bounds of tracking GPS clock. Section VII concludes the paper.

II. Assumptions and Computational Models

In general, complexity lower bounds of a computing problem are derived based on problem-specific assumptions and conditions, and a computational model that specifies what operations are allowed in solving the problem and how they are "charged" in terms of complexity. In Section II.A, we describe a network load and resource allocation condition called CBFS (continuously backlogged fair sharing) under which all later lower

[3] Leap Forward Virtual Clock ($LFVC$) scheduler [10] has a low implementation complexity of $O(\log(\log n))$ using timestamp discretization, but may incur $O(n)$ GPS-relative delay in the worst case. This is because, with small but positive probability, the "discretization error" may add up rather than cancel out.

bounds will be derived. In Section II.B, we introduce two computational models that will be used in Sections III and IV, respectively. Finally in Section II.C, we discuss why decision tree computational models are chosen for studying complexity lower bounds.

A. CBFS condition

All lower bounds in this paper will be derived under a network load and resource sharing condition called *continuously backlogged fair sharing* (CBFS). Let n be the number of sessions and r be the total bandwidth of the link. In CBFS,

- (Fair Sharing) Each session has equal weight, that is, for any $1 \leq i < j \leq n$, $\phi_i = \phi_j$.
- (Continuously Backlogged) Each session has a packet arrival at time 0. Also, for any $t > 0$ and $1 \leq i \leq n$, $A_i(t) \geq \frac{r}{n}t$. Here $A_i(t)$ is the amount of session i traffic that has arrived during the interval $[0, t]$.

We call the second part of the condition "continuously backlogged" because if these sessions are served by a GPS scheduler, they will be continuously backlogged from time 0. This is proved in the next proposition. In the sequel, we assume that a packet arrival at time t is an impulse, the value of which is the size of the packet, at time t. This is equivalent to the assumption used in [4] that the input link speeds are infinite. This assumption is widely used in QoS literature.

Proposition 1: For any packet arrival instance that conforms to the CBFS condition, each and every session will be continuously backlogged when served by a GPS scheduler.

Proof: The proof is by contradiction. Suppose some sessions may become unbacklogged at certain points of time. We can view packet scheduling as an event-driven system in which the events are the arrivals and departures of the packets. Since all sessions are backlogged at time 0, the following is the only possible way that session i may become unbacklogged at time t: a packet departs from session i at time t, and its next packet does not arrive until time $\tau > t$ ($\tau = \infty$ if there is no such arrival). Let t^* be the time that the *earliest* such packet departure event happens. Suppose this happens to session i^*, and session i^* does not become backlogged again until $\tau^* > t^*$. By the definition of t^*, all sessions are continuously backlogged between $[0, t^*]$. So, under the GPS scheduler, the amount of service each session receives during this period is the same, which is $\frac{r}{n}t^*$. Let $\tau^* > t' > t^*$ (to avoid the case $\tau^* = \infty$). Then the amount of service session i^* receives during the interval $[0, t']$ is $\frac{r}{n}t^* < \frac{r}{n}t'$, which violates the second part of the CBFS condition. ■

Since our lower bounds are on the computational complexity in the worst case, the general lower bounds can only be **higher than or equal to** the bounds derived under the CBFS condition (i.e., we don't gain from this condition). The significance of this condition is profound:

- First, computing the GPS virtual finish time of a packet p becomes an $O(1)$ operation (see remark after Proposition 2). So CBFS condition allows us to naturally exclude the cost of tracking GPS clock so that we do not need to answer the question "What is the computational complexity of tracking the GPS clock?". This is quite a blessing since this question is itself open in 2002 and was only recently settled in [12] (to be discussed in Sec. VI).
- Second, we will show that under the CBFS condition, many existing scheduling algorithms such as Virtual Clock (VC) [13], Frame-based Fair Queuing (FFQ) [14] and WF^2Q+ [15] are equivalent to either WFQ or WF^2Q (Proposition 3). So whenever we need to relate our results to these scheduling algorithms, we only need to study WFQ and WF^2Q.
- Third, the complexity lower bounds that are proved under this condition are still tight enough. In other words, we are not "losing" too much ground on complexity lower bounds when restricted by this condition.

In our later proofs, we assume that the size of a packet can take any real number between 0 and L_{max}, where L_{max} denotes the maximum packet size. This is, in general, not true for packet networks. However, it can be shown that if we remove part one (fair sharing) of the CBFS condition and instead allow weighted sharing (with part two adjusted accordingly), we do not need to insist on such freedom in packet size. In fact, our proofs will work even for ATM networks where fixed packet size is used. Since this proof is less interesting, we omit it here to save space.

In the rest of this section, we prove that the computation of GPS virtual finish times is an O(1) operation under the CBFS condition, and state without proof that a few existing algorithms (VC, WF^2Q+, FFQ) are equivalent to either WFQ or WF^2Q under the CBFS condition.

Definition 1: We say that two scheduling algorithms are *equivalent* under a condition $\mathcal{C}$ if given any arrival instance conforming to $\mathcal{C}$, these two algorithms will generate the same packet service schedule.

Notation 1: For the $k'th$ packet in session i, let $L_{i,k}$, $T_{i,k}$, and $F_{i,k}$ denote its length, arrival time, and GPS virtual finish time, respectively. Let $V(t)$ denote the GPS virtual time as a function of real time t.

Proposition 2: Under the CBFS condition,

- (a) $F_{i,k} = F_{i,k-1} + \frac{L_{i,k}}{r_i}$, $1 \leq i \leq n$ and $k > 0$. Here we let $F_{i,0} = 0$ by definition.
- (b) $V(t) \equiv t$

Proof: (a) In GPS,

$$F_{i,k} = max(F_{i,k-1}, T_{i,k}) + \frac{L_{i,k}}{r_i} \tag{3}$$

It is clear that $T_{i,k} \leq F_{i,k-1}$, since otherwise, during the time period $[F_{i,k-1}, T_{i,k}]$ the session i is idle under GPS, violating the *continuously backlogged* (Proposition 1) property of CBFS. Therefore the formula (3) becomes $F_{i,k} = F_{i,k-1} + \frac{L_{i,k}}{r_i}$.

(b) Recall that $V(t)$ is defined as follows:

$$V(0) = 0 \tag{4}$$

$$V(t+\tau) = V(t) + r\tau / (\sum_{i \in B(t)} r_i) \tag{5}$$

where $B(t)$ is the set of sessions that are active during the interval $[t, t+\tau]$. Here $t+\tau$ can be anytime before the occurrence of the first *event* (the arrival or departure of a packet) after t. Since all sessions are backlogged all the time under GPS (Proposition 1), $B(t)$ is exactly the set of all sessions. Therefore, $\sum_{i \in B(t)} r_i = r$ and consequently (5) becomes $V(t+\tau) = V(t) + \tau$. This, combined with (4), implies that $V(t) \equiv t$. ■

Remark: It is clear from (a) that the calculation of GPS virtual finish time is an $O(1)$ operation (under the CBFS condition) per packet, as the program can store the result of $F_{i,k-1}$ from the prior computation. The (b) part would be used in the proof of the following proposition. However, due to the lack of space, we omit its proof here, which can be found in [16].

Proposition 3: Under the CBFS condition, Virtual Clock (VC) [13] and Frame-based Fair Queuing (FFQ) [14] are equivalent to WFQ, and WF^2Q+ [15] is equivalent to WF^2Q.

B. Decision tree models

We adopt a standard and commonly-used computational model in proving lower bounds: the *decision tree*. A decision tree program in general takes as input a list of real variables $\{x_i\}_{1 \le i \le n}$. Each internal and external (leaf) node of the tree is labeled with a predicate of these inputs. The algorithm starts execution at the root node. When the program control is centered at any internal node, the predicate labeling that node is evaluated, and the program control is passed to its left or right child when the value is "yes" or "no" respectively. Before the control is switched over, the program is allowed to execute *unlimited number* of sequential operations such as data movements and arithmetic operations. In particular, the program is allowed to store all results (i.e., no constraint on storage space) from prior computations. When program control reaches a leaf node, the predicate there is evaluated and its result is considered as the output of the program. The complexity of such an algorithm is defined as the depth of the tree, which is simply the number of predicates that needs to be evaluated in the *worst case*. Fig. 3 shows a simple decision tree with six nodes. Each P_i ($1 \le i \le 6$) is a predicate of the inputs.

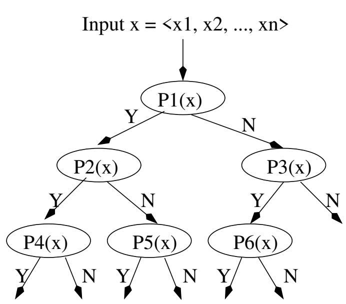

Fig. 3. Decision tree computational model

The decision tree was originally proposed for *decision problems*, in which the output is binary: simply "yes" or "no". The model can be extended to handling more general problems the output of which is not necessarily binary. For example, in the context of this work, the output will be the sequence in which packets get scheduled.

