アルゴリズム 79－7
（2001. 7. 27）

パケット損失を防止するスケジューリングアルゴリズム

古賀 久志
富士通研究所/東京大学大学院理学系研究科情報科学専攻

〒 211-8588 川崎市中原区上小田中 4-1-1
E-mail: koga@flab.fujitsu.co.jp

現在のネットワーク通信におけるパケットロスはルータでバッファが足りなくなるのが原因である。本稿では、ルータでオンラインスケジューラが m 本のキューからパケットをスケジューリングする際に、どれだけバッファがあればパケットロスを防げるかを調べる。少ないバッファ量でパケットロスを防ぐには最大キュー長を小さく抑える必要があり、この新しい問題を我々は Balanced Scheduling Problem (BSP) と名付けた。BSP はタスクが負のコストを持つことを許す新しい負荷分散問題である。
オンラインスケジューリングアルゴリズムの評価は、パケットロスを起こさない最適オフラインアルゴリズムの何倍のバッファがあればパケットロスが起きないかという競合比を使って行った。その結果、(1) いかなる決定的／確率的アルゴリズムも $\Omega(\log m)$-competitive より良くない、(2) GREEDY アルゴリズムは $O(\log m)$-competitive を達成し、ほぼ最適となるという結果を得た。

Balanced Scheduling toward Loss-Free Packet Queuing

Hisashi Koga
Fujitsu Laboratories Ltd. and University of Tokyo.

4-1-1 Kamikodanaka, Nakahara-ku, Kawasaki 211-8588, Japan
E-mail: koga@flab.fujitsu.co.jp

Packet losses in the current best-effort networks take place because of buffer shortage in a router. This paper investigates how many buffers should be prepared in a router enough to eliminate packet losses in the context that an on-line scheduling algorithm in the router must decide the order of transmitting packets among m queues each of which corresponds to a single traffic stream. To exclude packet losses with a small amount of buffers, the maximum queue length must be kept low. This new problem is named the *balanced scheduling problem (BSP)*. By competitive analysis, we judge the power of on-line algorithms from how many times the on-line algorithms must prepare as many buffers as the loss-free off-line algorithm to guarantee no packet loss. The BSP is a new on-line load balancing problem which accompanies tasks with negative loads. To solve an on-line problem which admits tasks to have negative costs is our main theoretical contribution.
Speicifically we show that no deterministic/randomized on-line algorithm is better than $\Omega(\log m)$-competitive. Then we prove a simple greedy algorithm is $\Theta(\log m)$-competitive and nearly optimal.

1 Introduction

Network communication represented by the Internet has been expanded to public steadily during the past decade. However, the current Internet is inadequate for commercial use because of its best-effort natures which admits low throughput, long end-to-end delay and packet losses, when the network links are congested.

In the current best-effort network the end source host has to retransmit the discarded packets like TCP protocol to recover the lost information when a packet loss occurs. Unfortunately this solution has the disadvantage that additional traffic may make the congestion worse, which leads to more awful network performances. For this reason, it is marvelous if one can construct a network environment where the network itself guarantees no packet loss in the first place.

In general packet losses are caused when buffers in a network router run short because of sudden burst traffic. Therefore, there are two means to prevent packet losses:

1. To restrict the amount of the total traffic flowing into the router.
2. To prepare many buffers in the router.

The former approach is called the admission control in the research area of QoS (quality of services) networks. As for the latter approach that is the theme of this paper, avoiding packet losses is possible if the router is ideally given infinite buffers, which is of course an unrealistic assumption. This paper studies the amount of buffers which should be given to a single router to eliminate packet losses in the context that m traffic streams into a router R shares the same output port and a scheduling algorithm in R must decide the order of transmitting packets among m FIFO queues each of which is responsible for exactly one traffic stream (Figure 1). At each time unit t, N_t packets arrive at R. Here N_t depends on t and $N_t \geq 0$. To which traffic stream a packet belongs is identified by a label attached to the packet. According to this label, the packet is once stored into the corresponding FIFO queue. As for the output, R chooses one non-empty queue and outputs the packet at its head per time unit. Therefore the phenomenon that $N_t > 1$ implies some burst traffic is breaking out. This paper assumes a simple fair Complete Buffer Partitioning (CBP [7]) scheme which allocates the same number of buffers to the m queues staticly, i.e. reassigning buffers dynamically among the queues is disallowed. The CBP scheme has the strong advantage of being easily implemented and is used in actual routers.

