Algorithmica (2004) 38: 363–376
DOI: 10.1007/s00453-003-1064-z

Algorithmica
© 2003 Springer-Verlag New York Inc.

Balanced Scheduling toward Loss-Free Packet Queuing and Delay Fairness[1]

Rudolf Fleischer[2] and Hisashi Koga[3]

Abstract. In current networks, packet losses can occur if routers do not provide sufficiently large buffers. This paper studies how many buffers should be provided in a router to eliminate packet losses. We assume a network router has m incoming queues, each corresponding to a single traffic stream, and must schedule at any time on-line from which queue to take the next packet to send out. To exclude packet losses with a small amount of buffers, the maximum queue length must be kept low over the entire scheduling period. We call this new on-line problem the *balanced scheduling problem* (*BSP*). By competitive analysis, we measure the power of on-line scheduling algorithms to prevent packet losses. We show that a simple greedy algorithm is $\Theta(\log m)$-competitive which is asymptotically optimal, while `Round-Robin` scheduling is not better than m-competitive, as actually is any deterministic on-line algorithm for BSP. We also give a polynomial time algorithm for solving off-line BSP optimally.

We also study another on-line balancing problem that tries to balance the delay among the m traffic streams.

Key Words. Packet loss, Delay, Competitive analysis, Load balancing, Routing.

1. Introduction.

Network usage of the Internet by the general public has steadily increased in the past decade. In particular, commercial use of the Internet has risen dramatically. However, the current Internet is inadequate for commercial use because of its best-effort nature which admits packet losses when the network links are congested. In the current situation, a source host has to retransmit the discarded packets when a packet loss happens to recover the lost data due to buffer size limitations, like TCP (or another best-effort) protocol. Unfortunately, this retransmission of data has the disadvantage of making the congestion worse by adding more traffic to the network. For this reason, networks without packet losses in the first place would be highly desirable.

In general, packet losses occur when buffers in a network router overflow because of sudden burst traffic. There are two ways to prevent these packet losses: (i) to restrict the total number of packets flowing into a router and/or (ii) to provide sufficiently large buffers in the routers.

The former approach is called *admission control* [13], [17] in the research area of QoS (quality of service) networks. The latter approach is the main theme of this paper. Of course, there are no packet losses if the router is given infinite size buffers. We study, more realistically, the amount of buffers that should be given to a router to eliminate packet losses when m traffic streams flowing into a router R are sharing the same output

[1] The work by R. Fleischer described in this paper was partially supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. HKUST6010/01E).
[2] Department of Computer Science, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong. rudolf@cs.ust.hk.
[3] Graduate School of Information Systems, University of Electro-Communications, Tokyo, Japan. koga@is.uec.ac.jp.

Received February 8, 2002; revised January 22, 2003. Communicated by T. Takaoka.
Online publication November 10, 2003.

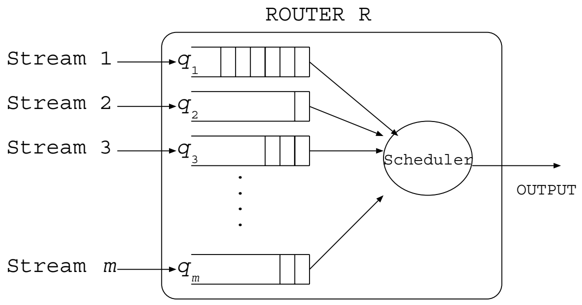

Fig. 1. Scheduling in a router.

port and a scheduling algorithm in R must decide the order of transmitting packets among the m FIFO queues handling the m input streams (see Figure 1). At each time t (for convenience, we measure time in integer units), N_t packets arrive at R. Here N_t depends on t, and $N_t \geq 0$. $N_t > 1$ indicates burst traffic. Which traffic stream a packet belongs to is identified by a label attached to the packet. According to this label, the packet is stored into the corresponding FIFO queue.

After the new packets have arrived, R chooses exactly one non-empty queue and outputs its head packet (if all queues are empty nothing happens). We assume a simple and fair Complete Buffer Partitioning (CBP [16]) scheme that fairly allocates static buffers of the same size to the m queues, i.e., we do not reassign buffers dynamically among the queues. The CBP scheme has the advantage that it can easily be implemented.

To prevent buffer overflow, the buffer size should be chosen larger than the maximum queue length, where the maximum is taken over the entire period during which the scheduling algorithm in R serves all the packets. Of course, for a given packet arrival sequence this quantity depends on the scheduling policy. So we can judge the power of scheduling algorithms to prevent packet losses from the maximum queue length they incur. The *balanced scheduling problem* (*BSP*) is a new problem whose objective is to minimize the maximum queue length.