Allowing different types of predicates to be used in the decision tree results in models of different computational powers. On the one extreme, if the decision tree program allows the magic predicate $P(x_1, x_2, \cdots, x_n)$ that exactly solves the problem, then the complexity of the problem is precisely 1. On the other extreme, if the decision tree program only allows constant predicates, then nontrivial (nonconstant) decision problems are simply not solvable under this model, no matter how much computation is performed. In this work, we consider predicates that are reasonable in the sense that existing scheduling algorithms are able to provide $O(1)$ GPS-delay bounds using only such predicates.

The first computational model we consider is the decision tree that allows comparisons only between its inputs. It has been used in proving the $\Omega(n \log n)$ lower bound for comparison-based sorting algorithms [17]. This model will be used in proving our lower bounds in Sections III and V. In the context of this work, the inputs will be the lengths and the arrival times of the packets. This model is reasonable for the instances used in these proofs because we will show that under the CBFS condition, allowing comparisons between inputs is equivalent to allowing comparisons between GPS virtual finish times of the packets in these instances. Since both WFQ and WF^2Q are able to provide $O(1)$ GPS-relative delay bounds using such comparisons only, this model is not restrictive.

The second computational model we introduce is the decision tree that allows *linear tests* [18]. In this model, each predicate allowed by the decision tree is in the form of "$h(x_1, x_2, \cdots, x_n) \ge 0$?", where h is a linear function (defined below) of the inputs $\{x_i\}_{1 \le i \le n}$.

Definition 2—Linear Function: A linear function f of the variables $\{x_i\}_{1 \le i \le n}$ is defined as $f(x_1, x_2, \cdots, x_n) = \sum_{i=1}^{n} a_i x_i + a_0$, where $\{a_i\}_{0 \le i \le n}$ are real constants.

The second model is strictly stronger than the first model since the set of predicates that are allowed in the second model is a proper superset of what is allowed in the first model. The second model will be used in our proofs in Section IV. It is necessary to introduce this stronger model since more sophisticated (hypothetical) packet scheduling algorithms might involve comparisons other than between GPS finish times, although no such comparisons were used in existing algorithms.

Under the CBFS condition, the linear decision tree is practical in our context in the sense that many existing scheduling algorithms, including WFQ, VC, FFQ, WF^2Q, and WF^2Q+, use only the operations allowed in the model. Due to Proposition 3, under the CBFS condition, we only need to consider WFQ and WF^2Q. Note that in both WFQ and WF^2Q, (1) GPS time estimation is an O(1) operation and does not require branching statements under the CBFS condition (see remark after Proposition 2), and (2) comparisons between virtual finish times (shown to be the linear functions of the inputs) are all that is needed in making scheduling decisions. Careful readers would point out that WF^2Q also involves comparisons with virtual start times. However, note that under the CBFS condition, the virtual start time of a packet is exactly the virtual finish time of the previous packet in the same session. In summary, both computational models are practical and nonrestrictive, in the sense that they are actually being used by existing scheduling algorithms.

C. Remarks on the decision tree model

A decision tree program allowing certain branching predicates is computationally stronger than a computer program that allows the same types of branching predicates and is memory-constrained. This is because (1) the decision tree can be totally different when the size of input changes (so-called *nonuniform circuit*), and (2) the computational complexity counted in the decision tree model is only the *depth* of the tree, not the *size* of the tree. Neither is true about a regular computer program. So a tight lower bound derived under the decision tree model may not be reachable by a computer program. For example, Knapsack[4], a well-known NP-complete problem, has an $O(n^5 \log^2 n)$ algorithm[5] in the decision tree model that allows linear comparisons [19]. Despite the fact that a decision tree algorithm can be computationally stronger than a computer program, when allowing the same branching predicates, many lower bound proofs are based on decision tree. This is because (1) they provide powerful tools for proving lower bounds, and (2) so far there is no model that exactly captures the computational power of a computer program and at the same time provides such powerful tools.

III. Complexity–Delay Tradeoffs when Allowing Comparisons between Inputs

In this section, we prove that if only comparisons between inputs are allowed, the complexity to assure $O(1)$ or $O(n^a)$ $(0 < a < 1)$ GPS-relative delay bound is $\Omega(\log n)$. In Section III.A, we introduce two general lemmas used in later proofs. Sections III.B and III.C proves the $\Omega(\log n)$ complexity lower bounds for the case of $O(1)$ and $O(n^a)$ respectively.

A. Preliminaries

In this section, we state and prove some lemmas that will be used in later proofs. We first state a well-known complexity lower bound for comparison-based sorting [17]. It is clear from the proof that this lower bound holds even if all the real numbers are between two numbers m and M $(0 \leq m < M)$.

Lemma 1—sorting lower bound [17]: To sort a set of n numbers $\{x_i\}_{1 \leq i \leq n}$ using only comparisons between them, requires $n \log_2 n - o(n \log_2 n)$ steps in the worst case.

Proof: [Sketch] We only summarize the main idea from the proof that can be found in several algorithm textbooks, including [17]. A sorting algorithm can be modeled as a binary decision tree similar to the one shown in Fig. 3. This tree consists of $n!$ leaves, corresponding to $n!$ possible ordering of the n numbers to sort. Each comparison corresponds to a tree node and has up to two children corresponding to further actions taken based on the result of comparison (clearly binary). It can be shown from the Stirling's formula that the longest path of this binary tree must be no shorter than $n \log_2 n - o(n \log_2 n)$, which corresponds to the worst-case execution time. ■

Lemma 1 is sufficient for proving the lower bounds (when allowing direct comparisons between inputs) for scheduling throughout this section. However, to prove stronger results (when allowing linear tests) in Section IV, we need to resort to a stronger lemma (Lemma 3). Since the steps for proving stronger lower bounds in Section IV can be reused for proving the weaker results in this section, for the overall succinctness of the proofs, proofs in this section will also be based on Lemma 3 (stronger version) instead of Lemma 1 (weaker version).

Definition 3: A *set membership problem* is to determine whether the input $\{x_i\}_{1 \leq i \leq n}$, viewed as a point $(x_1, x_2, \cdots, x_n)$ in the Euclidean space R^n, belongs to a set $L \subseteq R^n$.

Next, we state a general lemma concerning the complexity of set membership problems (defined above) under the decision tree model that allows linear tests. This lemma, due to Dobkin and Lipton [18], has been used extensively in lower bound proofs (e.g., [20]). In complexity theory, the lower bound for solving a set membership problem is closely related to the geometric properties of the set. The following lemma states that if the set consists of N disconnected open sets, determining its membership requires at least $\log_2 N$ complexity.

Lemma 2: Any linear search tree that solves the membership problem for a disjoint union of a family $\{A_i\}_{i \in I}$ of open subsets of R^n requires at least $\log_2 |I|$ queries in the worst case [18].

Proof: [(adapted from [18])] Consider a decision tree algorithm for deciding membership in a set $L \subseteq R^n$. At any leaf node, the algorithm must answer "yes" or "no" to the questions of whether the inputs $x_1, x_2, \cdots, x_n$ are coordinates of a point in L. Let the set of points "accepted" at leaf p be denoted by T_p (i.e., T_p is the set of points for which all tests in the tree have identical outcomes and lead to leaf node p, for which the algorithm answers "yes"). The leaf nodes of the tree partition R^n into disjoint convex regions because all comparisons are between linear functions of the coordinates of the input point, so in particular each of the accepting sets T_p is convex.

We prove the lemma by contradiction. Suppose that the level of the tree is less than $\log_2 |I|$. Then the number of leaf nodes must be strictly less than I. Now since L consisting of $|I|$ disjoint regions, some accepting node T_p must accept points in two regions due to the pigeon-hole principle, say L_α and L_β. Choose any points $P_1 \in T_p \bigcap L_\alpha$ and $P_2 \in T_p \bigcap L_\beta$. Note that the linear comparisons (viewed as hyperplanes) dissect R^n into convex polytopes. By the convexity of T_p, every point on the line P_1P_2 is in T_p. So for every such point the algorithm answers "yes". However, L_α and L_β are disjoint open sets, so the line P_1P_2 contains points not in L. This contradicts the correctness of the membership algorithm. ■

Now we are ready to introduce the aforementioned stronger lemma, concerning sorting complexity lower bound when allowing linear tests. Let $0 \leq m < M$ be two real numbers. The following Lemma (Lemma 3) essentially states that, when linear tests are allowed, the same sorting complexity lower bound $(n \log_2 n - o(n \log_2 n))$ still holds when these n numbers lie in the following n neighborhoods respectively: $\{(m + \frac{i(M-m)}{n+1} - \delta, m + \frac{i(M-m)}{n+1} + \delta)\}_{1 \leq i \leq n}$, where $0 < \delta < \frac{M-m}{3(n+1)}$ is a small real constant so that these regions are disjoint. To see this, we show that this sorting problem is at

[4] Among a set $T = \{x_1, x_2, \cdots, x_n\}$ of n real numbers, decide whether there exists $S \subseteq T$ such that $\sum_{x \in S} x = 1$.