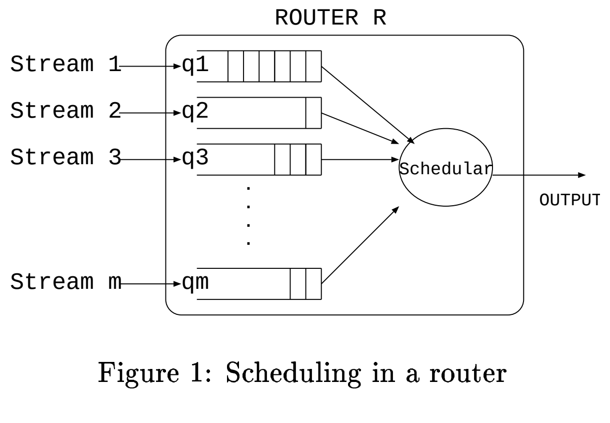

Figure 1: Scheduling in a router

In the above setting, the number of buffers enough to exclude packet losses is equal to m times the maximal queue length where the maximum is taken over the whole period during which the scheduling algorithm in R serves all the packets. This quantity depends on the scheduling policy, so that we can judge the power of scheduling algorithms in terms of the prevention of packet loss from the maximal queue length over the whole scheduling period. For this reason, this paper investigates a new on-line scheduling problem whose purpose is to minimize the maximal queue length over the whole scheduling period. We name this fresh problem the on-line *balanced scheduling problem* (BSP). We would like to emphasize that the basic essence of the BSP to balance some objective function among traffic streams is getting more and more important as QoS is essential in network communications, though many previous theoretical works on on-line scheduling problems try to suppress only the *total* completion time or the *total* waiting time of all the tasks. For example, a promising QoS model named *Proportional Delay Differentiation Service* suggested by Dovrolis *et al.* [5] requires that the weighted delay per packet should be balanced among traffic streams so that each traffic stream may receive a different level of service proportional to its importance.

We evaluate the power of on-line algorithms for the BSP using competitive analysis [8] which compares the performance of an on-line algorithm to that of the optimal off-line algorithm. Concretely, we examine how many times on-line algorithms must prepare as many buffers as the optimal off-line algorithm so as to eliminate packet losses. Let $L_A(\sigma)$ be the maximal queue length over the whole scheduling period when a scheduling algorithm A serves a packet arrival sequence σ. An on-line algorithm A is called c-competitive if $L_A(\sigma) \leq c \times L_{opt}(\sigma)$ for any packet arrival sequence σ. Here *opt* is the optimal off-line algorithm which can know the entire packet arrival sequence in advance. Competitive analysis is particularly suited for problems handling packet losses for the two reasons below. It is important to find application areas for which competitive analysis is effective, because it is often said that competitive analysis is too much theoretical.

1. Under burst traffic, assuming a probabilistic distribution over packet arrivals is difficult.
2. To warrant no packet loss, it is essential to examine the worst case like competitive analysis rather than the average case.

First in Section 2, the balanced scheduling problem is formally defined. Then we show that the competitiveness of m becomes a trivial upper bound for the BSP. In Section 3 the lower bounds of the competitiveness of on-line algorithms are investigated. We show that any deterministic on-line algorithm cannot exceed the competitiveness of $\Omega(\log m)$. Then we prove that randomization technique cannot overcome this lower bound of $\Omega(\log m)$. Specifically the popular algorithm *ROUND ROBIN* is not better than the trivial upper bound of m-competitiveness. In Section 4, the greedy algorithm named *GREEDY* which always selects the longest queue at each time unit is studied. We show that *GREEDY* is a nearly optimal on-line algorithm and achieves the competitiveness of $O(\log m)$. Thus, *GREEDY* is by far superior to *ROUND ROBIN* regarding to the prevention of packet losses.

However, this paper does not intend to argue that *ROUND ROBIN* is useless, because *ROUND ROBIN* achieves the throughput fairness among the streams which *GREEDY* does not. For practical use, it would be reasonable to combine *GREEDY* with another scheduling algorithm. That is, *GREEDY* should be adopted after the length of the longest queue goes beyond some threshold value and another scheduling algorithm like *ROUND ROBIN* should be used till then.

1.1 Related Work

Theoretically the BSP is related to the so-called on-line load balancing problem initiated by Graham [6]. In the load balancing problem, given m servers, we must assign each incoming task to one of the m servers in such a way that the maximum load on the servers is minimized. Each task arrives one by one and holds its own positive load vector of length m whose coordinates indicate the increase in load when it runs on the corresponding server. Many variants of on-line load balancing problems have been considered so far. In the identical machines model [6] all the coordinates of a load vector are the same. In the restricted machines model [4], each task can be handled only by a subset of the servers, though all the coordinates of a load vector are still the same. In the unrelated machines model [1], there is no constraint on the coordinates of load vectors except that they are positive. The natural greedy algorithm becomes $(2-\frac{1}{m})$-competitive in the identical machines model [6], $\Theta(\log m)$-competitive in the restricted machines model [4] and $\Theta(m)$-competitive in the unrelated machines model [1] respectively. The temporal tasks model of the on-line load balancing problem [2][3] assumes that tasks have a limited duration and disappear after their duration. The greedy algorithm becomes $\Theta(m^{\frac{2}{3}})$-competitive [2] in the temporal tasks model.

