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

Jun Xu and Richard Lipton
College of Computing
Georgia Institute of Technology
{jx,rjl}@cc.gatech.edu

March 25, 2002

Abstract: In this work, we clarify, extend and solve an 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 $\Omega(log_2 n)$ (widely believed as a "folklore theorem" but never proved). 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 extend both complexity results (for $O(1)$ or $O(n^a)$ delay) to a much stronger computational model. Finally, we show that the same complexity lower bounds are applicable to guaranteeing tight end-to-end delay bounds. This is done by untangling the subtle relationship between the GPS-relative delay bound and the end-to-end delay bound.

1 Introduction

Packet scheduling is an important mechanism in providing QoS guarantees in data networks [Zha95]. The fairest algorithm for packet scheduling is General Processor Sharing (GPS) [PG93]. 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 can be compared with in terms of end-to-end delay bounds and fair bandwidth allocation.

In a link of rate r served by a GPS scheduler, each session $i, i = 1, 2, \cdots, n$ is assigned a weight value ϕ_i. Each backlogged 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 [PG93], is that it can guarantee tight end-to-end delay bound to traffic that is leaky-bucket [Tur86] 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 the GPS scheduler, respectively. It has been shown in [PG93, BZ96] that WFQ and WF^2Q 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 and r is the total link bandwidth. 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 achieves 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-session packets, the one with 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 $O(log_2 n)$. In other words the computational cost to "pay" for the $\frac{L_{max}}{r}$ GPS-relative delay bound in both WFQ and WF^2Q is $O(log_2 n)^1$.

[1] Here the cost of the GPS clock tracking ((a) part) is not

On the other hand, round-robin algorithms such as DRR (Deficit Round Robin) [SV95] and WRR (Weighted Round Robin) [KSC91] 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 have the 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. So the GPS-relative delay of the short packet can be calculated as $\frac{(n-1)(M-m)}{r}$, which is $O(n)$.

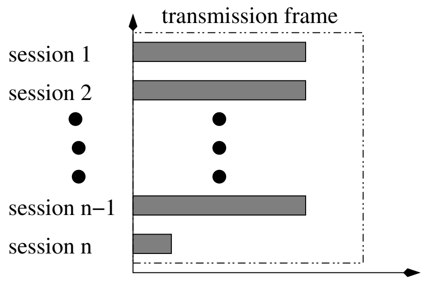

Figure 1: How round robin algorithms incur $O(n)$ GPS-relative delay

We have just shown that algorithms with $O(log_2 n)$ 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 $O(n)$. An open problem proposed in Sigcomm'01 by Guo (author of [Guo01]) is whether this represents indeed the fundamental tradeoff between computational complexity of the scheduling algorithm and the GPS-relative delay bound they can achieve. More specifically, Guo asks whether $\Omega(log_2 n)$ is the asymptotic complexity lower bound for any scheduling algorithm to guarantee $O(1)$ GPS-relative delay bound. Our work clarifies and extends this question, and answers it in a comprehensive way.

The first major result of this paper is to show that $\Omega(log_2 n)$ is indeed the complexity lower bound to guarantee $O(1)$ GPS-relative delay[2], excluding the

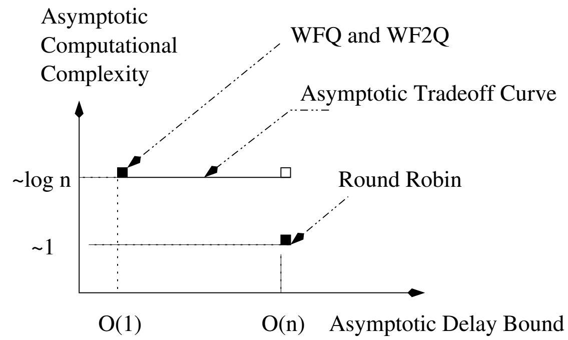

Figure 2: The asymptotic tradeoff curve between delay bound and computational complexity

cost of tracking GPS time. This bound is established under the decision tree computation model that allows direct comparisons between its inputs (in our context between GPS virtual 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 [PG93, BZ96]. This result granted for the moment, we now have two points on the complexity-delay tradeoff curve, as shown in Fig. 2. One is $O(n)$ delay at the complexity of $\Omega(1)$ and the other is the $O(1)$ delay at the complexity of $\Omega(log_2 n)$. One interesting question to ask is how do other parts of the "tradeoff curve" look like. 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 $\Omega(log_2 n)$, say $\Omega(\sqrt{log_2 n})$? The result we discovered and proved 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 $\Omega(log_2 n)$ 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

included.