In the *on-line BSP*, the scheduling algorithm A must decide on-line, i.e., without knowledge of future packet arrivals, which packet to send out next. We evaluate the power of on-line scheduling algorithms using competitive analysis [8], [11] which compares the performance of an on-line algorithm with that of an optimal off-line algorithm `OPT` that knows the entire packet arrival sequence in advance and can therefore plan ahead much better when deciding on the next packet to send out. Let $L_A(\sigma)$ be the maximum queue length over the entire scheduling period, where σ is the packet arrival sequence. For a constant $c \geq 1$, the on-line algorithm A is *c-competitive* if $L_A(\sigma) \leq c \cdot L_{\mathrm{OPT}}(\sigma)$, for any σ.

In Section 2 we define the BSP more formally, and we show that any on-line algorithm is m-competitive for BSP. In Section 3 we describe an optimal off-line algorithm for BSP which runs in time polynomial in the total number of packets. In Section 4 we give lower bounds on the competitiveness of on-line algorithms for BSP. We show that no deterministic or randomized on-line algorithm can be better than $\Omega(\log m)$-competitive. We also show that the popular `Round-Robin` scheduling algorithm is only m-competitive, as is any algorithm. In Section 5 we study the greedy algorithm LQF (Longest Queue First) that at any time always selects the longest queue. We show that LQF achieves $(3 + \lceil \log_2 m \rceil)$-competitiveness and becomes asymptotically optimal. Thus, LQF is far superior to Round-Robin. Because we have the optimal off-line solution at hand, it is possible to obtain the actual buffer size sufficient for the on-line algorithms to run without packet losses from their competitive ratios, given any input sequence.

We note that the main idea of BSP to balance some objective function among traffic streams is getting more and more attention in QoS network design, though many previous on-line scheduling problems tried to suppress only the *total* completion time or the *total* waiting time of all the tasks [12], [15].

While BSP aims at balancing the queue length, balancing other objective functions (like delay, for example), is also important for certain practical applications. For example, a promising QoS model named *Proportional Delay Differentiation Service* [10] 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. Motivated by this work, in Section 6 we consider an on-line problem with the objective of decreasing the maximum sum-of-delays incurred in a single queue. This problem is called the *delay balanced scheduling problem* (*DBSP*). It turns out that no deterministic on-line scheduling algorithm can be better than $\Omega(\log m)$-competitive for DBSP.

Note that we do not want to argue that Round-Robin is useless in practice. For example, Round-Robin achieves throughput fairness among the streams which LQF does not. For practical purposes, a good strategy might be to use Round-Robin until the length of some queue reaches a certain threshold, and then switch to LQF.

1.1. *Related Work.* Bar-Noy et al. [5] investigated the same on-line problem as BSP independently of our work. They first analyzed the continuous model which we do not deal with, and then studied the discrete model which coincides with our model. Regarding the discrete model, they gave the same lower bound with the same proof as we do in Section 4. They also gave a polynomial optimal off-line algorithm which has some similarity with the algorithm we present in Section 3. Then they showed that the cost of a generic discrete algorithm simulating the continuous LQF algorithm is bounded from above by $H_m \cdot L_{\mathrm{OPT}}(\sigma) + m - 1$. Here $H_m = \sum_{i=1}^{m} 1/i$ is the mth harmonic number. LQF is a special instance of the generic algorithm. Since we show that the cost of LQF is within $(3 + \lceil \log_2 m \rceil) \cdot L_{\mathrm{OPT}}(\sigma)$, our analysis is tighter if $L_{\mathrm{OPT}}(\sigma)$ is smaller than $2.5 \cdot (m/\ln m)$ (approximately). They also showed that another special instance of the generic algorithm has cost within $H_m \cdot L_{\mathrm{OPT}}(\sigma) + 1 + \log_2 m$ which is superior to our bound for LQF.

Chrobak et al. [9] studied a more general problem than BSP in the context of multiprocessor scheduling with conflicts. In their problem, multiple processors compete for resources on an undirected graph G (called a *conflict graph*) where the vertices correspond to the processors and the edges represent conflicts so that only processors forming an independent set of G can process their loads simultaneously. BSP corresponds to the case when G is the complete graph K_m. However, their paper only considers the continuous model.

BSP is also related to the on-line load balancing problem initiated by Graham [14]. 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 total load on the servers is minimized. Tasks arrive one by one, and each task comes with 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 studied [1]–[4], [6], [7], [14]. In the *identical machines model* [14], all the coordinates of a load vector are the same. This is also the case in the *restricted assignment model* [4], but there is the additional constraint that each task can only be handled by a subset of the servers. The natural greedy algorithm that always chooses the least loaded server is $(2 - 1/m)$-competitive in the identical machines model [14] and $\Theta(\log m)$-competitive in the restricted assignment model [4]. In the *temporal tasks model* [2], [3], tasks have a limited duration (which is unknown to the on-line algorithm until departure) and disappear after they are finished. In this model the greedy algorithm is only $\Theta(m^{2/3})$-competitive [2].

BSP differs from the traditional on-line load balancing problem in that the packets "appear" in the queues and the algorithm must balance the load (or queue length) by selecting departing packets. We obtain the upper bound for BSP by extending the technique for the restricted assignment model of the on-line load balancing problem [4], unlike Bar-Noy et al. [5] who reduce the discrete problem to a simulation of the continuous problem.