The BSP differs greatly from the traditional on-line load balancing problem in that load of all the servers must be balanced by selecting departing packets. By the procedure illustrated in Figure 2, the BSP is transformed to a new on-line load balancing problem that must face two difficulties that all the previous models do not have:

- The coordinates of a load vector of a task may take a negative value.
- A subset of servers on which a individual task can run depends on the behavior of the scheduling algorithm.

Especially, handling the former difficulty is a large contribution of this paper, because tasks with negative costs usually make on-line problems so intricate. Roughly speaking, the transformed BSP seems to be the extension of the restricted machines model that admits tasks with negative costs. Because *GREEDY* is $\Theta(\log m)$-competitive for the BSP, the complexity of the BSP does not differ from that of the restricted machines model of the on-line load balancing problem interestingly, despite tasks with negative costs are introduced. By contrast, *GREEDY* achieves a quite smaller upper bound in the BSP than in the temporal tasks model, though both problems admit tasks to leave servers. We infer the reason for this difference is that the scheduler decides aggressively the finish time of tasks in the BSP unlike the temporal tasks model. On the other hand, constructing the optimal off-line algorithm for the BSP is very difficult like the temporal tasks model.

1. A packet arrival in the BSP is mapped to a task of size 1 which only a single specific server can process.
2. A packet output from R in the BSP is mapped to a task of size -1 which only non-idol servers can process.

Figure 2: Transforming Procedure of the BSP

2 Balanced Scheduling Problem

The balanced scheduling problem (BSP) is formally defined as follows. We are given m FIFO queues $q_1, q_2, \ldots q_m$ in a router R and a sequence of packet arrivals at R. Initially at time 0, all the m queues are empty. At each time $t > 0$, N_t packets expressed by m-tuples $(N_t^1, N_t^2, \ldots N_t^m)$ first arrive at R, where each N_t^i is a non-negative integer and $N_t = \sum_{i=1}^m N_t^i$. The packets that have just arrived are stored into the m queues such that N_t^i packets go into q_i for $1 \le i \le m$. Then a scheduling algorithm A operating in R selects one non-empty queue and outputs a packet at its head unless all the queues are empty. We assume that at least one queue is not empty until the end of the whole scheduling period. This assumption does not lose generality, because, if there is a time when all the queues are empty, we can partition the packet arrival sequence into multiple subsequences for each of which the BSP is separately solved.

Let $l_A^i(t)$ be the length of q_i at time t in the running of the scheduling algorithm A after arriving packets are stored into the corresponding queue. Since the length of a queue before A outputs a packet from R may differ from that after A outputs it within the same time instance t, we distinguish the time after the output of the packet by attaching a superscript a to t like t^a specially when necessary. Since the maximum instantaneous queue length must be considered to avoid packet losses, we normally pay attention to the length of the queues before A outputs a packet, i.e. $l_A^i(t)$, not $l_A^i(t^a)$. The notation of t^a is used only for the analysis of algorithms. The maximal queue length at time t by A is defined as

$$l_A(t) = \max_{1\leq i\leq m} \{l_A^i(t)\}.$$

Let σ be a sequence of packet arrivals and $|\sigma|$ be the time of the last arrival. Then, the maximal queue length over the whole scheduling period for σ by A is defined as

$$L_A(\sigma) = \max_{0\leq t\leq|\sigma|} l_A(t). \tag{1}$$

The purpose of the BSP is to find a scheduling algorithm A that reduces $L_A(\sigma)$.

Let us describe the total number of packets stored in the m queues at time t as $C(t)$. Note that $C(t)$ does not depend on the scheduling algorithm, because the number of packets that have left R before t is independent of the scheduling algorithm. For any on-line algorithm A, $l_A(t) \leq C(t)$ trivially. For the optimal off-line algorithm *opt*, $l_{opt}(t) \geq \frac{C(t)}{m}$ because $C(t)$ packets are distributed among the m queues. Therefore it holds that $l_A(t) \leq m \cdot l_{opt}(t)$ for any t. This inequality readily yields $L_A(\sigma) \leq m \cdot L_{opt}(\sigma)$ for any σ. Therefore,

Theorem 1 *Any on-line algorithm for the BSP is m-competitive at worst.*

3 The Lower Bound

3.1 General Lower Bounds

At the beginning, we obtain lower bounds for general on-line algorithms. A technique similar to the lower bound technique [4] for the restricted machines model for the on-line load balancing problem is exploited. In the proof, an adversary constructs a packet arrival sequence σ which annoys on-line algorithms.