[5] This, however, does not imply $P = NP$, since a decision tree algorithm can be more powerful than a computer program.

least asymptotically as hard as the membership problem for the following set L: $L = \{(y_1, y_2, ..., y_n) \in R^n$: there exists a permutation π of 1, ..., n such that $m + \frac{i(M-m)}{n+1} - \delta < y_{\pi(i)} < m + \frac{i(M-m)}{n+1} + \delta$, $i = 1, 2, \cdots, n\}$. Sorting is at least asymptotically as hard, since if there is an algorithm for sorting with computational complexity B, then there is a $B + O(n)$ algorithm for the membership problem (just sort the numbers using B time and check using $O(n)$ time if they are in the corresponding neighborhoods).

Lemma 3: Under the decision tree model that allows linear tests, given the inputs $\{x_i\}_{1 \le i \le n}$, determining whether $(x_1, x_2, \cdots, x_n) \in L$ requires at least $n \log_2 n$ - $o(n \log_2 n)$ linear tests.

Note that this result is stronger than Lemma 1 since here the computational model (allowing linear tests) is stronger and there are restrictions on the values that these n numbers can take.

Proof: Let Π be the set of permutations on the set $\{1, 2, \cdots, n\}$. Then by the definition of L, $L = \bigcup_{\pi \in \Pi} L_\pi$. Here $L_\pi = \{(y_1, y_2, ..., y_n) : m + \frac{i(M-m)}{n+1} - \delta < y_{\pi(i)} < m + \frac{i(M-m)}{n+1} + \delta$, $i = 1, 2, \cdots, n\}$. Each L_π is obviously an open set. Also L_{π_1} and L_{π_2} are disjoint if $\pi_1 \neq \pi_2$. To see this, note that if $\pi_1(i) \neq \pi_2(i)$ for some i, then each point in L_{π_1} and each point in L_{π_2} must have a minimum distance of δ between their $i'th$ coordinates.

The number of such regions $\{L_\pi\}_{\pi \in \Pi}$ is $n!$ because $|\Pi| = n!$. So by Lemma 2, the number of comparisons must be at least $\log_2(n!)$, which by Stirling's formula ($n! \sim \sqrt{2\pi n}(\frac{n}{e})^n$), is equal to $n \log_2 n$ - $o(n \log_2 n)$. ∎

Remark: We emphasize that the floor (and equivalently the ceiling) function is not allowed in the decision tree. Otherwise, an $O(n)$ algorithm obviously exists for deciding L-membership based on bucket sorting. Note that the floor function is a not a linear function (piecewise linear instead). The linearity of the test is very important in the proof of Lemma 2 since it relies on the fact that the linear tests dissect the space R^n into *convex* regions (polytopes). These regions are no longer convex when the floor function is allowed. For this reason, the floor function[6] is disallowed in almost all lower bound proofs. Nevertheless, despite the fact that the floor function will "spoil" our lower bound proofs (and many other proofs), no existing scheduling algorithm (certainly allowed to use "floor") is known to have a *worst case* computational complexity of $o(\log n)$ and guarantee O(1) or $O(n^a)$ $(0 < a < 1)$ *worst-case* GPS-relative delay. Studying the computation power of "floor" on this scheduling problem can be a challenging topic for future research.

B. $\Omega(\log n)$ *complexity for* $O(1)$ *delay*

In this section, we prove that $\Omega(\log n)$ complexity is required to guarantee $O(1)$ GPS-relative delay, when only comparisons between inputs (equivalently GPS virtual finish times) are allowed. A naive argument for this would be that it takes $\Omega(\log n)$ per packet to schedule the packets according to the sorted order of their GPS virtual finish times. However, this argument is not

[6]Its computational power is discussed in [21] in detail.

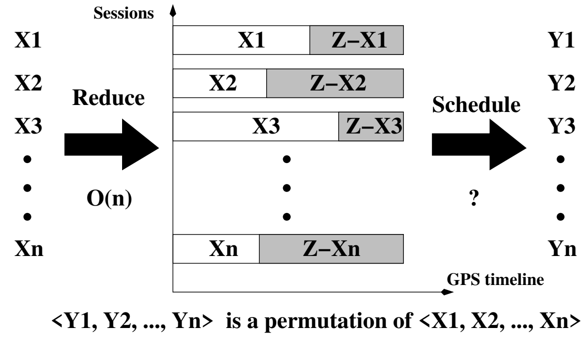

Fig. 4. Converting a sorting instance to a scheduling instance

a proof since it can be shown that to be sorted is not a necessary condition (although sufficient [4]) to assure $O(1)$ GPS-relative delay. For example, if a GPS-relative delay bound of 10 maximum size packets needs to be assured, then given a service schedule sorted according to their GPS virtual finish times, any 9 packets can be relocated (intra-session packet service order should however be preserved) without violating this delay bound.

Before stating the lower bounds and their proofs, we would like to explain the intuition behind them. The core idea is to reduce the problem of scheduling with $O(1)$ delay bound to the problem of sorting, as shown in Fig. 4 (here $Z = L_{max}$). Given any sorting instance, we reduce it to a scheduling instance in $O(n)$ time and "feed" it to a scheduler that guarantees $O(1)$ delay bound. Then we can show that the resulting output can be sorted in $O(n)$ time. Since the sorting complexity is $n \log_2 n - o(n \log_2 n)$, the scheduling complexity has to be at least $n \log_2 n - o(n \log_2 n)$. Otherwise, we have an algorithm that asymptotically beats the sorting complexity lower bound, which is impossible.

The proof is split into two parts. In the first part (Theorem 1), we explain the reduction algorithm and establish the complexity equations. In the second part (Theorem 2), we show that this reduction program is correct in the sense that the resulting program (output of the reduction process) indeed performs sorting correctly. This is proved using standard invariant-based techniques for establishing program correctness, and an assertion that a scheduling program should satisfy (Lemma 4), when comparisons are only allowed between inputs. For reasons explained before, the stronger version of the sorting problem (i.e., Lemma 3) is used for the reduction proof instead of the weaker version (i.e., Lemma 1), although the latter is sufficient for the proof.

In proving the following theorem, we assume that there is an $O(1)$-Delay-Scheduler procedure which guarantees that the GPS-relative delay of any packet will not exceed $K\frac{L_{max}}{r}$ (i.e., $O(1)$). Here $K \ge 1$ is a constant integer independent of the number of sessions n and the total link bandwidth r. We also assume that the CBFS condition is satisfied.

Theorem 1—Complexity: The computational complexity lower bound of the procedure $O(1)$-Delay-Scheduler is $\Omega(\log n)$ per packet.

Proof: To reduce scheduling to L-membership, we construct a procedure for solving L-membership (defined in the

```
1. Procedure L-Membership I
2. input:  $x_1, x_2, \cdots, x_n$
3. output:  ``yes'' if $(x_1, x_2, \cdots, x_n) \in L$ and ``no'' otherwise
4. begin
5.    /* Part I: Create a packet arrival instance and feed it to scheduler */
6.    if $0 < x_i < L_{max}$ for $1 \leq i \leq n$ then proceed
7.    else answer ``no'' endif
8.    for i=1 to n begin
9.       create (first) packet arrival $A_{i,1}$ to session $i$ of length $x_i$ at time 0
10.      create (second) packet arrival $A_{i,2}$ to session $i$ of length $L_{max} - x_i$
          at time 0
11.   end /* for */
12.   call Procedure $O(1)$-Delay-Scheduler with
13.      input:  arrival instance $A = \{A_{i,j}\}_{1 \leq i \leq n, 1 \leq j \leq 2}$
14.      output:  schedule $S = \{S_i\}_{1 \leq i \leq 2n}$ with O(1) delay guarantee
15.   j:=1
16.   for i=1 to 2n begin
17.      if $S_i$ is the first packet of a session then T[j] = $S_i$ endif
18.      j:=j+1
19.      /*T will only have $n$ elements:  first packets of the $n$ sessions*/
20.   end /* for */

21.   /* Part II: ``sort'' the output schedule from the scheduler */
22.   for i:= 2 to $K+2$ begin
23.      perform binary insertion of $T_i$ into the list $T_1, T_2, \cdots, T_{i-1}$ according
         to their lengths
24.      /* sort the first $K+2$ packets using binary insertion according
         to their lengths */
25.   end /* for */
26.   for i:= $K+3$ to $n$
27.      perform binary insertion of $T_i$ into the list $T_{i-K-2}, T_{i-K-1}, \cdots, T_{i-1}$
         according to their lengths
28.      /* binary insertion into a ``window'' of size $K+2$ */
29.   end {for}

30.   /* Part III: check if the ``sorted'' list, viewed a point in $R^n$, is
         in $L$ */
31.   if $\frac{iL_{max}}{n+1} - \delta < length(T_i) < \frac{iL_{max}}{n+1} + \delta$ for $i = 1, 2, \cdots, n$ then answer ``yes''
32.   else answer ``no'' endif
33. end /* procedure */
```

Fig. 5. Algorithm I for L-Membership Test.

previous section) as follows. Recall that $L = \{(y_1, y_2, ..., y_n)$: there exists a permutation π of $\{1, 2, ..., n\}$ such that $m + \frac{i(M-m)}{n+1} - \delta < y_{\pi(i)} < m + \frac{i(M-m)}{n+1} + \delta,\ i = 1, 2, \cdots, n\}$, where $0 < \delta < \frac{M-m}{3(n+1)}$. Here we let $m = 0$ and $M = L_{max}$, where L_{max} is the maximum packet size. We proved in Lemma 3 that the number of linear tests that are needed in determining L-membership is $n \log_2 n - o(n \log_2 n)$. Now, given the inputs $\{x_i\}_{1 \leq i \leq n}$ to the L-membership problem, we convert it to an instance of packet arrivals. We then feed the packet arrival instance to the procedure $O(1)$-Delay-Scheduler. Finally, we process the output from the procedure to solve the L-membership problem. Since the total number of comparisons for solving L-membership is $n \log_2 n - o(n \log_2 n)$ in the worst case, a simple counting argument allows us to show that $O(1)$-Delay-Scheduler must use $\Omega(n \log n)$ comparisons in the worst case. This reduction is illustrated in Fig. 5.