[2]Leap Forward Virtual Clock ($LFVC$) scheduler has a low implementation complexity of $O(log(logn))$ 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.

GPS-relative delay. Nevertheless, the second result is provably stronger than our first major result (for both $O(1)$ and $O(n^a)$ cases).

Our third and final result is to show that the same complexity lower bounds are to certain extent applicable to guaranteeing tight end-to-end delay bounds. This is done by understanding the relationship between the GPS-relative delay bound and the end-to-end delay bound. In particular we show that, providing tight GPS-relative delay bound of $\frac{L_{max}}{r}$ in Latency Rate (LR) schedulers (introduced in [SV96b]) is conditionally equivalent to providing the tight *latency bound* (also introduced in [SV96b]) of $\frac{L_{max}}{r} + \frac{L_{max,i}}{r_i}$. Here $L_{max,i}$ is the maximum size of a packet in session i and r_i is the guaranteed rate of session i.

Though it is widely believed as a "folklore theorem" that scheduling algorithms which can provide tight end-to-end delay bounds require $\Omega(log_2 n)$ 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 $\Omega(log_2 n)$ 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 2, we introduce the computational models and assumptions we will use in proving our results. The aforementioned three major results are established in Section 3, 4, and 5 respectively. Section 6 concludes the paper.

2 Assumptions and Computational Models

In general, complexity lower bounds on 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 2.1, we describe a network load and resource allocation condition called CBFS (continuously backlogged fair sharing) under which all later lower bounds will be derived. In Section 2.2, we introduce two computational models that will be used in Section 3 and 4, respectively. This section concludes with a discussion on the difficulty of obtaining lower bound proofs in general.

2.1 CBFS condition

All our 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. To see this, note that for a traffic arrival instance that conforms to the CBFS condition, a GPS scheduler will serve each session continuously at the equal rate $\frac{r}{n}$.

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 bound 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 since it is equal to the amount of traffic that has arrived no later than p (including p itself) in the same session, divided by the per session rate $\frac{r}{n}$. So CBFS condition allows us to "naturally exclude" the cost of tracking GPS tracking time.
- Second, we will show that under CBFS condition, many existing scheduling algorithms such as Virtual Clock (VC) [Zha91], Frame-based Fair Queuing (FFQ) [SV96a] and WF^2Q+ [BZ97] are equivalent to either WFQ or WF^2Q. 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 grounds 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}. This is, in general, not true for packet networks. However, we can show 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 in the interest of space.

2.2 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 $\{x_i\}_{1\leq i\leq n}$. 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. In general, when 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. When program control reaches a leaf node the predicate there is evaluated and its result is considered as the result 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 $(1 \leq i \leq 6)$ is a predicate of the inputs.

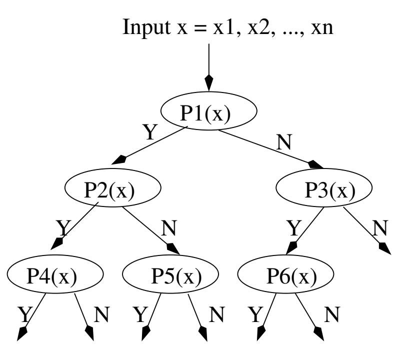

Figure 3: Decision tree computational model

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 computation powers. **The first computational model** we consider is the decision tree that allows *linear tests* [DL78]. In this model, each predicate in the decision tree is in the form of "$h(x_1, x_2, \cdots, x_n) \geq 0$?", where h is a linear function (defined below) of the inputs $\{x_i\}_{1\leq i\leq n}$.

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

This model will be used in our proofs in Section 4. In the context of this work, the inputs will be the lengths and the arrival times of the packets. Note that the linear comparison model is quite generous: functions like h in the above definition may take up to $O(n)$ steps to compute, but the model charge only "1" for it. However, one may still argue that linear test can be restrictive for packet scheduling since it does not offer an efficient way to calculate GPS virtual finish times. Note that GPS virtual finish time is in general not a linear function (actually piece-wise linear) of the inputs. Fortunately, as we have explained in Section 2.1, under the CBFS condition, GPS virtual finish time of any packet is indeed a linear function of these inputs! So under the CBFS condition, this model is not restrictive.