3.1.1 The Deterministic Lower Bound

Theorem 2 *Let A be any deterministic on-line algorithm for the BSP. Then A is not better than $(1 + \lfloor\log_2 m\rfloor)$-competitive.*

Proof: Let j be the largest integer satisfying $2^j \leq m$, i.e. $j = \lfloor\log_2 m\rfloor$. The adversary constructs σ by dividing it into $j+1$ phases. For $1 \leq k \leq j$, the k-th phase starts at time $1 + \sum_{r=1}^{k-1} 2^{j-r}$ and lasts for 2^{j-k} time units. The final $(j+1)$-th phase starts at time $1 + \sum_{r=1}^{j} 2^{j-r}$ and continues only for one time unit. For example, the first phase starts at time 1 and finishes at time 2^{j-1}, the second phase starts at time $1 + 2^{j-1}$ and finishes at time $2^{j-1} + 2^{j-2}$ and so on. How to construct σ in the k-th phase is shown below in detail.

Step 1: 2^{j-k+1} packets arrive at R when the k-th phase starts, so that exactly one new packet is assigned to each of the 2^{j-k+1} longest queues in A's running. Ties are broken arbitrarily. Note that the adversary predicts accurately the lengths of all the m queues since A is deterministic.

Step 2: No more packet arrives during the k-th phase after Step 1.

σ has a property that the number of leaving packets in the k-th phase is equal to the number of arriving packets in the $(k+1)$-th phase for $1 \leq k \leq j$. From now on, we show that $L_A(\sigma)$ reaches $1 + \lfloor\log_2 m\rfloor$ while $L_{opt}(\sigma) = 1$. Let T_k be the time when the k-th phase begins.

As for *opt*, in the k-th phase, *opt* selects the 2^{j-k} queues to which a packet is assigned in the $(k+1)$-th phase exactly once. This scheduling keeps the invariant that, at the beginning of every phase, exactly 2^j queues hold just one packet and the rest of the queues are empty. That is, for any t, $l_{opt}(t) = 1$. Hence, $L_{opt}(\sigma) = 1$.

As for A, by induction on the index of phases k, we prove that the 2^{j-k+1} longest queues have a length of k at time T_k. The base case is trivial, since 2^j queues have a length of 1 at time 1 from the structure of σ. Assume that the 2^{j-k+1} longest queues have a length of k at time T_k in A's running. 2^{j-k} packets leave R in the k-th phase because its duration is 2^{j-k}. Thus, at least $2^{j-k+1} - 2^{j-k} =$

2^{j-k} queues still have a length of k when the k-th phase terminates. Because the 2^{j-k} longest queues increase their lengths by 1 at the beginning of the next phase, it follows that the 2^{j-k} longest queues have a length of $k+1$ at T_{k+1}, which completes the proof of the induction step. Thus it holds that $l_A(T_k) = k$ for $1 \leq k \leq j+1$. Therefore,

$$L_A(\sigma) = 1 + \lfloor \log_2 m \rfloor, \qquad (2)$$

which completes the proof of Theorem 2. □

3.1.2 The Randomized Lower Bound

The proof technique in Theorem 2 enables us to derive a randomized lower bound also.

Theorem 3 *Let A be any randomized on-line algorithm for the BSP. Then A is not better than* $(1 + \frac{\lfloor \log_2 m \rfloor}{2})$*-competitive against the oblivious adversary.*

The proof is omitted due to space limitations.

3.2 Lower Bound for ROUND ROBIN

Next we examine the lower bound of a specific popular algorithm *ROUND ROBIN*. *ROUND ROBIN* cannot exceed the trivial upper bound of m. This contrasts with another popular algorithm *GREEDY* for which the upper bound of $\Theta(\log m)$-competitiveness is proved in the next section.

Algorithm ROUND ROBIN: Initially the algorithm may select any non-empty queue. On condition that the algorithm selects a queue q_i at time t, the queue selected at time $t+1$ is the queue $q_{(r+i) \bmod m}$. Here r is the minimum positive integer satisfying the condition that $q_{(r+i) \bmod m}$ is not empty.

Theorem 4 *ROUND ROBIN is not better than m-competitive.*

Proof: Without loss of generality, we assume that *ROUND ROBIN* selects q_1 initially. Again an adversary passes a bad packet arrival sequence σ to *ROUND ROBIN*. σ is constructed in the next way.

Step1: At time 1, m^2 packets arrive at R such that m packets are assigned to each q_i for $1 \leq i \leq m$.