The procedure in Fig. 5 is divided into three parts. In the first part (line 5 through 20), the program first checks if all the inputs are in the legitimate range (0, L_{max}). It then generates two packets for each session i that arrive at time 0. The first and second packets of session i are of length x_i and $L_{max} - x_i$, respectively. Clearly, between time 0 and $\frac{nL_{max}}{r}$, the CBFS condition holds. The arrival instance is fed as input to the procedure $O(1)$-delay-scheduler that guarantees a delay bound of $K\frac{L_{max}}{r}$. The output is the service order of these $2n$ packets determined by the scheduling procedure. Then the second packet of each session is removed from the schedule (line 16 through 20). In the second part (lines 21 through 29), these packets are sorted according to their lengths, if $(x_1, x_2, \cdots, x_n) \in L$ and the procedure $O(1)$-Delay-Scheduler indeed guarantees $O(1)$ GPS-relative delay. In the third part (line 30 through 32), the processed (sorted) sequence is checked to see if it is indeed in L.

Recall that the procedure $O(1)$-Delay-Scheduler is allowed to perform comparisons between its inputs, which are arrival times (0) and lengths of the packets $\{x_i\}_{1 \leq i \leq n}$. In addition, the constant L_{max} is allowed to be compared with any input[7]. Note that this is equivalent to allowing comparisons between

[7] We can artificially create a dummy session which has a packet arrival of length L_{max} at time 0.

GPS virtual finish times of the packets, which are in the form of either $\frac{nx_i}{r}$ (first packet of session i), $i = 1, 2, \cdots, n$, or $\frac{nL_{max}}{r}$ (second packets of all sessions). Both are linear functions of the inputs which can be used in L-membership without compromising its $n \log_2 n - o(n \log_2 n)$ complexity lower bound (by Lemma 3). Now it is straightforward to verify that excluding the procedure $O(1)$-Delay-Scheduler, a total of $O(n)$ linear comparisons/tests are performed throughout the L-membership procedure. They include (a) comparisons in line 17 between the GPS virtual finish time of T_i and $\frac{nL_{max}}{r}$, (b) comparisons between GPS virtual finish times of packets from line 21 through 29, and (c) comparisons in line 31 to check if the (sorted) input is in L. So the number of comparisons used in the procedure $O(1)$-Delay-Scheduler must be $\Omega(n \log n)$. Otherwise, L-membership uses only $o(n \log n)$ comparisons, which contradicts Lemma 3. Therefore, the **amortized complexity per packet** is $\Omega(\log n)$.

We have yet to prove the correctness of the L-membership procedure, i.e., it solves the L-membership correctly for any inputs. This is shown next in Theorem 2. ∎

Theorem 2—Correctness: The procedure in Fig. 5 will return yes if and only if $(x_1, x_2, ..., x_n) \in L$.

Proof: The "only if" part is straightforward since line 30 through 32 (validity check) will definitely answer "no" if $(x_1, x_2, \cdots, x_n) \notin L$. We only need to prove the "if" part.

Note that after the execution of line 20, $\{length(T_i)\}_{1 \le i \le n}$ is a permutation of the inputs $\{x_i\}_{1 \le i \le n}$. Right after the execution of line 25, the lengths of $T_1, T_2, \cdots, T_{K+2}$ are in increasing order. We prove by induction that the lengths of all packets are sorted in increasing order after the execution of the loop from line 26 to 29. We refer to the iterations in the loop as $I_{K+3}, I_{K+4}, ..., I_n$, indexed by the value of i in each iteration. We prove that the first i numbers are sorted after iteration i, $i = K+3, \cdots, n$. This is obviously true for $i = K+3$. Suppose it is true for $i = q \ge K+3$. We prove that it is also true for $i = q+1$.

We claim that, right after the execution of line 20, in the schedule $\{T_i\}_{1 \le i \le n}$, for $K+3 \le i \le n$, there can be no more than $K+2$ elements among $T_1, T_2, \cdots, T_{i-1}$ that are longer than T_i. This is proved below in Lemma 4. Then since the lengths of $T_1, T_2, \cdots, T_q$ are sorted in increasing order after iteration q by the induction hypothesis, we know that $length(T_{q-K-2}) \le length(T_{q+1})$. Otherwise, there are at least $K+3$ packets $(T_{q-K-2}, T_{q-K-1}, \cdots, T_q)$ that are longer than T_{q+1}. So for correct binary insertion, the program only needs to search between the index $q-k-2$ and q, as the program does in line 27. So the lengths of the first $q+1$ packets remain sorted after the insertion: the $i = q+1$ case proved. Finally, note that line 31 correctly checks for L-membership if the numbers $\{length(T_i)\}_{1 \le i \le n}$ are sorted in increasing order. ∎

The following lemma states that no packet will be scheduled behind more than $K+2$ packets that have larger GPS virtual finish times. The intuition behind its proof is the following. Suppose a packet P is scheduled behind $K+2$ packets that have larger timestamps. We convert the current packet arrival instance into another instance in which (a) all timestamps that are no larger than $P's$ (including P itself) are changed to small positive numbers that are close to 0 and all timestamps that are larger are changed to large numbers that are close to L_{max}, and (b) the order of any pair of timestamps remains unchanged. The condition (b) guarantees that the resulting schedule will be the same if only direct comparisons between inputs are allowed. However, P is scheduled behind that $K+2$ packets under the new service schedule, which can be shown to violate the $O(1)$ GPS-relative delay guarantee for P.

Lemma 4: Suppose that $(x_1, x_2, \cdots, x_n) \in L$. Then for any i, $1 \le i \le n$, there can be no more than $K+2$ packets among $T_1, T_2, \cdots, T_{i-1}$ that are longer than T_i, in the scheduler output right after the execution of line 20 in Fig. 5.

Proof: Note that $length(T_k) \ne length(T_l)$ when $k \ne l$, since $(x_1, x_2, \cdots, x_n) \in L$. So there exists a unique permutation π of $\{1, 2, ..., n\}$, such that $length(T_{\pi(1)}) < length(T_{\pi(2)}) < \cdots < length(T_{\pi(n)})$. We prove the lemma by contradiction. For any $i > K+3$, suppose there are more than $K+2$ packets that are scheduled before T_i and are longer than T_i. Suppose $\pi(j) = i$, i.e., T_i is the $j'th$ smallest packet among $\{T_k\}_{1 \le k \le n}$. We argue that $i \le j + K + 2$. In other words, T_i should not be displaced backward by more than $K+2$ positions. To see this, we generate two arbitrary sets of real numbers $\{\alpha_k\}_{1 \le k \le n}$ and $\{\beta_k\}_{1 \le k \le n}$, where $0 < \alpha_1 < \alpha_2 < ... < \alpha_n < \delta$ and $0 < \beta_n < \beta_{n-1} < ... < \beta_1 < \delta$. Here $\delta < \frac{L_{max}}{3(n+1)}$ as before. We consider what happens if we modify the inputs $\{x_k\}_{1 \le k \le n}$ to the L-membership in the following way: x_k is changed to $\alpha_{\pi(k)}$ if $x_k \le x_i$ and is changed to $L_{max} - \beta_{\pi(k)}$ if $x_k > x_i$. It is not hard to verify that the relative order of any two numbers x_l and x_m is the same after the change. Note that the procedure $O(1)$-Delay-Scheduler is only allowed to compare between the inputs, which are $\{x_i\}_{1 \le i \le n}$, 0, and L_{max}. Clearly, with the modified inputs, the decision tree of the procedure $O(1)$-Delay-Scheduler will follow the same path from the root to the leaf as with the original inputs, since all predicates along the path are evaluated to the same values as with the original inputs. Consequently, the output schedule of the packets remains the same with the modified inputs. In the new schedule with the modified inputs, since there are more than $K+2$ packets that are scheduled before T_i and are longer than $L_{max} - \delta$, the actual finish time of T_i is larger than $(K+2)\frac{L_{max}-\delta}{r} > (K+1)\frac{L_{max}}{r}$. However, its GPS virtual finish time is no larger than $\frac{n\delta}{r} < \frac{L_{max}}{r}$. So the GPS-relative delay of the packet T_i must be larger than $(K+1)\frac{L_{max}}{r} - \frac{L_{max}}{r} = K\frac{L_{max}}{r}$. This violates the assumed property of $O(1)$-Delay-Scheduler. ∎

C. $\Omega(\log n)$ *complexity for* $O(n^a)$ *delay*

In this section, we prove that the tradeoff curve is flat as shown in Fig. 2: $\Omega(\log n)$ complexity is required even when $O(n^a)$ delay $(0 < a < 1)$ can be tolerated. Its reduction proof is mostly similar to that of Theorem 1. The main difference is that the constant factor before the asymptotic term $(n \log_2 n)$ becomes critical in this case.