2. Problem Statement

2.1. *The Balanced Scheduling Problem.* The 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 σ at R. Initially at time 0, the m queues are empty. At each time $t > 0$, N_t packets, denoted by $(N_t^1, N_t^2, \ldots, N_t^m)$, 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 in the m queues such that N_t^i packets go into q_i, for $1 \leq i \leq m$. Let $l_A^i(t)$ denote the length of q_i at time t, after the N_t packets have arrived but before the scheduling algorithm A operating in R selects exactly one non-empty queue and outputs its head packet (if all the queues are empty nothing happens). We denote the length of q_i after this has happened by $l_A^i(t^a)$. Note that if $N_t > 1$, then the total number of packets in R after time t is larger than after time $t - 1$, and if $N_t = 0$, then the total number of packets after time t is smaller than after time $t - 1$. We assume that at least one queue is not empty until the end of the entire scheduling period. This assumption does not lose generality, because, if there is a time when all the queues are empty, then we can partition σ into multiple subsequences and solve BSP separately and independently on each subsequence.

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., we deal with $l_A^i(t)$, not with $l_A^i(t^a)$. We only speak about t^a in the analysis of the algorithms. The maximal queue length at time t incurred 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 let $|\sigma|$ be the time of the last arrival. Then the maximal queue length over the whole scheduling period for σ by A is defined as

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

Our aim in BSP is to find a scheduling algorithm A that minimizes $L_A(\sigma)$.

We denote 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 scheduling algorithm A, we clearly have $l_A(t) \le C(t)$. For the optimal off-line algorithm OPT, we have $l_{\mathrm{OPT}}(t) \ge C(t)/m$ because $C(t)$ packets are distributed among the m queues. Thus, $l_A(t) \le m \cdot l_{\mathrm{OPT}}(t)$ at any time t. This proves the following theorem.

THEOREM 1. *Any on-line algorithm for BSP is m-competitive* (*or better*).

2.2. *The Delay Balanced Scheduling Problem.* We must first define what we mean by "delay". Suppose that a packet p arrives at R at time t_1 and leaves R at time t_2. Then the *delay* d_p of p is $t_2 - t_1$. Let P_i be the set of all the packets assigned to q_i over the entire scheduling period. Then the *total delay* D_A^i of q_i incurred by A is defined as $\sum_{p\in P_i} d_p$. In DBSP we want to find a scheduling algorithm A that minimizes $\max_{1\le i\le m} D_A^i$.

When analyzing algorithms for DBSP, we assume without loss of generality that a packet p incurs a delay of 1 for each time unit while it is waiting in the queue, rather than it incurs a delay of d_p all at once at the end when it is released. Then the total delay of q_i up to time t by A, $D_A^i(t)$, is defined as follows:

$$\begin{aligned} D_A^i(1) &= 0. \\ D_A^i(t+1) &= \begin{cases} D_A^i(t) + l_A^i(t) - 1, & \text{if } q_i \text{ is selected by } A \text{ at time } t; \\ D_A^i(t) + l_A^i(t), & \text{if } q_i \text{ is not selected by } A \text{ at time } t. \end{cases} \end{aligned}$$

Let $D_A(t) = \max_{1\le i\le m} D_A^i(t)$. Obviously, $D_A^i(t) = D_A^i$ and $D_A(t) = \max_{1\le i\le m} D_A^i$ at the end of the scheduling period. We discuss DBSP in Section 6.

3. A Polynomial Time Optimal Off-Line Algorithm.

In this section we describe a polynomial time optimal off-line algorithm for BSP. For a constant $M \ge 1$, we call a schedule *M-feasible* if it selects output packets such that no queue ever has length more than M. We first describe an efficient algorithm named JUDGE_M that can find an M-feasible schedule for a given packet arrival sequence, or decides that no M-feasible schedule exists. Using this algorithm as a subroutine we can easily find the smallest possible value of M by binary search, thus solving the off-line BSP. If we start the search with M equal to N, the total number of all packets in the sequence, we can find the optimal value of M in $O(\log N)$ steps. Thus, the running time is polynomial in the size of the input.

At any time, JUDGE_M selects the non-empty queue that would be earliest in the future overflow (i.e., fill up with more than M packets) if we immediately stopped outputting any packets.

We call a queue q_i *active* at time t if $l^i_{\mathrm{JM}}(t) > 0$. For an active queue q_i and time $t' \geq t$, let $\bar{l}^i_{\mathrm{JM}}(t') = l^i_{\mathrm{JM}}(t) + \sum_{j=t+1}^{t'} N^i_j$ denote the length of q_i at time t' assuming that no packets are outputted between time t and t'. The selection rule at time t is as follows. If no queue is active, then no action is performed. Otherwise, let $t' \geq t$ be the earliest time such that $\bar{l}^i_{\mathrm{JM}}(t') > M$ for some active queue q_i, if such t' exists. Then JUDGE_M selects q_i, breaking ties arbitrarily. If t' does not exist, then JUDGE_M selects an arbitrary active queue.