Step2: At time $km+1$ for $1 \leq k \leq m$, m new packets are assigned to q_m.

The analysis advances by dividing the scheduling period into $m+1$ phases. The k-th phase begins at time $(k-1)m+1$ and finishes at time km for $1 \leq k \leq m+1$. Note that the number of packets that leave R in the k-th phase is equal to the number of arriving packets in the $(k+1)$-th phase for $1 \leq k \leq m$. We only write RR instead of the algorithm name *ROUND ROBIN* when it appears in mathematical expressions to save the space.

The optimal off-line algorithm *opt* chooses q_m all the time. This assures that the length of q_m equals 0 at the end of each phase and that it increases to m at the beginning of the k-th phase for $k > 2$. The lengths of all the other queues take a constant value of m all the time. Hence it holds that $l_{opt}(t) = m$ for an arbitrary time t. Thus.

$$L_{opt}(\sigma) = m. \qquad (3)$$

On the other hand, *ROUND ROBIN* selects all the m queues once in each phase except the last $(m+1)$-th phase, since all the queues have at least one packet at the beginning of the k-th phase ($k \leq m$). Since the algorithm initially selects q_1, q_m is always the longest queue in the running of *ROUND ROBIN*. The length of q_m at the beginning of the k-th phase is calculated as:

$$l_{RR}((k-1)m+1) = mk - (k-1) = (m-1)k + 1.$$

This value reaches to the maximum when $m+1$ is substituted for k. Hence,

$$L_{RR}(\sigma) = m^2. \qquad (4)$$

Comparing (4) with (3) completes the proof. □

4 The Upper Bound

This section analyzes the performance of a specific algorithm *GREEDY*. This algorithm has the advantage of being very simple.

Algorithm GREEDY: At any time, *GREEDY* selects the longest queue. Ties are broken arbitrary.

Since simple greedy policies are analyzed in many load-balancing problems [1][2][4][6], measuring the performance of *GREEDY* enables us to estimate the relative difficulty of the BSP to other problems.

Theorem 5
GREEDY is $(3 + \lceil \log_2 m \rceil)$*-competitive.*

Theorem 5 together with Theorem 2 claims that *GREEDY* is a nearly optimal on-line algorithm. We extend the proof technique to derive the upper bound for the restricted machines model of the on-line load balancing by Azar *et al.* [4].

Proof: We introduce a function named *gap* which maps packets in a FIFO queue in *GREEDY*'s running to some integers. Let σ be an arbitrary packet arrival sequence. Suppose p be the r-th packet from the top (i.e. the output port) of a FIFO queue q_i at time t in *GREEDY*'s running. Then the gap of the packet p at time t, denoted by $gap(p,t)$ is defined as

$$r - l^i_{opt}(t).$$

Intuitively the function *gap* presents the height of packets in a FIFO queue in *GREEDY*'s running relative to the length of the corresponding queue in *opt*'s running.

According to the value of *gap*, we partition FIFO queues in *GREEDY*'s running into layers as follows. Denote $L_{opt}(\sigma)$ by l. See Figure 3 (A). The k-th layer of a queue q_i at time t consists of packets p stored in q_i at t in *GREEDY*'s running such that $(k-1)l+1 \le gap(p,t) \le kl$. The number of packets contained in the k-th layer of q_i is expressed as $W^i_k(t)$. The next property about $W^i_k(t)$ is crucial in the analysis of *GREEDY*.

Lemma 1 *$\forall k \ge 1$, $W^i_k(t) = l$ if $W^i_{k+1}(t) > 0$.*

Proof of Lemma 1: Because $W^i_{k+1}(t) > 0$, $l^i_G(t) > l^i_{opt}(t) + kl$. Hence, the number of packets in q_i whose gaps are greater than or equal to $(k-1)l+1$ but less than or equal to kl is exactly l in *GREEDY*'s running. □

Corollary 1 *For any $k \ge 1$, $W^i_k(t) \ge W^i_{k+1}(t)$.*

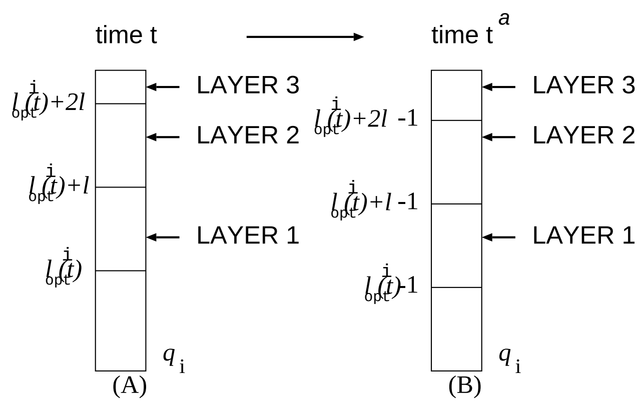

Figure 3: Partition of a queue into layers

Furthermore, the following notations are required to proceed the proof.