Theorem 3: Suppose we have a procedure $O(n^a)$-Delay-Scheduler that guarantees a GPS-relative delay of no more than $Kn^a\frac{L_{max}}{r}$. Here $K \ge 1$ is an integer constant and $0 < a < 1$ is a real constant. Then the complexity lower bound of $O(n^a)$-Delay-Scheduler is $\Omega(\log n)$ if it is allowed to compare only between any two inputs.

```
1. Procedure L-Membership II
................  same as in Fig.  5 ................
12.   call Procedure O(n^a)-Delay-Scheduler with
................  same as in Fig.  5 ................
21.   /* Part II: ``sort'' the output schedule from the scheduler */
22.   for i:= 2 to Kn^a + 2 begin
23.      perform binary insertion of T_i into the list T_1,T_2,...,T_{i-1}
         according to their lengths
24.      /* sort the first Kn^a + 2 numbers using binary insertion */
25.   end /* for */
26.   for i:= Kn^a + 3 to n begin
27.      perform binary insertion of T_i into the list T_{i-Kn^a-2},...,T_{i-1} according
          to their lengths
28.      /* binary insertion into a ``window'' of size Kn^a + 2 */
29.   end {for}
................  same as in Fig.  5 ................
```

Fig. 6. Algorithm II for L-Membership Test.

Proof: [Sketch] The proof of this theorem is very similar to that of Theorems 1 and 2. We construct a procedure L-membership-II, which makes "oracle calls" to $O(n^a)$-Delay-Scheduler, shown in Fig. 6. Since it is mostly the same as the program shown in Fig. 5, we display only the lines that are different.

Analysis of the complexity is similar to the proof of Theorem 1. The number of comparisons that are used in line 21 through line 29 is no more than $n\log_2(Kn^a+2) = an\log_2 n + o(n\log_2 n)$. Note that the number of *operations* performed from line 26 through 29 is actually n^{2a} if the data movements are also counted. However, as we have explained earlier in Section II.B, we "charge" only for the comparisons. So the number of comparisons used in $O(n^a)$-Delay-Scheduler must be at least $(1-a)n\log_2 n - o(n\log_2 n)$ since otherwise L-membership II uses less than $n\log_2 n - o(n\log_2 n)$ comparisons in the worst case. This would contradict Lemma 3.

Proof of correctness for the procedure L-membership II is similar to that of Theorem 2. We only need to show the following lemma, the proof of which is omitted since it is similar to that of Lemma 4. ■

Lemma 5: Suppose that $(x_1, x_2, \cdots, x_n) \in L$. Then for any i, $1 \le i \le n$, there can be no more than Kn^a+2 packets among $T_1, T_2, \cdots, T_{i-1}$ that are longer than T_i, in the scheduler output (right after line 20) in Fig. 6.

D. The delay-complexity tradeoff when allowing "amortization"

A careful reader might ask the following interesting question after reading our proofs in the previous two sections: can the complexity of $\Omega(n\log n)$ for scheduling the first $2n$ packets be "amortized" over the long run. In other words, is it possible that such a high cost only needs to be paid at the very beginning and it can be amortized to $o(\log n)$ or even $O(1)$ per packet when more and more packets are processed. The following theorem shows that the answer to this question is unfortunately negative.

Theorem 4: For any integer $N > 0$, there exists a packet arrival instance of size $N' \ge N$ such that, the minimum complexity to schedule these N' packets to achieve $O(1)$ or $O(n^a)$ $(0 < a < 1)$ delay bound is $\Omega(\log n)$ per packet.

Proof: [Sketch] The idea of the proof is illustrated in Fig. 7 (here again $Z = L_{max}$). Given any integer $N > 0$, we choose another integer $N' > 0$ such that $N' \ge N$ and N' is a multiple of n. Then given N'/n independent L-membership instances of size n each, we convert, in the same way as in Fig. 4, the first, second, ..., $(N'/n)_{th}$ L-membership instances into scheduling instances for the first, second, ..., and $(N'/n)_{th}$ busy periods (defined below) of the scheduler, respectively. These N'/n busy periods are separated by a small constant time t (real time). Since these L-membership instances are independent, the complexity of deciding each instance is $n\log_2 n - o(n\log_2 n)$ by Lemma 3. Then, using the results of Theorems 1 and 3 (for the cases of $O(1)$ and $O(n^a)$ delay) above, we can show that during each busy period, the scheduling cost must be $\Omega(n\log n)$. This is exactly $\Omega(\log n)$ per packet. ■

Definition 4: We denote as $b(t)$ be the amount of traffic in the service queue at time t. We say that a busy period starts at τ when $b(\tau) > 0$ and $b(\tau^-) = 0$. We say that a busy period ends at s when $B(s^-) > 0$ and $B(s) = 0$.

IV. Complexity–Delay Tradeoffs when Allowing Linear Tests

In the previous section, we have established the lower bound of $\Omega(\log n)$ for guaranteeing $O(n^a)$ GPS-relative delay for $0 \le a < 1$. However, the computational model is slightly restrictive: we only allow the comparisons between the inputs (equivalently the GPS virtual finish times). In this section, we extend the complexity lower bounds to a much stronger computational model, namely, the decision tree that allows comparisons between linear combinations of the inputs. However, to be able to prove the same complexity bounds in the new model, we require that the same ($O(n^a)$ for $0 \le a < 1$) delay bounds are achieved for a different and stronger type of delay called *disadvantage delay*. Despite this restriction, the overall result is provably stronger (by Theorem 6) than results (Theorems 1 and 3) in the last section. Whether the same complexity lower bound holds when linear comparisons are allowed and $O(1)$ or $O(n^a)$ GPS-relative delay bound needs to be guaranteed is left as an open problem for further exploration.

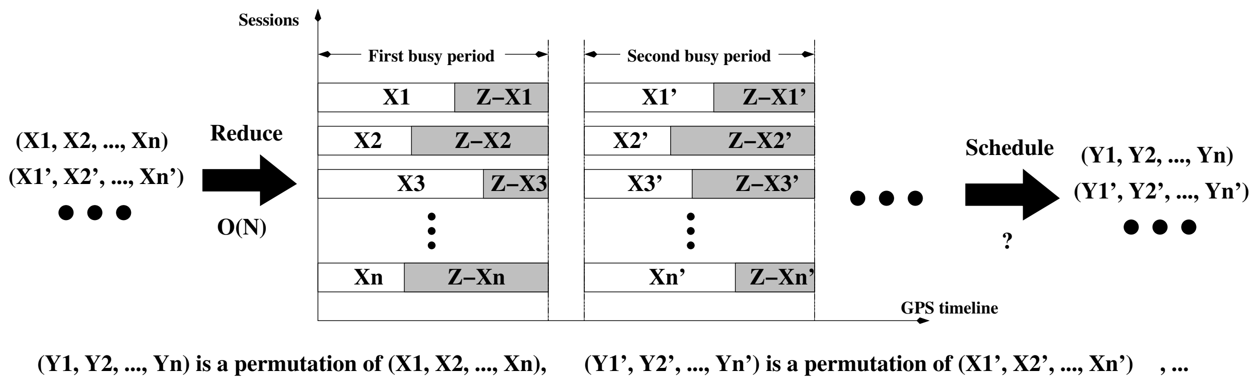

Fig. 7. Why "amortized" scheduling complexity remains $\Omega(\log n)$ per packet

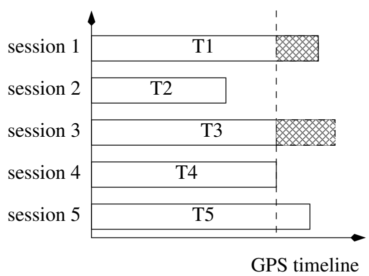

Fig. 8. Disadvantage of packet T_4 (the shaded area)

With respect to a service schedule of packets $T = T_1, T_2, \cdots, T_n$, we define *disadvantage* of a packet T_i (denoted as $disadv(T_i)$) as the amount of traffic that has actually been served in the schedule T, which should have been served after the virtual finish time of T_i in GPS. The *disadvantage delay* is defined as *disadvantage* divided by the link rate r.

In Fig. 8, the shaded area adds up to the disadvantage of the packet T_4 when the service schedule is in the order $T_1, T_2, \cdots, T_5$. Recall that F_p^{GPS} denotes the virtual finish time of the packet p served by GPS scheduler. Formally, the disadvantage of the packet T_i $(i = 1, 2, \ldots, n)$ is

$$disadv(T_i) = \sum_{j=1}^{i-1} max(0, F_{T_j}^{GPS} - F_{T_i}^{GPS}) \qquad (6)$$

So $disadv(T_i)$ can be viewed as the total amount of undue advantage in terms of service other packets have gained over the packet T_i. The following lemma states that the *disadvantage delay* of a packet is always no smaller than its GPS-relative delay.