THEOREM 2. *For any packet arrival sequence σ*, JUDGE_M *computes an M-feasible schedule if such a schedule exists for σ.*

PROOF. Consider a packet arrival sequence that has an M-feasible schedule. Fix such a schedule. It will produce queues identical to the queues produced by JUDGE_M, at least for a few initial time units before it selects an output packet from a different queue. Note that the queues will be identical initially and after the first arrival of packets. Let Opt_M denote an M-feasible schedule that maximizes the length of this initial identical subschedule.

Assume Opt_M differs from the schedule produced by JUDGE_M, and let $t > 0$ be the time when it differs first. Then $l^h_{\mathrm{opt\_M}}(t) = l^h_{\mathrm{JM}}(t)$, for $h = 1, \ldots, m$, and this is not true at time $t + 1$. Assume JUDGE_M selects queue q_i at time t and Opt_M selects queue q_j, where $i \neq j$. Clearly, q_i and q_j must be active at time t. Let $t'_i \geq t$ be the first time when q_i would overflow. If such t'_i does not exist, then let $t'_i = \infty$. Since Opt_M is M-feasible q_i did not overflow at time t, so we have $t'_i > t$.

Let $t'_{\mathrm{Opt\_M}} > t$ be the first time when Opt_M selects q_i. Since q_i would overflow at time t'_i, Opt_M must output a packet from q_i before time t'_i. Thus, $t < t'_{\mathrm{Opt\_M}} < t'_i$. By our choice of q_i at time t, no queue which is active at time t can accumulate more than M packets before time $t'_{\mathrm{Opt\_M}}$, independent of the scheduling strategy. Thus, we can exchange the two packets scheduled at times t and $t'_{\mathrm{Opt\_M}}$ in Opt_M, i.e., at time t select q_i and at time $t'_{\mathrm{Opt\_M}}$ select q_j. In this new schedule no queue will overflow before time $t'_{\mathrm{Opt\_M}}$, and afterwards the queues will again be identical to the queues in Opt_M. So this new schedule is also M-feasible. However, it coincides with JUDGE_M's schedule for one more time unit, contradicting our assumption that Opt_M maximizes the length of the initial identical subschedule with JUDGE_M.

Thus, we can assume that Opt_M is identical to JUDGE_M, i.e., JUDGE_M computes an M-feasible schedule. $\square$

As mentioned at the beginning of this section, we can discover the optimal off-line schedule by binary search, while invoking JUDGE_M as a subroutine.

THEOREM 3. *An optimal off-line schedule for BSP can be computed in time polynomial in the total number of packets.*

4. Lower Bounds.

This section investigates lower bounds on the competitiveness of on-line algorithms for BSP.

4.1. *Lower Bounds for General On-Line Algorithms*. First, we obtain lower bounds for general on-line algorithms. We exploit a technique similar to the lower bound proofs for the restricted assignment model of the on-line load balancing problem [4]. In the proof an adversary constructs a packet arrival sequence σ that annoys on-line algorithms.

4.1.1. *The Deterministic Lower Bound*

THEOREM 4. *No deterministic on-line algorithm for BSP can be better than* $(1 + \lfloor \log_2 m \rfloor)$*-competitive.*

PROOF. Let A be a deterministic on-line algorithm for BSP. Let j be the largest integer satisfying $2^j \leq m$, i.e., $j = \lfloor \log_2 m \rfloor$. Let $m' = 2^j$. The adversary constructs σ in $j+1$ phases. At the beginning of phase 1, exactly one packet for each of the first m' queues arrives. Phase 1 lasts for $m'/2$ time units, so A can only empty $m'/2$ queues. At the same time, the adversary empties the other $m'/2$ queues (since A is deterministic, the adversary knows in advance how A will react), and has one packet arrive for each of these queues at the beginning of phase 2. Of course, these queues now have length 2 in A's schedule. Phase 2 only lasts for $m'/4$ time units, so again A can shorten at most half of its long queues (of length 2). The adversary will again empty the other half, and so on. At the beginning of phase $j+1$, A has at least one queue of length j, and this queue will grow to length $j+1$ by the single packet arriving for this queue at the beginning of phase $j+1$.

Since the adversary can maintain queues of length at most 1 all the time, A is not better than $(j+1)$-competitive. □

4.1.2. *The Randomized Lower Bound*. We can also derive the randomized lower bound for BSP by applying the technique for the restricted assignment model of the on-line load balancing problem [4]. This bound holds against an oblivious adversary that must fix its request sequence right at the beginning, i.e., it cannot react to the random choices of the on-line algorithm (see [8]).

THEOREM 5. *No randomized on-line algorithm for BSP can be better than H_m-competitive against an oblivious adversary, where $H_m = \sum_{i=1}^{m} 1/i$, i.e., the m*th *harmonic number.*

PROOF. Let A be a randomized on-line algorithm for BSP. The adversary constructs σ in m phases, where the duration of phase k is $m-k$ time units for $k < m$ and 1 time unit for $k = m$. At the beginning of phase k, the adversary assigns exactly one new packet to each of the $m+1-k$ *expectedly* longest queues (ties are broken arbitrarily). Note that the adversary can obtain the $m+1-k$ expectedly longest queues from the description of A.