- $W_k(t) = \sum_{i=1}^m W^i_k(t)$.
- $R^i_k(t) = \sum_{j>k} W^i_j(t)$.
- $R_k(t) = \sum_{j>k} W_j(t)$.

$W_k(t)$ presents the total number of packets contained in the k-th layer over all the m queues, while $R_k(t)$ presents the total number of packets contained in the layers strictly higher than the k-th layer over all the m queues. $R^i_k(t)$ is equal to the total number of packets in q_i contained in the layers strictly higher than the k-th layer. Therefore, it holds, for any $k \ge 1$, that

$$R_{k+1}(t) = R_k(t) - W_{k+1}(t). \tag{5}$$

Note that $W_k(t^a) = W_k(t+1)$ and $R_k(t^a) = R_k(t+1)$ because the number of packets in each layer of a certain queue q_i is not affected even if the same number of packets arrive at q_i at the beginning of time $t+1$ both in *opt* and in *GREEDY*. By contrast $W_k(t^a)$ ($R_k(t^a)$) may be different from $W_k(t)$ ($R_k(t)$ respectively) depending on the queues selected by the two algorithms at time t.

Our strategy is to compare the simultaneous running of the two algorithms *GREEDY* and *opt* on an arbitrary packet arrival sequence σ and to prove the next relation is maintained all the time for all $k \ge 1$.

$$W_k(t) \ge R_k(t). \tag{6}$$

Assume that (6) is correct. Then, from (5) and (6), we have $R_{k+1}(t) = R_k(t) - W_{k+1}(t) \le R_k(t) - R_{k+1}(t)$. Thus, $R_{k+1}(t) \le \frac{1}{2}R_k(t)$.

Then, by applying this inequality $\lceil \log_2 m \rceil$ times, the next inequality is derived. Note that $R_1(t) \le ml$, because $L_{opt}(\sigma) = l$.

$$\begin{aligned} R_{\lceil \log_2 m \rceil + 1}(t) &\le (\frac{1}{2})^{\lceil \log_2 m \rceil} R_1(t) \\ &= \frac{1}{m} R_1(t) \le \frac{ml}{m} = l. \end{aligned}$$

Therefore the number of packets included in the layers strictly higher than the $(\lceil \log_2 m \rceil + 1)$-th layer is at most l. As the result, the length of the longest queue in *GREEDY*'s running at time t is bounded from above as follows.

$$\begin{aligned} l_G(t) &\le l_{opt}(t) + (\lceil \log_2 m \rceil + 1)l + l \\ &\le (3 + \lceil \log_2 m \rceil)l. \end{aligned} \tag{7}$$

Since (7) holds without regard to time t, the proof of Theorem 5 is complete.

From now on we verify (6) for all $k \geq 1$ and for any t. Pick up an arbitrary positive integer as k. The proof makes use of the induction on time t. As for the base case, (6) is trivial at time 1 before the packet is output, since all the m queues have the same number of packets both in *opt* and in *GREEDY*.

In the next, suppose that $W_k(t) \geq R_k(t)$ at t before the two algorithms select queues from which a packet is output. It suffices to show that

$$W_k(t^a) \geq R_k(t^a),$$

because $W_k(t+1) = W_k(t^a)$ and $R_k(t+1) = R_k(t^a)$. Assume q_i is the queue selected by *opt* and q_j is the one selected by *GREEDY* at t. If q_i is identical with q_j, it is obvious that $W_k(t^a) = W_k(t) \geq R_k(t) = R_k(t^a)$. Let us suppose $q_i \neq q_j$ hereafter. Because $W_k^h(t) = W_k^h(t^a)$ and $R_k^h(t) = R_k^h(t^a)$ for any queue q_h except q_i and q_j, we may concentrate on how q_i and q_j behave only. There are two cases depending on whether $l_G^i(t) < l_{opt}^i(t)$ or not.

(Case I) Suppose that $l_G^i(t) < l_{opt}^i(t)$: Because $l_G^i(t^a) \leq l_{opt}^i(t^a)$, $W_k^i(t) = W_k^i(t^a) = 0$ and $R_k^i(t) = R_k^i(t^a) = 0$. Hence $W_k^i(t)$ and $R_k^i(t)$ does not change.