Lemma 6: Under the CBFS condition, for any packet service schedule $T = T_1, T_2, \cdots, T_n$, the *disadvantage delay* of T_i $(1 \leq i \leq n)$ is always no smaller than its GPS-relative delay.

Proof: Let B and A be the set of bits that should have been served before and after $F_{T_i}^{GPS}$ in a GPS scheduler, respectively. Then, under the CBFS condition, the GPS-relative delay of T_i can be written as $max(0, \frac{1}{r}\sum_{j=1}^{i-1} length(A \bigcap T_j)$ $-\frac{1}{r}\sum_{j=i+1}^{n} length(B \bigcap T_j))$. The disadvantage delay of a packet, on the other hand, is $\frac{1}{r}\sum_{j=1}^{i-1} length(A \bigcap T_j)$. So the disadvantage delay is no smaller than the GPS-relative delay. ■

Remark: The above lemma implies that, for the same amount, guaranteeing disadvantage delay bound is harder than guaranteeing GPS-relative delay bound. In other words, the complexity lower bound result for the former is weaker than the result for the latter. However, guaranteeing disadvantage delay bound is only slightly harder: the disadvantage delay bound of WFQ is zero and, in the scenarios used for proving our lower bounds, that of WF^2Q is also zero.

Now we are ready to state and prove the main theorem of this section: $\Omega(\log n)$ per packet is needed to guarantee a disadvantage delay bound of no more than $O(n^a)$ $(0 \leq a < 1)$. In proving the following theorem, we assume that there is a $O(n^a)$-Disadvantage-Scheduler $(0 \leq a < 1)$ that guarantees a disadvantage delay bound of $Kn^a\frac{L_{max}}{r}$ (i.e., $O(n^a)$), where $K \geq 1$ is an integer constant.

Theorem 5: The number of linear tests used in the procedure $O(n^a)$-Disadvantage-Scheduler $(0 \leq a < 1)$ have a lower bound of $\Omega(n \log n)$ in the worst case.

Proof: The framework of the proof is the same as those of Theorems 1, 2, and 3. A procedure for L-membership test is shown in Fig. 9. Since it is very similar to the program shown in Fig. 5, we show only the lines that are different. The comparisons used in the procedure include (a) comparisons used in $O(n^a)$-Disadvantage-Scheduler, (b) no more than $n\log_2(\lceil\sqrt{2(n+1)(Kn^a+1)}\rceil) = \frac{a+1}{2}n\log_2 n + o(n\log_2 n)$ comparisons used in line 21 through 29, and (c) $O(n)$ comparisons used line 16 through 20. Since $(a) + (b) + (c) = n\log_2 n - o(n\log_2 n)$, we know that the (a) part must be at least $(1 - \frac{a+1}{2})n\log_2 n = \Omega(n \log n)$.

It remains to prove the correctness of L-membership III. Again its proof is quite similar to that of Theorem 2. We claim that the $\{T_i\}_{1\leq i\leq n}$ are sorted after the execution of line 29. Similar to the proof of theorem 2, it suffices to show that the following lemma holds. ■

Lemma 7: Suppose $(x_1, x_2, \cdots, x_n) \in L$. Then right before the execution of line 22 in Fig. 9, for any packet T_i, there can be no more than $\lceil\sqrt{2(n+1)(Kn^a+1)}\rceil$ packets, among $T_1, T_2, \cdots, T_{i-1}$, that are longer than T_i.

Proof: We prove by contradiction. Let $\Gamma = \{T_j : 1 \leq j \leq i-1, length(T_j) > length(T_i)\}$ and let $N = |\Gamma|$. Since $(x_1, x_2, \cdots, x_n) \in L$, we know that each interval $(\frac{jL_{max}}{n+1} - \delta, \frac{jL_{max}}{n+1} + \delta)$, $1 \leq j \leq n$, must contain the length of one and

```
Procedure L-Membership III
................ same as in Fig. 5 ................
  call Procedure $O(n^a)$-Disadvantage-Scheduler with
................ same as in Fig. 5 ................
  /* Part II: ``sort'' the output schedule from the scheduler */
  for i:= 2 to $\lceil\sqrt{2(n+1)(Kn^a+1)}\rceil$ begin
     perform binary insertion of $T_i$ into the list $T_1, T_2, \cdots, T_{i-1}$ according to
      their lengths
     /* sort the first $\lceil\sqrt{2(n+1)(Kn^a+1)}\rceil$ numbers using binary insertion */
  end /* for */
  for i:= $\lceil\sqrt{2(n+1)(Kn^a+1)}\rceil$+1 to $n$ begin
     binary insertion of $T_i$ into the list $T_{i-\lceil\sqrt{2(n+1)(Kn^a+1)}\rceil}, \cdots, T_{i-1}$ according
      to their lengths
     /* binary insertion into a ``window'' of size $\lceil\sqrt{2(n+1)(Kn^a+1)}\rceil$ */
  end {for}
................ same as in Fig. 5 ................
```

Fig. 9. Algorithm III for L-Membership Test.

exactly one packet among $\{T_j\}_{1\leq j\leq n}$. So in the sorted order of their lengths, packets in Γ must be longer than T_i for at least $\frac{L_{max}}{n+1} - 2\delta$, $\frac{2L_{max}}{n+1} - 2\delta$, $\cdots$, and $\frac{NL_{max}}{n+1} - 2\delta$, respectively. Suppose $N > \lceil\sqrt{2(n+1)(Kn^a+1)}\rceil$. Then

$$
\begin{aligned}
disadv(T_i) &> \frac{1}{r}\sum_{j=1}^{N}(\frac{jL_{max}}{n+1} - 2\delta) \\
&= \frac{1}{r}(\frac{\frac{1}{2}N(N+1)L_{max}}{n+1} - 2N\delta) \\
&> Kn^a\frac{L_{max}}{r} \qquad (7)
\end{aligned}
$$

This contradicts the guarantee provided by the procedure $O(n^a)$-Disadvantage-Scheduler. Therefore, N must be no more than $\lceil\sqrt{2(n+1)(Kn^a+1)}\rceil$. ■

Compared to Theorems 1 and 3, Theorem 5 allows for a much stronger computational model. However, it has to enforce a slightly stronger type of delay (disadvantage delay) than GPS-relative delay to maintain the same lower bounds. Nevertheless, the overall result of Theorem 5 is provably stronger than that of Theorems 1 and 3, shown next.

Theorem 6: If a scheduler assures $O(n^a)$ GPS-relative delay bound using only comparisons between inputs (equivalently GPS virtual finish times), it also necessarily assures $O(n^a)$ disadvantage delay bound.

Proof: The proof of Lemma 4 can be adapted to show that among $\{T_j\}_{1\leq j\leq i-1}$ there can be no more than $Kn^a + 2$ packets that are longer than T_i. So the disadvantage delay of T_i is no more than $(Kn^a+2)\frac{L_{max}}{r}$, which is $O(n^a)$. ■

We conclude this section by the following conjecture that was the initial goal of this work.

Conjecture 1: The complexity lower bound for a scheduling algorithm to achieve a delay bound of $O(1)$, under the decision tree model that allows *linear tests*, is $\Omega(\log n)$ per packet. A stronger result would be to generalize it further to the case of $O(n^a)$ $(0 < a < 1)$ delay bound.

V. On The Computational Complexity of Guaranteeing End-to-End Delay

In the previous two sections, we obtain complexity lower bounds for achieving $O(n^a)$ $(0 \leq a < 1)$ GPS-relative or disadvantage delay bounds. However, it is more interesting to derive complexity lower bounds for scheduling algorithms that provide tight end-to-end delay. In this section, we study the computational complexity of providing end-to-end delay bounds, under the Latency Rate framework [11].

A. Background on Latency Rate framework

In [11], Stiliadis and Varma defined a general class of latency rate (LR) schedulers capable of describing the worst-case behavior of numerous scheduling algorithms. From the viewpoint of a session i, any LR scheduler is characterized by two parameters: *latency bound* Θ_i and minimum guaranteed rate r_i. We further assume that the $j'th$ busy period of session i starts at time τ. Let $W_{i,j}(\tau, t)$ denote the total service provided to packets in session i that arrive after time τ and until time t by the scheduler. A scheduler S belongs to the class LR if, for all times t after time τ and until the packets that arrived during this period are serviced,

$$W_{i,j}(\tau, t) \geq max(0, r_i(t - \tau - \Theta_i)) \qquad (8)$$

It has been shown that, for a large class of LR schedulers (including WFQ [4], FFQ [14], VC [13], WF^2Q [6], WF^2Q+ [15]), the latency bound of session i, denoted as Θ_i, is

$$\Theta_i = \frac{L_{max}}{r} + \frac{L_{max,i}}{r_i} \qquad (9)$$

Here $L_{max,i}$ is the maximum size of a packet in session i and r_i is the service rate guaranteed to session i. Note in (9) that the first term in RHS is the GPS-relative delay bound in both WFQ and WF^2Q.