For phase k, $1 \leq k \leq m-1$, $\texttt{OPT}$ selects the $m-k$ queues to which a packet is assigned at the beginning of the next phase. Thus, as for $\texttt{OPT}$, the length of any queue does not go beyond 1, and we have $L_{\texttt{OPT}}(\sigma) = 1$.

As for A, by induction on k, we prove that the sum of the expected lengths of the $m+1-k$ longest queues just before phase k begins is at least $(m+1-k)(H_m - H_{m+1-k})$

for $k \geq 1$. For $k = 1$, the claim is trivial, because all the m queues are empty before the first packet arrives. Now suppose the claim is also true for some $k \geq 1$. Since $m - k$ packets leave R in phase k, the sum of the expected lengths of the $m + 1 - k$ longest queues (at the beginning of phase k) just before phase $k + 1$ begins is at least $(m + 1 - k)(H_m - H_{m+1-k}) + 1 = (m + 1 - k)(H_m - H_{m-k})$. Thus, the sum of the expected lengths of the $(m + 1 - (k + 1))$ longest queues just before phase $k + 1$ begins is at least $(m + 1 - k)(H_m - H_{m-k})((m - k)/(m + 1 - k)) = (m - k)(H_m - H_{m-k}) = (m + 1 - (k + 1))(H_m - H_{m+1-(k+1)})$, which completes the proof of the induction step. Hence just before the final phase m starts, the expected longest queue will grow to at least $H_m - H_1$. In phase m, one packet is assigned to this expected longest queue. Hence $E[L_A(\sigma)] \geq H_m$. □

Thus, no randomized on-line algorithm is better than $(\ln m)$-competitive against an oblivious adversary, because $\ln m < H_m \leq 1 + \ln m$.

4.2. *The Lower Bound for* `Round-Robin`. Next we examine the lower bound of the popular `Round-Robin` scheduling algorithm. Unfortunately, `Round-Robin` cannot exceed the trivial upper bound of m.

Algorithm `Round-Robin`**:** Initially, the algorithm may select any non-empty queue. If the algorithm selected queue q_i at time t, then the queue selected at time $t + 1$ is $q_{(r+i) \bmod m}$, where r is the smallest positive integer satisfying the condition that $q_{(r+i) \bmod m}$ is not empty.

THEOREM 6. `Round-Robin` *is not better than m-competitive.*

PROOF. Without loss of generality, we assume that `Round-Robin` initially selects q_1. Again, an adversary constructs a hard packet arrival sequence σ as follows:

- Step 1: At time 1, m packets arrive at R such that one packet is assigned to each q_i, for $1 \leq i \leq m$.
- Step 2: At time k, for $2 \leq k \leq m$, one new packet is assigned to q_m.

The optimal off-line algorithm `OPT` chooses q_m all the time. This assures that the length of q_m does not exceed 1. The lengths of all the other queues have constant length 1 all the time. Thus, $l_{\text{OPT}}(t) = 1$ at any time t, and $L_{\text{OPT}}(\sigma) = 1$.

On the other hand, `Round-Robin` selects q_k at time k, for $k \leq m$. Since the algorithm initially selects q_1, q_m is always the longest queue. As q_m is never selected before time m, $l^m_{\text{Round-Robin}}(m) = 1 + (m - 1) = m$. Thus, $L_{\text{Round-Robin}}(\sigma) = m = m \cdot L_{\text{OPT}}(\sigma)$. □

5. Upper Bounds. In this section we analyze the performance of a specific greedy algorithm, `LQF`. Since simple greedy policies are analyzed in many load-balancing problems [1], [2], [4], [14], measuring the performance of `LQF` enables us to estimate the relative difficulty of BSP to other problems.

Algorithm `LQF`**:** At any time, `LQF` selects the longest queue. Ties are broken arbitrarily.

THEOREM 7. `LQF` *is* $(3 + \lceil \log_2 m \rceil)$*-competitive.*

Theorem 7 together with Theorem 4 shows that `LQF` is a nearly optimal on-line algorithm. To prove the theorem, we extend the proof technique for the restricted assignment model of the on-line load balancing problem [4].

We introduce a function *gap* that maps the packets in `LQF`'s queues to some integer values. Let σ be an arbitrary packet arrival sequence. If p is the rth packet from the top (i.e., the output port) of queue q_i at time t, then the gap of packet p at time t, denoted by $gap(p, t)$, is defined as $r - l^i_{\mathrm{OPT}}(t)$. Intuitively, the function *gap* compares the height of a packet in a queue with the length of the corresponding queue in `OPT`.