The second computational model we introduce is the decision tree that allows comparisons only between its inputs. This model is weaker than the previous one. It has been used in proving the $\Omega(nlog_2 n)$ lower bound for comparison-based sorting algorithms [AU73]. We will show that under the CBFS condition, this is equivalent to allowing comparisons between GPS virtual finish times of the packets in our reduction proof. As we have explained in Section 1, this model is not too restrictive either.

2.3 Hardness of obtaining lower bounds

In general proving lower bounds on computational problems is a very difficult problem. Computational complexity theory today cannot even rule out the possibility that integer factoring, for example, can be done in linear time (recall RSA relies on integer factoring being intractable for large numbers). The reason for this is that algorithms can be very "clever": they do not have to solve the problem in the "obvious" way. Since, by definition, a lower bound states that no algorithm can solve the problem in a certain time *no matter how it attacks the problem*, proving lower bounds remains very difficult, if not impossible.

Complexity theory, therefore, often attempts to prove lower bounds in restricted models. Obviously, if the restrictions are too strong, then we can prove lower bounds, but they are not very important. The reason is that it may be likely that an algorithm can "break the rules" and solve the problem faster than the lower bound allows. On the other hand, if the restrictions are natural, then the bounds, if provable, are very interesting. They at least show that any algorithm that beats the lower bound has to operate in some "un-natural" manner. Since no such algorithm is often likely to exist, now the lower bounds give us much greater confidence that the problem's complexity has been determined. In any event, such strong lower bounds can help algorithm designers: at least they will know what lines of attack that cannot work.

Our results are based on the decision tree model. This model is very well studied in computational complexity. It has three interesting properties. First, because it is highly geometric, it allows powerful methods of geometry to be used that yield very strong lower bounds. Second, the model is quite "natural". Often the best known algorithms for problems are essentially decision tree computations. For example, the best known algorithms for sorting, many geometric problems, and even NP-complete problems such as Knapsack, are easily modeled as decision tree computations. Finally, the decision tree model is in a sense too "generous". Thus, it is known, for example, that the Knapsack problem, has an $O(n^5 log^2 n)$ algorithm in the decision tree model (allowing linear comparisons) [Hei84]. So the lower bound derived under the decision tree model can be smaller than the lower bound achievable by a computer program! (Note, this is the case since the decision cost is only the *depth* of the tree, not the *size* of the tree.) In summary, we believe that using the decision tree model is a reasonable choice for the type of scheduling problems that we consider here in this paper.

3 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 $\Omega(log_2 n)$. In Section 3.1, we introduce two general lemmas used in later proofs. Section 3.2 and 3.3 proves the $\Omega(log_2 n)$ complexity lower bounds for the case of $O(1)$ and $O(n^a)$ respectively.

3.1 Preliminaries

A *set membership problem* is to determine whether the inputs $\{x_i\}_{1 \le i \le n}$, viewed as a point $(x_1, x_2, \cdots, x_n)$ in the Euclidean space R^n, belongs to a set $L \subseteq R^n$. In complexity theory, proof of the lower bound for solving a set membership problem typically reasons about the geometric properties of the set. In the following, we state a general lemma concerning complexity of set membership problems under the the decision tree model that allows linear tests. This lemma, due to Dobkin and Lipton [DL78], has been used extensively in lower bound proofs (e.g., [FW78]). Since this is a well-known result, its proof is moved to the Appendix.

Lemma 1 *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 [DL78].*

Next we prove another lemma concerning the computational complexity of solving membership problem for a specific set L. Let $0 \le m < M$ be two real numbers. Let $L \subseteq R^n$ and $L = \{(y_1, y_2, ..., y_n) :$ there exists a permutation π of 1, ..., n such that $m + \frac{i(M-m)}{n+1} - \delta < y_{\pi(i)} < m + \frac{i(M-m)}{n+1} + \delta,\ i = 1, 2, \cdots, n\}$. Here $0 < \delta < \frac{M-m}{3(n+1)}$ is a "small" real constant.

Lemma 2 *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* $nlog_2 n$ *-* $o(nlog_2 n)$ *linear tests*[3].