One important property of the latency bound Θ_i, shown in [11], is that it can be viewed as the worst-case delay seen by a session i packet arriving into an empty session i queue. It has been shown in [11] that the latency bound is further connected

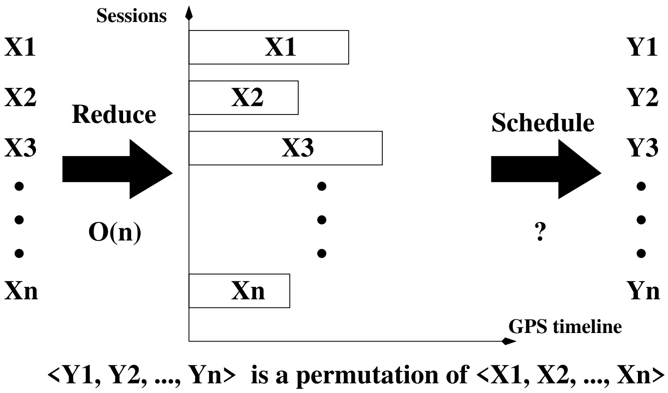

Fig. 10. Converting a sorting instance to a scheduling instance

to the end-to-end delay bound of session i, denoted as D_i^N, by the following inequality:

$$D_i^N \leq \frac{\sigma_i}{r_i} + \sum_{j=1}^{N} \Theta_i^j \tag{10}$$

Here N is the number of nodes (routers) that traffic in session i traverses and Θ_i^j is the latency bound of session i in $j'th$ scheduler. Also, traffic in session i is leaky-bucket constrained and σ_i is the size of the leaky bucket. This result is strong and important since different routers on the path may use different LR schedulers, and (10) still holds in this heterogeneous setting.

B. Our complexity results

We show next, in Theorem 7, that to provide a tight latency bound of $O(n^a)\frac{L_{max}}{r} + \frac{L_{max,i}}{r_i}$ for all sessions i and $0 \leq a < 1$, when only direct comparisons between inputs are allowed, the complexity lower bound is $\Omega(\log n)$. Its implications are profound. Note that for all schedulers on the path to be LR schedulers with tight delay bounds is a sufficient rather than necessary condition for achieving tight overall end-to-end delay bounds. Therefore, Theorem 7 does not establish in full generality the complexity lower bounds for achieving tight end-to-end delay bounds. However, there is substantial evidence [11] that this is a "fairly tight" sufficient condition, since most existing scheduling algorithms that can collectively achieve tight end-to-end delay bounds are LR schedulers. Theorem 7 essentially states that such complexity lower bounds hold if the end-to-end delay bounds are provided through a series of good LR schedulers. One possible way of not using good LR schedulers to establish tight end-to-end delay bounds is the dynamic packet state (DPS) (introduced first in SCORE [22]) approach to convey scheduling state information from one scheduler to another, in the context of DiffServ. However, all current DiffServ algorithms [22], [23] that provide tight delay bounds still require the sorting of timestamps at the core routers. In other words, they are not computationally cheaper than LR based approach, although they are indeed cheaper in terms of packet classification and storage costs at the core routers (therefore suitable for DiffServ).

Theorem 7: When only comparisons between inputs are allowed, the per packet computational complexity for an LR scheduler to guarantee a latency bound of $O(n^a)\frac{L_{max}}{r} + \frac{L_{max,i}}{r_i}$, for $0 \leq a < 1$, is $\Omega(\log n)$.

Proof: [sketch] This proof is similar to that of Theorems 1 and 2. We only prove the case of $a = 0$ (i.e., for latency bound of $O(1)\frac{L_{max}}{r} + \frac{L_{max,i}}{r_i}$), since the proof can be extended to the case $0 < a < 1$ in the same way proof of Theorems 1 and 2 is extended to that of Theorem 3. The idea is again to reduce a sorting instance to a scheduling instance. The scheduling instance used in this proof (shown in Fig. 10), however, is different from the instance used in the proof of Theorems 1 and 2 (shown in Fig. 4). The difference is that this instance only has one packet per session starting at time 0, compared to two packets per session in the previous instance. This difference, however, does not affect our assertion[8] that comparing between the virtual finish time of the packets in such scheduling instances is equivalent to comparing between the inputs, since the virtual finish time of the session i packet is nx_i in both instances. Therefore, comparing between the inputs is still a reasonable computational complexity model under the new instance[9]. It will be clear next why we switch to this new instance.

The rest of the proof again is similar to that of Theorems 1 and 2. A procedure for L-membership test is shown in Fig. 11. Since it is very similar to the program shown in Fig. 5, we show only the lines that are different. The main difference lies in lines 8 – 11, where the program generates a packet arrival instance in which only one (instead of two) packet per session arrives at time 0, as discussed above. Then the program feeds this scheduling instance into the procedure $(O(1)\frac{L_{max}}{r} + \frac{L_{max,i}}{r_i})$-latency-Scheduler, which guarantees that it will not delay a session i packet, which arrives at an empty session i queue, by more than $K\frac{L_{max}}{r} + \frac{L_{max,i}}{r_i}$. Such a constant K exists by the definition of $O(1)$. Then, we show next in Lemma 8 that no packet will be displaced by more than $K + 2$ locations. Therefore, by the same arguments used in the proof of Theorem 1, the computational complexity of the procedure $(O(1)\frac{L_{max}}{r} + \frac{L_{max,i}}{r_i})$-Latency-Scheduler must be $\Omega(n \log n)$, or $\Omega(\log n)$ per packet. ■

Lemma 8: Suppose the scheduler guarantees $K\frac{L_{max}}{r} + \frac{L_{max,i}}{r_i}$ latency bound for all i, and only direct comparisons between the inputs are allowed. Suppose that $(x_1, x_2, \cdots, x_n) \in L$. Then for any i, $1 \leq i \leq n$, there can be no more than $K + 2$ packets among $T_1, T_2, \cdots, T_{i-1}$ that are longer than T_i, in the scheduler output right after the execution of the procedure $(O(1)\frac{L_{max}}{r} + \frac{L_{max,i}}{r_i})$-Latency-Scheduler. (line 14 in Fig. 11).

Proof: This proof is similar to that of Lemma 4. In the following, we refer to the procedure $(O(1)\frac{L_{max}}{r} + \frac{L_{max,i}}{r_i})$-Latency-Scheduler as *the scheduler*. Since $(x_1, x_2, \cdots, x_n) \in L$, there exists a unique permutation π of $\{1, 2, ..., n\}$, such that $length(T_{\pi(1)}) < length(T_{\pi(2)}) < \cdots < length(T_{\pi(n)})$. For any $i > K + 3$, suppose there are more than $K + 2$ packets that are scheduled before T_i and are longer than T_i. Suppose $\pi(j) = i$, i.e., T_i is the $j'th$ smallest packet among $\{T_k\}_{1 \leq k \leq n}$.

[8] Recall that this assertion allows us to avoid discussing the complexity of tracking GPS clock.

[9] We did not use this instance in earlier proofs, since the CBFS condition satisfied by our previous instance has other important implications.

```
 1. Procedure L-Membership IV
 2. input:  x_1, x_2, ..., x_n
 3. output:  ``yes'' if (x_1, x_2, ..., x_n) ∈ L and ``no'' otherwise
 4. begin
 5.    /* Part I: Create a packet arrival instance and feed it to scheduler */
 6.    if 0 < x_i < L_max for 1 ≤ i ≤ n then proceed
 7.    else answer ``no'' endif
 8.    for i=1 to n begin
 9.       create packet arrival A_i to session i of length x_i at time 0
10.    end /* for */
11.    /* an idle line added for the alignment with other figures */
12.    call Procedure (O(1) L_max/r + L_max,i/r_i)-Latency-Scheduler with
13.       input:  arrival instance A = {A_i}_{1≤i≤n}
14.       output:  schedule T = {T_i}_{1≤i≤n} with the latency guarantee

15.    /* Lines 15 to 20 left empty to synchronize with Fig.  5 */

21.    /* Part II: ``sort'' the output schedule from the scheduler */
 ................  same as in Fig.  5 ................
```

Fig. 11. Algorithm IV for L-Membership Test.

We claim that $i \leq j + K + 2$, that is, T_i should not be displaced backward by more than $K + 2$ positions. We prove this claim by contradiction. We generate two arbitrary sets of real numbers $\{\alpha_k\}_{1 \leq k \leq n}$ and $\{\beta_k\}_{1 \leq k \leq n}$, where $0 < \alpha_1 < \alpha_2 < ... < \alpha_n < \delta$ and $0 < \beta_n < \beta_{n-1} < ... < \beta_1 < \delta$. Here δ is set such that $\delta < \frac{L_{max}}{n}$. We consider what happens if we modify the inputs $\{x_k\}_{1 \leq k \leq n}$ to the L-membership in the following way: x_k is changed to $\alpha_{\pi(k)}$ if $x_k \leq x_i$ and is changed to $L_{max} - \beta_{\pi(k)}$ if $x_k > x_i$. It is not hard to verify that the relative order of any two numbers x_l and x_m is the same after the change. Note that the scheduler is only allowed to compare between the inputs, which are $\{x_i\}_{1 \leq i \leq n}$, 0, and L_{max}. One can easily verify that, with the modified inputs, the decision tree of the procedure will follow the same path from the root to the leaf as with the original inputs, since all predicates along the path are evaluated to the same values as with the original inputs. Consequently, the output schedule of the packets remain the same with the modified inputs. In the new schedule with the modified inputs, since there are more than $K + 2$ packets that are scheduled before T_i and are longer than $L_{max} - \delta$, the actual finish time of T_i is larger than $(K + 2)\frac{L_{max} - \delta}{r} = K\frac{L_{max}}{r} + \frac{2L_{max} - (K+2)\delta}{r} \geq K\frac{L_{max}}{r} + \frac{L_{max}}{r} > K\frac{L_{max}}{r} + \frac{L_{max,i}}{r_i}$. The last two inequalities hold since we can assume $L_{max,i} = \delta < \frac{L_{max}}{n}$. This assumption is valid because there is no session i packet in the scheduling instance that is longer than δ. Therefore the latency of the packet T_i is larger than $K\frac{L_{max}}{r} + \frac{L_{max,i}}{r_i}$. This violates the latency rate guarantee of the scheduler. ■

Finally, we identify one open problem that we feel very likely to be solvable and its solution can be a very exciting result, stated as Conjecture 2 below. Note that Conjecture 2 is strictly stronger than Conjecture 1, as it can be shown that the former implies the latter.