Now let us consider the behavior of q_j. If $l_G^j(t) \leq l_{opt}^j(t)$, $l_G^j(t^a) < l_{opt}^j(t^a)$. Thus $W_k^j(t) = W_k^j(t^a) = 0$ and $R_k^j(t) = R_k^j(t^a) = 0$ and (6) obviously holds for this case. Next consider the case when $l_G^j(t) > l_{opt}^j(t)$. Let X be $\lceil \frac{l_G^j(t) - l_{opt}^j(t)}{l} \rceil$. From the definition of $W_k^j(t)$ and $R_k^j(t)$, the next arithmetic formulas are obtained.

$$W_k^j(t^a) = \begin{cases} W_k^j(t), & \text{if } k \neq X \\ W_k^j(t) - 1, & \text{if } k = X \end{cases} \tag{8}$$

$$R_k^j(t^a) = \begin{cases} R_k^j(t) - 1, & \text{if } k < X \\ R_k^j(t), & \text{if } k \geq X \end{cases} \tag{9}$$

From (8) and (9), if $k \neq X$, $W_k^j(t^a) = W_k^j(t)$ and $R_k^j(t^a) \leq R_k^j(t)$ so that (6) is proved. If $k = X$, because q_j is the longest at t in *GREEDY*'s running, we have, for all integers h satisfying $1 \leq h \leq m$,

$$\begin{aligned} l_G^h(t^a) &\leq l_G^h(t) \leq l_G^j(t) \\ &\leq l_{opt}^j(t) + Xl \leq (X+1)l. \end{aligned} \tag{10}$$

Since $l_{opt}^h(t^a) \geq 0$, the $(X+2)$-th layers in all the m queues contain no packet at t^a from (10). Hence, from Corollary 1, $W_X(t^a) \geq W_{X+1}(t^a) = R_X(t^a)$. This finishes the proof for this case.

(Case II) Suppose that $l_G^i(t) \geq l_{opt}^i(t)$: First assume that $l_G^j(t) \leq l_{opt}^j(t) \leq l$. Since q_j is the longest at t in *GREEDY*'s running, the lengths of all the queues are not larger than l at t (and hence at t^a also) in *GREEDY*'s running. Therefore $W_{k'}(t^a) = 0$ for any $k' \geq 2$. Hence the next inequality holds for any $k \geq 1$: $R_k(t^a) = \sum_{k'>k} W_{k'}(t^a) \leq \sum_{k'>1} W_{k'}(t^a) = 0 \leq W_k(t^a)$. Thus (6) is assured in this case.

Finally we go on to the case $l_G^j(t) > l_{opt}^j(t)$. Let Y be $\lfloor \frac{l_G^i(t) - l_{opt}^i(t)}{l} \rfloor + 1$. From the definition of $W_k^i(t)$ and $R_k^i(t)$, the next arithmetic expressions are obtained. See Figure 3 (B).

$$W_k^i(t^a) = \begin{cases} W_k^i(t), & \text{if } k \neq Y \\ W_k^i(t) + 1, & \text{if } k = Y \end{cases} \tag{11}$$

$$R_k^i(t^a) = \begin{cases} R_k^i(t) + 1, & \text{if } k < Y \\ R_k^i(t), & \text{if } k \geq Y \end{cases} \tag{12}$$

From (8), (9), (11), (12) and the assumption that $W_k(t) \geq R_k(t)$ at t, in order to break the relation (6) at t^a, at least either of the next two conditions need to be satisfied.

1. $W_k^j(t^a) = W_k^j(t) - 1$.
2. $R_k^i(t^a) = R_k^i(t) + 1$.

From now on, we show that (6) is preserved even if either of the above conditions takes place.

1. Suppose that $W_k^j(t^a) = W_k^j(t) - 1$. From (8), k must be X. Hence $l_G^j(t) \leq l_{opt}^j(t) + lX$. We show the $(X+2)$-th layers of all the m queues are empty at t^a by contradiction. Since $l_G^j(t) \leq l_{opt}^j(t) + lX$, $l_G^j(t^a) = l_G^j(t) - 1 \leq l_{opt}^j(t^a) + lX$ and the $(X+2)$-th layer of q_j is empty at t^a. Assume there exists a queue $q_h (\neq q_j)$ whose $(X+2)$-th layer contains some packets at t^a. Since q_h is not selected by *GREEDY* at t, we have

$$\begin{aligned} l_G^h(t) &= l_G^h(t^a) \\ &\geq (X+1)l + 1 \\ &\quad \text{(the } (X+2)\text{-th layer of } q_h \text{ is not empty)} \\ &> l_{opt}^j(t) + lX \geq l_G^j(t). \end{aligned}$$

This contradicts with the fact that q_j is selected by *GREEDY* at t. Thus the $(X+2)$-th layers of all the m queues must be empty at t^a. Thus, by applying Corollary 1, $W_X(t^a) \geq W_{X+1}(t^a) = R_X(t^a)$, which shows (6) holds for this case.