According to the value of *gap*, we partition `LQF`'s queues into layers (see Figure 2(a)). Let $l = L_{\mathrm{OPT}}(\sigma)$. The kth layer of q_i at time t consists of packets p stored in q_i at that time such that $(k-1)l + 1 \le gap(p, t) \le kl$. The number of packets contained in the kth layer of q_i is denoted by $W^i_k(t)$.

LEMMA 8. *For* $k \ge 1$, *if* $W^i_{k+1}(t) > 0$, *then* $W^i_k(t) = l$.

PROOF. $W^i_{k+1}(t) > 0$ implies $l^i_{\mathrm{LQF}}(t) > l^i_{\mathrm{OPT}}(t) + kl$. Hence, the number of packets in `LQF`'s q_i whose gaps are greater than or equal to $(k-1)l + 1$ but less than or equal to kl is exactly l. □

COROLLARY 9. *For* $k \ge 1$, $W^i_k(t) \ge W^i_{k+1}(t)$. □

We also use the following notations:

- $R^i_k(t) = \sum_{j>k} W^i_j(t)$;

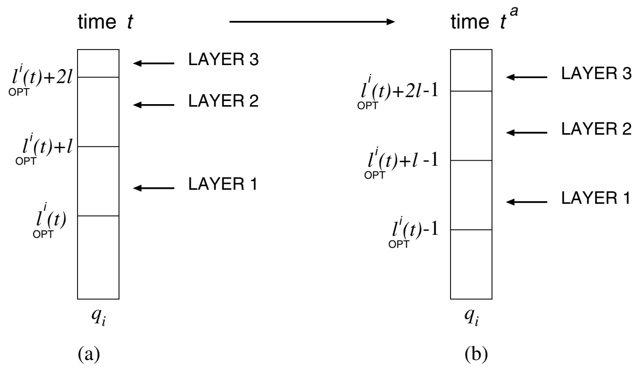
time t
time t^a
LAYER 3
LAYER 2
LAYER 1
$l^i_{\mathrm{OPT}}(t)+2l$
$l^i_{\mathrm{OPT}}(t)+l$
$l^i_{\mathrm{OPT}}(t)$
$l^i_{\mathrm{OPT}}(t)+2l-1$
$l^i_{\mathrm{OPT}}(t)+l-1$
$l^i_{\mathrm{OPT}}(t)-1$
q_i
q_i
(a)
(b)

Fig. 2. Partition of a queue into layers.

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

$R_k^i(t)$ is the total number of packets in q_i contained in the layers strictly higher than the kth layer. $W_k(t)$ is the total number of packets contained in the kth layer over all the m queues, and $R_k(t)$ is the total number of packets contained in the layers strictly higher than the kth layer over all the m queues. Thus, for $k \geq 1$,

$$R_{k+1}(t) = R_k(t) - W_{k+1}(t). \tag{2}$$

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 q_i is not affected since the same number of packets arrive at q_i at the beginning of time $t+1$ both in OPT and in LQF. By contrast $W_k(t^a)$ (or $R_k(t^a)$) may be different from $W_k(t)$ (or $R_k(t)$), depending on the queues selected by the two algorithms at time t.

PROOF OF THEOREM 7. Our strategy is to run the two algorithms LQF and OPT simultaneously on a given packet arrival sequence σ and to prove that the following inequality is maintained all the time for $k \geq 1$:

$$W_k(t) \geq R_k(t). \tag{3}$$

In this case we conclude from (2) that $R_{k+1}(t) = R_k(t) - W_{k+1}(t) \leq R_k(t) - R_{k+1}(t)$, and thus

$$R_{k+1}(t) \leq \tfrac{1}{2} R_k(t) .$$

Then, by applying this inequality $\lceil \log_2 m \rceil$ times, we derive the following inequality. Since $L_{\mathrm{OPT}}(\sigma) = l$, we have $R_1(t) \leq ml$, and thus

$$R_{\lceil \log_2 m \rceil + 1}(t) \leq (\tfrac{1}{2})^{\lceil \log_2 m \rceil} R_1(t) \leq (\tfrac{1}{2})^{\log_2 m} R_1(t) = \frac{1}{m} R_1(t) \leq \frac{ml}{m} = l .$$

So the number of packets in the layers strictly above the $(\lceil \log_2 m \rceil + 1)$th layer is at most l. As a result, the length of the longest queue of LQF at time t is bounded from above by

$$l_{\mathrm{LQF}}(t) \leq l_{\mathrm{OPT}}(t) + (\lceil \log_2 m \rceil + 1) \cdot l + l \leq (3 + \lceil \log_2 m \rceil) \cdot l . \tag{4}$$

Since (4) holds at any time t, the proof of Theorem 7 is complete.

It remains to show (3) for $k \geq 1$ and for any t. Let $k \geq 1$ be fixed. The proof is by induction on the time t. As for the base case, (3) is true at time $t = 1$ before packets are outputted because all the m queues have the same number of packets both in OPT and in LQF.