Proof: Let Π be the set of permutations on the numbers $1, 2, \cdots, n$. Then by the definition of L, $L = \bigcup_{\pi \in \Pi} L_\pi$. Here $L_\pi = \{(y_1, y_2, ..., y_n) : m + \frac{i(M-m)}{n+1} - \delta < y_{\pi(i)} < m + \frac{i(M-m)}{n+1} + \delta\}$. Each L_π is obviously an open set. Also L_{π_1} and L_{π_2} are disjoint if $\pi_1 \ne \pi_2$. To see this, note that if $\pi_1(i) \ne \pi_2(i)$ 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 1, 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 $nlog_2 n + o(nlog_2 n)$. □

Remark: We emphasize that the floor (and equivalently the ceiling) function is disallowed in the decision tree. Otherwise, an $O(n)$ algorithm obviously

[3] Here we use this representation instead of $\Omega(nlog_2 n)$ in order to emphasize that the constant factor is precisely 1.

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 1 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[4] 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 $o(log_2n)$ 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 topic for future research.

3.2 $\Omega(log_2n)$ complexity for guaranteeing $O(1)$ delay

In this section, we prove that $\Omega(log_2n)$ 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 $\Omega(log_2n)$ per packet to schedule the packets according to the sorted order of their GPS virtual finish times. However, this argument is not a proof since it can be shown that to be sorted is not a necessary condition (although sufficient [PG93]) to assure $O(1)$ GPS-relative delay.

In the following theorem, we assume that there is a $O(1)$-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 $K \geq 1$ 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 complexity lower bound of the procedure $O(1)$-Delay-Scheduler is $\Omega(log_2n)$ per packet.*

Proof: Our proof uses the reduction method in computational complexity, similar to those used in NP-completeness proofs. We construct a procedure for solving L-membership (defined in the previous section) as follows. Recall that $L = \{(y_1, y_2, ..., y_n)$: there exists a permutation π of 1, ..., n such that $m + \frac{i(M-m)}{n+1} - \delta < y_{\pi(i)} < m + \frac{i(M-m)}{n+1} + \delta$, $i = 1, 2, \cdots, n\}$, 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 2 that the number of linear tests that are needed in determining L-membership is $nlog_2n + o(nlog_2n)$. 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 are $nlog_2n + o(nlog_2n)$ in the worst case, a simple counting argument allows us to show that $O(1)$-Delay-Scheduler must use $\Omega(nlog_2n)$ comparisons in the worst case. This reduction is illustrated in Fig. 4.

The procedure in Fig. 4 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 schedule of these $2n$ packet 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. In addition, the constant L_{max} is allowed to be compared with any input. Note that this is equivalent to allowing comparisons between 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 $nlog_2n + o(nlog_2n)$ complexity lower bound (by Lemma 2). 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 34 to check if the (sorted) input is in L. So the number of comparisons used in the

[4] Its computational power is discussed in [Yur76] in detail.

```
1.  Procedure L-Membership I
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 (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≤i≤n,1≤j≤2}
14.       output:  sorted schedule S = {S_i}_{1≤i≤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, ..., 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}, ..., T_{i-1} according to 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 iL_max/(n+1) − δ < length(T_i) < iL_max/(n+1) + δ for i = 1, 2, ..., n then answer ``yes''
32.    else answer ``no'' endif
33. end /* procedure */
```

Figure 4: Algorithm I for L-Membership Test.

procedure $O(1)$-Delay-Scheduler must be $\Omega(nlog_2n)$. Otherwise, L-membership uses only $o(nlog_2n)$ comparisons, which contradicts Lemma 2. Therefore, the **amortized complexity per packet** is $\Omega(log_2n)$.

We have yet to prove the correctness of the L-membership procedure, shown next. □

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

Proof: The "only if" part is straightforward since line 31 through 34 (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, $\{length(T_i)\}_{1\leq i\leq n}$ is a permutation of the inputs $\{x_i\}_{1\leq i\leq n}$. Right after the execution of line 25, the lengths of $T_1, T_2, \cdots, T_{K+2}$ are in the increasing order. We prove by induction that the lengths of all packets are sorted in the 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 $i = q \geq K+3$. We prove that it is also true for $i = q+1$.