Conjecture 2: The complexity lower bound for an LR scheduler (introduced in [11]) to achieve a tight latency bound of $O(1)\frac{L_{max}}{r} + \frac{L_{max,i}}{r_i}$ is $\Omega(\log n)$ per packet, under the decision tree model that allows *linear tests*.

Remark: Note that in this conjecture the CBFS condition is not a part of the assumption, and therefore we cannot avoid the issue of tracking GPS clock anymore. However, interestingly FFQ, VC, and WF^2Q+ all achieve this tight latency bound at the complexity of $O(\log n)$ per packet and without tracking GPS clock perfectly. If this conjecture is true, it implies that these algorithms are asymptotically optimal for achieving tight latency bounds, which is an exciting result. Note that Corollary 1 proves this complexity lower bound under the weaker model that allows only comparisons between inputs.

VI. Related Work and Our Recent Result

Since the first submission of this paper, we are working on a closely related open problem[10] "What is the computational complexity lower bound of tracking the GPS clock?" During this endeavor, we debunked a legendary myth in the QoS community that this lower bound is $O(n)$ per packet, and discovered that there is an $O(\log n)$ algorithm for keeping track of GPS clock, proposed by Greenberg and Madras in [1]. An $O(\log n)$ algorithm that nicely integrates the GPS tracking and timestamp sorting is designed and implemented by Keshav [24]. In our recent work [12], among others, we show that tracking the GPS clock has a lower bound of $\Omega(\log n)$ per packet under the linear decision tree. This lower bound matches the upper bound established in [1] and [24]. However, neither tracking GPS perfectly nor sorting the timestamps are shown to be necessary conditions for achieving GPS-relative delay of $O(1)$ under the linear decision tree. Therefore conjectures 1 and 2 remain open despite this recent progress.

VII. Conclusions

In this work, we clarify, extend and solve an open problem concerning the computational complexity for packet scheduling algorithms to achieve tight delay bounds. To the best of our

[10] Recall that the CBFS condition shields us from having to answer this question in this paper.

knowledge, this is the first major step in establishing the complexity lower bounds for packet scheduling algorithms. Our three major results can be summarized as follows:

1) We prove that $\Omega(\log n)$ is indeed the per packet complexity lower bound to guarantee $O(1)$ GPS-relative delay (excluding the cost of tracking the GPS clock), if a scheduling algorithm is only allowed to compare between inputs (equivalently between GPS virtual finish times) in its decision tree. Moreover, we prove that the complexity lower bound remains the same even if the GPS-relative delay bound is relaxed to $O(n^a)$ for $0 < a < 1$, thus establishing the complete tradeoff curve.
2) We are able to extend our complexity results to a much stronger computational model: a decision tree that allows linear tests. However, this comes at the cost of having to enforce a slightly stronger type of delay (disadvantage delay) in the same asymptotic amount ($O(n^a)$, $0 \leq a < 1$). Nevertheless, we show that the overall results remain stronger.
3) We study the computational complexity of providing end-to-end delay bounds, under the Latency Rate framework [11]. We show that the lower bound complexity of providing a tight *latency bound* of $\frac{n^a L_{max}}{r} + \frac{L_{max,i}}{r_i}$ in a LR scheduler, when only comparisons between inputs are allowed, is $\Omega(\log n)$ per packet.

VIII. Acknowledgments

We thank the editor, Prof. Stavrakakis, for coordinating a careful and expeditious review. We thank Dr. Chuanxiong Guo for helpful discussions with the first author. We thank Prof. George Varghese for encouraging the first author to study this important open question. We thank Dr. Scott Shenker for shepherding the conference version of this paper. We thank Prof. Ellen Zegura, Prof. Yechezkel Zalcstein, Mr. Shashidhar Merugu and anonymous referees for their insightful comments and suggestions that help improve the quality and accessibility of the paper.

References

[1] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," *Internetworking: Research and Experience*, pp. 3–26, 1990, Also in Proceedings of ACM SIGCOMM'89.
[2] A. Greenberg and N. Madras, "How fair is fair queuing?," *Journal of the ACM*, vol. 39, no. 3, pp. 568–598, 1992, Also in Proc. of Performance 1990.
[3] H. Zhang, "Service disciplines for guaranteed performance service in packet switching networks," *Proceedings of the IEEE*, vol. 83, no. 10, Oct. 1995.
[4] A. Parekh and R. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: the single node case," *IEEE/ACM Transaction on Networking*, vol. 1, no. 3, pp. 344–357, June 1993.
[5] J. Turner, "New directions in communications (or which way to the information age?)," *IEEE Communications Magazine*, vol. 24, pp. 8–15, Oct. 1986.
[6] J. Bennett and H. Zhang, "wf^2q: worst-case fair weighted fair queuing," in *IEEE INFOCOM'96*, Mar. 1996.
[7] M. Shreedhar and G. Varghese, "Efficient fair queuing using deficit round robin," in *Proc. of ACM SIGCOMM'95*, Aug. 1995, pp. 231–242.
[8] M. Katevenis, S. Sidiropoulos, and C. Courcoubetis, "Weighed round-robin cell multiplexing in general-purpose ATM switch chip," *IEEE Journal on Selected Areas in Communications*, vol. 9, pp. 1265–1279, Oct. 1991.
[9] C. Guo, "SSR: an O(1) time complexity packet scheduler for flows in multi-service packet networks," in *Proc. of Sigcomm'01*, Sept. 2001.
[10] S. Suri, G. Varghese, and G. Chandranmenon, "Leap Forward Virtual Clock: an O(log(log(n))) fair queuing algorithm," in *IEEE Infocom'97*, Kobe, Japan, Apr. 1997.
[11] D. Stiliadis and A. Varma, "Latency-rate servers: a general model for analysis of traffic scheduling algorithms," in *Proc. of Infocom'96*, Mar. 1996.
[12] Q. Zhao and J. Xu, "On the computational complexity of maintaining gps clock in packet scheduling," in *Proc. of Infocom 2004*, Hong Kong, China, Mar. 2004.
[13] L. Zhang, "Virtualclock: a new traffic control algorithm for packet switching networks," *ACM Transactions on Computer Systems*, vol. 9, pp. 101–124, May 1991.
[14] D. Stiliadis and A. Varma, "Design and analysis of frame-based fair queuing: A new traffic scheduling algorithm for packet switched networks," in *Proc. of ACM Sigmetrics'96*, May 1996, pp. 104–115.
[15] J. Bennett and H. Zhang, "Hierarchical packet fair queuing algorithms," *IEEE/ACM Transactions on Networking*, vol. 5, no. 5, pp. 675–689, 1997.
[16] J. Xu and R. Lipton, "On fundamental tradeoffs between delay bounds and computational complexity in packet scheduling algorithms," in *Proc. of ACM Sigcomm*, Aug. 2002.
[17] A. Aho, J. Hopcroft, and J. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1973.
[18] D. Dobkin and R. Lipton, "A lower bound of $\frac{1}{2}n^2$ on linear search programs for the knapsack problem," *J. Comput. Syst. Sci.*, vol. 16, pp. 413–417, 1978.
[19] F. Heide, "A polynomial linear search algorithm for the n-dimensional knapsack problem," *J. of the ACM*, vol. 31, pp. 668–676, 1984.
[20] M. Fredman and B. Weide, "On the complexity of computing of measure of $\bigcup[a_i, b_i]$," *Communications of the ACM*, vol. 21, no. 7, July 1978.
[21] G. Yuval, "Finding nearest neighbors," *Information Processing Letters*, vol. 5, no. 3, pp. 63–65, Aug. 1976.
[22] I. Stoica and H. Zhang, "Providing guaranteed services without per flow management," in *Proc. of ACM SIGCOMM*, Sept. 1999.
[23] J. Kaur and H. Vin, "Core-stateless guaranteed rate scheduling algorithms," in *Proc. of Infocom 2001*, Anchorage, AK, Apr. 2001.
[24] S. Keshav, "On the efficient implementation of fair queueing," *Internetworking: Research and Experiences*, vol. 2, pp. 157–173, 1991.