Now suppose that $W_k(t) \geq R_k(t)$ at time $t \geq 1$ before the two algorithms select the queue from which to output a packet. 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 OPT selects q_i and LQF selects q_j at time t. If $i = j$, then clearly $W_k(t^a) = W_k(t) \geq R_k(t) = R_k(t^a)$. So we assume that $i \neq j$. Since $W_k^h(t) = W_k^h(t^a)$ and $R_k^h(t) = R_k^h(t^a)$ for any queue q_h except q_i and q_j, we can focus on how q_i and q_j change.

First we consider q_i. If $l^i_{\mathrm{LQF}}(t) < l^i_{\mathrm{OPT}}(t)$, then $l^i_{\mathrm{LQF}}(t^a) \leq l^i_{\mathrm{OPT}}(t^a)$ and therefore

$$W_k^i(t) = W_k^i(t^a) = 0 \quad \text{and} \quad R_k^i(t) = R_k^i(t^a) = 0. \tag{5}$$

If $l^i_{\mathrm{LQF}}(t) \geq l^i_{\mathrm{OPT}}(t)$, then let $Y = \lfloor (l^i_{\mathrm{LQF}}(t) - l^i_{\mathrm{OPT}}(t))/l \rfloor + 1$. See Figure 2(b). From the definitions of $W_k^i(t)$ and $R_k^i(t)$ we obtain

$$W_k^i(t^a) = \begin{cases} W_k^i(t), & \text{if } \quad k \neq Y, \\ W_k^i(t) + 1, & \text{if } \quad k = Y, \end{cases} \tag{6}$$

and

$$R_k^i(t^a) = \begin{cases} R_k^i(t) + 1, & \text{if } \quad k < Y, \\ R_k^i(t), & \text{if } \quad k \geq Y. \end{cases} \tag{7}$$

Next we consider q_j. If $l^j_{\mathrm{LQF}}(t) \leq l^j_{\mathrm{OPT}}(t)$, then $l^j_{\mathrm{LQF}}(t^a) < l^j_{\mathrm{OPT}}(t^a)$ and therefore

$$W_k^j(t) = W_k^j(t^a) = 0 \quad \text{and} \quad R_k^j(t) = R_k^j(t^a) = 0. \tag{8}$$

If $l^j_{\mathrm{LQF}}(t) > l^j_{\mathrm{OPT}}(t)$, then let $X = \lceil (l^j_{\mathrm{LQF}}(t) - l^j_{\mathrm{OPT}}(t))/l \rceil$. From the definitions of $W_k^j(t)$ and $R_k^j(t)$ we obtain

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

and

$$R_k^j(t^a) = \begin{cases} R_k^j(t) - 1, & \text{if } \quad k < X, \\ R_k^j(t), & \text{if } \quad k \geq X. \end{cases} \tag{10}$$

Equations (5)–(10) and the assumption that $W_k(t) \geq R_k(t)$ at time t imply that at least one of the following two conditions would have to be satisfied to invalidate the Inequality (3) at time t^a:

- Type I: $W_k^j(t^a) = W_k^j(t) - 1$.
- Type II: $R_k^i(t^a) = R_k^i(t) + 1$.

We will now show that (3) is still true in both cases.

Type I: *Suppose that* $W_k^j(t^a) = W_k^j(t) - 1$. From (9), $l^j_{\mathrm{LQF}}(t) > l^j_{\mathrm{OPT}}(t)$ and k must be equal to X. Hence $l^j_{\mathrm{LQF}}(t) \leq l^j_{\mathrm{OPT}}(t) + lX$. We show the $(X+2)$th layers of all the m queues are empty at t^a by contradiction. Since $l^j_{\mathrm{LQF}}(t) \leq l^j_{\mathrm{OPT}}(t) + lX$, $l^j_{\mathrm{LQF}}(t^a) = l^j_{\mathrm{LQF}}(t) - 1 \leq l^j_{\mathrm{OPT}}(t^a) + lX$ 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 LQF at t, we have

$$\begin{aligned} l^h_{\mathrm{LQF}}(t) = l^h_{\mathrm{LQF}}(t^a) &\geq (X+1)l + 1 \qquad \text{(since } q_h\text{'s } (X+2)\text{th layer is not empty)} \\ &> l^j_{\mathrm{OPT}}(t) + lX \geq l^j_{\mathrm{LQF}}(t). \end{aligned}$$

This contradicts the fact that q_j is selected by LQF at time t. Thus the $(X+2)$th layers of all the m queues must be empty at time t^a. However, then, by Corollary 9, $W_X(t^a) \geq W_{X+1}(t^a) = R_X(t^a)$, which shows that inequality (3) holds in this case.

Type II: *Suppose that* $R_k^i(t^a) = R_k^i(t) + 1$. Since $R_k^i(t^a) \geq 1$, we have $l_{\mathrm{LQF}}^i(t) = l_{\mathrm{LQF}}^i(t^a) \geq kl + 1$. Since LQF does not select q_i but selects q_j, we have

$$l_{\mathrm{LQF}}^j(t) \geq l_{\mathrm{LQF}}^i(t) \geq kl + 1 \geq l_{\mathrm{OPT}}^j(t) + (k-1)l + 1.$$

If $l_{\mathrm{OPT}}^j(t) + (k-1)l + 1 \leq l_{\mathrm{LQF}}^j(t) \leq l_{\mathrm{OPT}}^j(t) + kl$, we can show that the $(k+2)$th layers of all the m queues are empty at time t^a exactly in the same way as in the previous case. Thus, from Corollary 9, $W_k(t^a) \geq W_{k+1}(t^a) = R_k(t^a)$.