We claim that, right after the execution of line 20, in the schedule $\{T_i\}_{1\leq i\leq n}$, for $K+3 \leq i \leq n$, 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 3. Then since the lengths of $T_1, T_2, \cdots, T_q$ are sorted in the increasing order after iteration q by the induction hypothesis, we know that $length(T_{q-K-2}) \leq length(T_{q+1})$. 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 length 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 $\{length(T_i)\}_{1 \leq i \leq n}$ are sorted in the increasing order. □

Lemma 3 *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 line 20.*

Proof: Note that $length(T_k) \neq length(T_l)$ when $k \neq l$, since $(x_1, x_2, \cdots, x_n) \in L$. So there exists a unique permutation π of 1,2,...,n, such that $length(T_{\pi(1)}) < length(T_{\pi(2)}) < ... < length(T_{\pi(n)})$. 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 $\{T_k\}_{1 \leq k \leq n}$. We argue that $i \leq j + K + 2$. 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 $\{\alpha_k\}_{1 \leq k \leq n}$ 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 $\delta < \frac{L_{max}}{3(n+1)}$ as before. 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. Now since the procedure $O(1)$-Delay-Scheduler is only allowed to compare between GPS virtual finish times of the packets. In the context of the program, these comparisons are between $\frac{nx_l}{r}$ and $\frac{nx_m}{r}$ (equivalently between x_l and x_m), and between $\frac{nx_l}{r}$ and $\frac{nL_{max}}{r}$ (equivalently between x_l 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 value! 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 $K+2$ 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. □

Remark: The ideas contained in the proof bear some similarity to that of Knuth's 0-1 law [Knu98], which states the following. If a sorting network can correctly sort inputs consisting of any arbitrary combinations of 0's and 1's, it must be able to correctly sort all inputs. In our proof, $\{\alpha_i\}_{1 \leq i \leq n}$ and $\{L_{max} - \beta_i\}_{1 \leq i \leq n}$, to certain extent, can be viewed as such 0's and 1's.

3.3 $\Omega(log_2 n)$ complexity for guaranteeing $O(n^a)$ delay

In this section, we prove that the tradeoff curve is flat as shown in Fig. 2: $\Omega(log_2 n)$ complexity is required even when $O(n^a)$ delay $(0 < a < 1)$ can be tolerated.

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 $K \geq 1$ is an integer constant and $0 < a < 1$ is a real constant. Then the complexity lower bound of $O(n^a)$-Delay-Scheduler is $\Omega(log_2 n)$ if it is allowed to compare only between any two inputs.*

Proof (Sketch): The proof of this theorem is very similar to that of Theorem 1 and 2. We construct a procedure L-membership-II, which makes "oracle calls" to $O(n^a)$-Delay-Scheduler, shown in Fig. 5. Since it is mostly the same as the program shown in Fig. 4, 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 $nlog_2(Kn^a + 2) = anlog_2 n + o(nlog_2 n)$. 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 2.2, we "charge" only for the comparisons. So the number of comparisons used in $O(n^a)$-Delay-Scheduler must be at least $(1-a)nlog_2 n + o(nlog_2 n)$ since otherwise L-membership II uses less than $nlog_2 n + o(nlog_2 n)$ comparisons in the worst case. This would contradict Lemma 2.

Proof of correctness for the procedure L-membership II is also similar to that of Theorem 2. We only need to show the following lemma. We omit its proof here since it is similar to that of lemma 3. □

```
 1. Procedure L-Membership II
 ................  same as in Fig.  4 ................
12.    call Procedure O(n^a)-Delay-Scheduler with
 ................  same as in Fig.  4 ................
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.  4 ................
```

Figure 5: Algorithm II for L-Membership Test.

Lemma 4 *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 $Kn^a + 2$ packets among $T_1, T_2, \cdots, T_{i-1}$ that are longer than T_i, in the scheduler output after the execution of line 20.*

4 Complexity–Delay Tradeoffs when Allowing Linear Tests

In the previous section, we have established the lower bound of $\Omega(log_2 n)$ for guaranteeing $O(n^a)$ GPS-relative delay for $0 \leq a < 1$. However, the computational model is quite restrictive: we only allow the comparisons among 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 $0 \leq a < 1$) delay bounds are achieved for a different and stronger type of delay called *disadvantage delay*[5]. Despite this restriction, the overall result is provably stronger (by Theorem 5) than results (Theorem 1 and 3) in the last section.

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.

[5]The difficulty of obtaining lower bounds in full generality is explained in Section 2.3

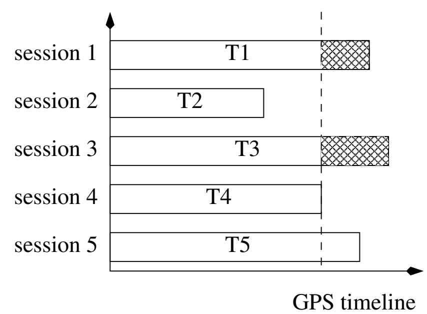

Figure 6: Disadvantage of packet T_4 (the shaded area)

In Fig. 6, 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

$$disadv(T_i) = \sum_{j=0}^{i-1} max(0, F_{T_j}^{GPS} - F_{T_i}^{GPS}) \qquad (3)$$

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.