2. Suppose that $R^i_k(t^a) = R^i_k(t) + 1$. Since $R^i_k(t^a) \geq 1$, we have $l^i_G(t) = l^i_G(t^a) \geq kl + 1$. Since *GREEDY* selects not q_i but q_j, we have

$$l^j_G(t) \geq l^i_G(t) \geq kl + 1 \geq l^j_{opt}(t) + (k-1)l + 1.$$

Especially if $l^j_{opt}(t) + (k-1)l + 1 \leq l^j_G(t) \leq l^j_{opt}(t) + kl$, we can show that the $(k+2)$-th layers of all the m queues are empty at t^a exactly in the same way as the previous paragraph. Thus, by applying Corollary 1, $W_k(t^a) \geq W_{k+1}(t^a) = R_k(t^a)$, which shows (6) holds for this case.

By contrast, if $l^j_G(t) > l^j_{opt}(t) + kl$, we have $R^j_k(t) > 0$. Hence, $R^j_k(t^a) = R^j_k(t) - 1$ after *GREEDY* outputs a packet at t from q_j. By comparing this with (9) we have $k < X$.

$$\begin{aligned} R_k(t^a) &= R^i_k(t^a) + R^j_k(t^a) + \sum_{h \neq i,j} R^h_k(t^a) \\ &= (R^i_k(t) + 1) + (R^j_k(t) - 1) + \sum_{h \neq i,j} R^h_k(t) \\ &= R_k(t) \qquad (13) \end{aligned}$$

About $W_k(t)$, as $k \neq X$, it follows that $W^i_k(t^a) \geq W^i_k(t)$ from (11) and that $W^j_k(t^a) = W^j_k(t)$ from (8). Hence,

$$W_k(t^a) \geq W_k(t). \qquad (14)$$

From (13) and (14), it follows that $W_k(t^a) \geq W_k(t) \geq R_k(t) = R_k(t^a)$. Thus, we have proved (6) for all the possible cases, the entire proof of Theorem 5 ends. $\square$

5 Conclusion

This paper investigates the balanced scheduling problem in order to evaluate power of scheduling algorithms running in a router in terms of prevention of packet losses. The BSP problem is a fresh on-line load balancing problem that faces new difficulties that tasks with negative costs have to be served. We prove a simple greedy algorithm is $O(\log m)$-competitive, where m is the number of queues. We also present the lower bound of $\Omega(\log m)$-competitiveness. Though we consider the BSP in association with packet losses in a router, the BSP will have many applications ranging in wide areas because of its simple structure.

There are many open problems regarding the BSP. One important open problem is to find the optimal off-line algorithm concretely. This enables us to compute the exact number of buffers in order for *GREEDY* to eliminate packet losses, since this paper has given the competitiveness of *GREEDY*. From the viewpoint of application research fields, the problems below are worth pursuing.

- Changing the amount of buffers assigned to each queue. In practical QoS networks, it is common that each traffic is given a different number of buffers proportional to its rate for efficient buffer consumption. In this model, the value of one single buffer varies for each queue according to how many buffers are prepared for it.
- Extending the BSP to dynamic buffer allocation policy. In this model, a communication stream with higher priority can also use the buffer memories prepared for the streams with lower priority.

References

[1] J. Aspens, Y. Azar, A. Fiat, S. Plotkin, and O .Waarts. On-line load balancing with applications to machine scheduling and virtual circuit routing. *Journal of ACM*, (44):486–504, 1997.

[2] Y. Azar, A.Z. Broder, and A.R. Karlin. On-line load balancing. In *Proceedings of 33rd Symposium on Foundations of Computer Science*, pages 218–225, 1992.

[3] Y. Azar, B. Kalyanasundaram, S. Plotkin, K.R. Pruhs, and O. Waarts. On-line load balancing of temporary tasks. *Journal of Algorithms*, 22:93–110, 1997.

[4] Y. Azar, J. Naor, and R. Rom. The competitiveness of on-line assignments. In *Proceedings of 3rd ACM-SIAM Symposium on Discrete Algorithms*, pages 203–210, 1992.

[5] C. Dovrolis, D. Stiladis, and P. Ramanathan. Proportional differentiated services: Delay differentiation and packet scheduling. In *Proceedings of ACM SIGCOMM'99*, 1999.

[6] R.L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

[7] A.M. Lin and J.A. Silvester. Priority queuing strategies and buffer allocation protocols for traffic control at an ATM integrated broadband switching system. *IEEE Journal on Selected Areas in Communications*, 9:1524–1536, 1991.

[8] R.E. Tarjan and D.D. Sleator. Amortized efficiency of list update and paging rules. *Communication of the ACM*, 28:202–208, 1985.