Lemma 5 *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 larger than its GPS-relative delay.*

Proof: Let $B(T_i)$ and $A(T_i)$ be the set of bits that should have been served before and after $F_{p_i}^{GPS}$

in a GPS scheduler, respectively. Then, under the CBFS condition, the GPS-relative delay of S_i can be written as $\frac{1}{r}max(0, \sum_{j=0}^{i-1} length(A(T_i) \bigcap T_j) - \frac{1}{r}\sum_{j=i+1}^{n} length(B(T_i) \bigcap T_j))$. The disadvantage delay of a packet, on the other hand is $\frac{1}{r}\sum_{j=0}^{i-1} length(A(S_i) \bigcap T_j)$. Obviously, the latter is no larger than the former. □

Remark: The above lemma implies that, for the same amount, guaranteeing disadvantage delay bound is stronger (harder) than guaranteeing GPS-relative delay bound. However, it is only slightly stronger: the disadvantage delay bound of WFQ is zero and that of WF^2Q is also zero if all packets arrive at the same time (so eligibility test [BZ96] is no longer an issue).

Now we are ready to state and prove our main theorem. In 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 4 *The number of linear tests used in the procedure $O(n^a)$-Disadvantage-Scheduler ($0 \leq a < 1$) have a lower bound of $\Omega(nlog_2 n)$ in the worst case.*

Proof: The framework of the proof is the same as those of theorem 1, 2, and 3. A procedure for L-membership test is shown in Fig. 7. Since it is very similar to the program shown in Fig. 4, 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 $nlog_2(\lceil\sqrt{2(n+1)(Kn^a+1)}\rceil) = \frac{a+1}{2}nlog_2 n + o(nlog_2 n)$ comparisons used in line 21 through 29, and (c) $O(n)$ comparisons used line 16 through 20. Since $(a) + (b) + (c) = nlog_2 n + o(nlog_2 n)$, we know that the (a) part must be at least $(1 - \frac{a+1}{2})nlog_2 n = \Omega(nlog_2 n)$.

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, proved next. □

Lemma 6 *Suppose $(x_1, x_2, \cdots, x_n) \in L$. Then right before the execution of line 22, for any packet T_i, $i = 1, 2, \cdots, n$, 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 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 $\frac{L_{max}}{n+1} - 2\delta$, $\frac{2L_{max}}{n+1} - 2\delta$, $\cdots$, $\frac{NL_{max}}{n+1} - 2\delta$. Suppose $N > \lceil\sqrt{2(n+1)(Kn^a+1)}\rceil$. Then $disadv(T_i) > \frac{1}{r}\sum_{j=1}^{N}(\frac{jL_{max}}{n+1} - 2\delta) = \frac{1}{r}(\frac{\frac{1}{2}N(N+1)L_{max}}{n+1} - 2N\delta) > Kn^a\frac{L_{max}}{r}$. This contradicts the guarantee provided by the procedure $O(n^a)$-Disadvantage-Scheduler. Therefore, $N \leq \lceil\sqrt{2(n+1)(Kn^a+1)}\rceil$ □

Compared to Theorem 1 and 3, Theorem 4 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 4 is provably stronger than that of Theorem 1 and 3, shown next.

Theorem 5 *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: Proof of Lemma 3 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 $O(n^a)$. □

5 Linking GPS-relative delay with 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 end-to-end delay bounds. In this section, we show that our lower bound complexity results can indeed be put into the context of providing tight end-to-end delay bounds. This is done by studying the relationship between the GPS-relative delay and the end-to-end delay.

In [SV96b], Stiliadis and Varma defined a general class of latency rate (LR) schedulers (called *servers* in [SV96b]) capable of describing the worst-case behavior of numerous scheduling algorithms. From the viewpoint of a session i, any LR scheduler is charactered 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

```
Procedure L-Membership III
................ same as in Fig. 4 ................
    call Procedure O(n^a)-Disadvantage-Scheduler with
................ same as in Fig. 4 ................
    /* Part II: ''sort'' the output schedule from the scheduler */
    for i:= 2 to ⌈sqrt(2(n+1)(Kn^a+1))⌉ begin
       perform binary insertion of T_i into the list T_1, T_2, ..., T_{i-1} according to their lengths
       /* sort the first ⌈sqrt(2(n+1)(Kn^a+1))⌉ numbers using binary insertion */
    end /* for */
    for i:= ⌈sqrt(2(n+1)(Kn^a+1))⌉+1 to n begin
       binary insertion of T_i into the list T_{i-⌈sqrt(2(n+1)(Kn^a+1))⌉}, ..., T_{i-1} according to lengths
       /* binary insertion into a ''window'' of size ⌈sqrt(2(n+1)(Kn^a+1))⌉ */
    end {for}
................ same as in Fig. 4 ................
```

Figure 7: Algorithm III for L-Membership Test.

LR if for all times t after time τ and until the packets that arrived during this period are serviced,

$$W_{i,j}(\tau, t) \geq max(0, r_i(t - \tau - \Theta_i)) \qquad (4)$$

It has been shown that, for a large class of LR schedulers (including WFQ [PG93], FFQ [SV96a], VC [Zha91], WF^2Q [BZ96], WF^2Q+ [BZ97]), the latency bound of session i, denoted as Θ_i, is

$$\Theta_i = \frac{L_{max}}{r} + \frac{L_{max,i}}{r_i} \qquad (5)$$

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. For (5) to hold, it should be true that the link is not oversubscribed, i.e., $\sum_{i=1}^{n} r_i \leq r$. Note in (5) 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 [SV96b] 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 [SV96b] that the latency bound is further connected to the end-to-end delay bound of session i, denoted as D_i^N, by the following inequality:

$$D_i^N \leq \frac{\sigma_i}{r_i} + \sum_{j=1}^{N} \Theta_i^j \qquad (6)$$

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. Here 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.

We show, in the following theorem, that under a special CBFS condition called $CBFS+$, providing tight GPS-relative delay bound is equivalent to providing tight *latency bound* in any LR scheduler. In $CBFS+$, the j_{th} packet ($j \geq 2$) in session i arrives just at the time the $(j-1)_{th}$ packet finishes service under the GPS scheduler. In other words, each packets in session i arrive just in time to satisfy the CBFS condition. The following theorem is one major step in connecting our complexity results to the complexity of providing tight end-to-end delay bounds.

Theorem 6 *Under the $CBFS+$ condition, an LR scheduler is able to guarantee a GPS-relative delay bound of B for all packets, if and only if, for all $L_{max,i} > 0$, it guarantees a latency bound of $B + \frac{L_{max,i}}{r_i}$ for session i.*

Proof: (if part): Given any packet p, let p' be the very first packet of the session i busy period where p is in. Let W be the total amount of session i traffic that arrive between p and p' (p and p' included). Note that if p arrives to see an empty queue i, then p and p' are the same packet. Let $\epsilon > 0$ be a constant. Suppose p' arrives at time τ. Then according to (4), p must finish service at the time $t = \tau + B + \frac{L_{max,i}+W}{r_i} + \epsilon$ since by time t the server must have accomplished $r_i(t - \tau - B - \frac{L_{max,i}}{r_i} + \epsilon) = W + r_i\epsilon > W$ amount of service. This includes the service of p since W is the total session i traffic between p and p'. Now under the CBFS condition, the GPS virtual finish time of the packet p will be no smaller than $\tau + \frac{W}{r_i}$. So the GPS-relative delay of the packet p will be no larger than $t - \tau - \frac{W}{r_i} = B + \frac{L_{max,i}}{r_i} + \epsilon$. By making $L_{max,i}$ and ϵ arbitrarily close to 0^+, we get the GPS-relative

delay bound of B. Note that we only need the CBFS condition (instead of $CBFS+$) in this part.
(only if part): Suppose that a packet p of size l arrives at time τ to see an empty session i queue. Then under the $CBFS+$ condition, τ is exactly the time that its previous packet finishes service in GPS. So the virtual finish time of p is exactly $\tau + \frac{l}{r_i}$, since its GPS virtual start time is τ. Since its GPS relative delay is no more than B, it must finish service by the time $\tau + \frac{l}{r_i} + B$. So the latency of p is at most $\frac{l}{r_i} + B \leq B + \frac{L_{max,i}}{r}$. □

Remark: We have just shown that, under the CBFS condition, an LR scheduler that provides a tight latency bound also provides a tight GPS-relative delay bound (the "if" part). This in general (without CBFS) is not true: LR schedulers such as FFQ [SV96a], VC [Zha91], WF^2Q [BZ96], and WF^2Q+ [BZ97] all provide a tight latency bound of $\frac{L_{max}}{r} + \frac{L_{max,i}}{r_i}$, but it can be shown that none of them can provide $O(1)$ or even $O(n^a)$ $(0 < a < 1)$ GPS-relative delay in the worst case. However, under the CBFS condition, it can be shown that FFQ and VC become equivalent to WFQ and WF^2Q+ becomes equivalent to WF^2Q, which guarantees tight delay bounds. This is because under the CBFS condition, the GPS time estimation operation of all these algorithms perfects tracks the GPS clock (GPS clock advances in a linear fashion under CBFS!).

6 Conclusions and an Open Problem

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

1. We prove that $\Omega(log_2 n)$ is indeed the per packet complexity lower bound to guarantee $O(1)$ GPS-relative delay (excluding the cost of tracking GPS time), if a scheduling algorithm are only allowed to compare among inputs (equivalently among GPS virtual finish times) in its decision tree. Moreover, we prove that surprisingly 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: 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 than the first one.

3. We show that in a Latency Rate (LR) [SV96b] scheduler, providing tight GPS-relative delay bound of $\frac{L_{max}}{r}$ is equivalent to providing the tight *latency bound* of $\frac{L_{max}}{r} + \frac{L_{max,i}}{r_i}$, under the $CBFS+$ condition. This to certain extent allows us to connect our lower bound results to the complexity that is needed to guarantee end-to-end delay.

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 the following conjecture.

Conjecture 1 *The complexity lower bound for an LR scheduler (introduced in [SV96b]) to achieve a tight latency bound of* $\frac{L_{max}}{r} + \frac{L_{max,i}}{r_i}$ *is* $\Omega(log_2 n)$ *per packet, under the decision tree model that allows linear tests.*

Remark: FFQ, VC, and WF^2Q+ all achieve this latency bound at the complexity of $O(log_2 n)$ per packet, without the restriction of CBFS condition. If the conjecture is true, this implies that these algorithms are asymptotically optimal for this purpose, which is an exciting result! Note that combining our Theorems 1, 2, and 6 only proves this complexity lower bound under the weaker model that allows only comparisons among inputs.

References

[AU73] A. Aho and J. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1973.

[BZ96] J. Bennett and H. Zhang. wf^2q: worst-case fair weighted fair queuing. In *IEEE INFOCOM'96*, March 1996.

[BZ97] J. Bennett and H. Zhang. Hierarchical packet fair queuing algorithms. *IEEE/ACM Transactions on Networking*, 5(5):675–689, 1997.

[DL78] 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.*, 16:413–417, 1978.

[FW78] M. Fredman and B. Weide. On the complexity of computing of measure of $\bigcup[a_i b_i]$. *Communications of the ACM*, 21(7), July 1978.

[Guo01] C. Guo. Ssr: an o(1) time complexity packet scheduler for flows in multi-service packet networks. In *Proc. of Sigcomm'01*, September 2001.

[Hei84] F. Heide. A polynomial linear search algorithm for the n-dimensional knapsack problem. *J. of the ACM*, 31:668–676, 1984.

[Knu98] D. Knuth. *The Art of Computer Programming, Sorting and Searching (Volume 3)*. Addison-Wesley, 1998.

[KSC91] 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*, 9:1265–1279, October 1991.

[PG93] 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*, 1(3):344–357, June 1993.

[SV95] M. Shreedhar and G. Varghese. Efficient fair queuing using deficit round robin. In *Proc. of ACM SIGCOMM'95*, pages 231–242, August 1995.

[SV96a] 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*, pages 104–115, May 1996.

[SV96b] D. Stiliadis and A. Varma. Latency-rate servers: a general model for analysis of traffic scheduling algorithms. In *Proc. of Infocom'96*, March 1996.

[Tur86] J. Turner. New directions in communications (or which way to the information age?). *IEEE Communications Magazine*, 24:8–15, October 1986.

[Yur76] G. Yural. Finding nearest neighbors. *Information Processing Letters*, 5(3):63–65, August 1976.

[Zha91] L. Zhang. Virtualclock: a new traffic control algorithm for packet switching networks. *ACM Transactions on Computer Systems*, 9:101–124, May 1991.

[Zha95] H. Zhang. Service disciplines for guaranteed performance service in packet switching networks. *Proceedings of the IEEE*, 83(10), October 1995.

7 Appendix

Proof of Lemma 1: Consider the 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. □