By contrast, if $l_{\mathrm{LQF}}^j(t) > l_{\mathrm{OPT}}^j(t) + kl$, we have $R_k^j(t) > 0$. Hence, $R_k^j(t^a) = R_k^j(t) - 1$ after LQF outputs a packet at time t from q_j. By comparing this with (8) and (10), we have $l_{\mathrm{LQF}}^j(t) > l_{\mathrm{OPT}}^j(t)$ and $k < X$. Thus,

$$\begin{aligned} (11) \qquad R_k(t^a) &= R_k^i(t^a) + R_k^j(t^a) + \sum_{h \neq i,j} R_k^h(t^a) \\ &= (R_k^i(t) + 1) + (R_k^j(t) - 1) + \sum_{h \neq i,j} R_k^h(t) = R_k(t). \end{aligned}$$

Regarding $W_k(t)$, it follows that $W_k^i(t^a) \geq W_k^i(t)$ from (6) and that $W_k^j(t^a) = W_k^j(t)$ from (9) as $k \neq X$. Hence,

$$(12) \qquad W_k(t^a) \geq W_k(t).$$

From (11) and (12), it follows that $W_k(t^a) \geq W_k(t) \geq R_k(t) = R_k(t^a)$. Thus, we have proved (3) for all possible cases. This finishes the proof of Theorem 7. □

6. The Lower Bound for DBSP. Another important performance measure for QoS networks is the delay of packets. The aim of DBSP is to balance the total delay per queue. Here the total delay of a queue q_i is defined as the sum of the delays of all the packets assigned to q_i. In this section we study a deterministic lower bound on the competitiveness of algorithms for DBSP. Interestingly, DBSP contains BSP as a subproblem and the lower bound for BSP in Theorem 4 immediately gives a lower bound for DBSP.

The next lemma compares a deterministic on-line algorithm with an off-line algorithm Off that is not necessarily optimal.

LEMMA 10. *Consider an arbitrary deterministic on-line algorithm A for DBSP. Suppose that there exists a time t such that $l_A(t) = X$ and $l_{\mathrm{Off}}(t) = Y$, where* Off *is a certain off-line algorithm in whose execution there are at least two non-empty queues at time t before* Off *selects a queue. Then A is not better than $(X-1)/Y$-competitive.*

PROOF. Let q_i be the longest of A's queues and let q_j be the longest of Off's queues at time t. If there are several possible choices for q_j, then we choose q_j such that $D_{\mathrm{Off}}^j(t)$ is not smaller than it would be for any other candidate. Since $l_A^i(t) = X$, we have

$l^i_A(t^a) \geq X - 1$. On the other hand, since $l^j_{\text{Off}}(t) = Y$, we have $l^j_{\text{Off}}(t^a) = Y$ provided that `Off` does not select q_j at time t.

Suppose that exactly one packet is assigned to q_i per time unit and that no packet is assigned to the rest of the queues after time t. Since no scheduling algorithm can output more than one packet at each time, the length of q_i in A is always greater than $X-1$ after time t. On the other hand, `Off` keeps on outputting the packets that have just arrived and keeps the length of the longest queue q_j at Y after time t. Hence for any $t' > t$, we have

$$D_A(t') \geq D^i_A(t') \geq D^i_A(t) + (X-1)(t'-t). \tag{13}$$

As for `Off`, q_j always becomes the longest queue after time t. Moreover, $D^j_{\text{Off}}(t)$ is larger than any other queues that have the same length as q_j at time t. Hence, for sufficiently large values of $t' > t$, it follows that

$$D_{\text{Off}}(t') = D^j_{\text{Off}}(t) + Y(t'-t). \tag{14}$$

The lower bound on the competitiveness for DBSP is obtained by dividing $D_A(t')$ by $D_{\text{Off}}(t')$: $D_A(t')/D_{\text{Off}}(t') \geq (D^i_A(t) + (X-1)(t'-t))/(D^j_{\text{Off}}(t) + Y(t'-t))$. This expression approaches $(X-1)/Y$ as t' approaches ∞. □

Using the hard request sequence σ for BSP in Section 4 for DBSP, we get the following lower bound for DBSP.

THEOREM 11. *No deterministic on-line algorithm for DBSP can be better than* $\lfloor \log_2 m \rfloor$*-competitive.*

PROOF. We use the same notations as in the proof of Theorem 4. Let A be an on-line algorithm for DBSP. Let $j = \lfloor \log_2 m \rfloor$ and let σ be the hard request sequence for BSP in the proof of Theorem 4. Then the same discussion as in that proof yields $l_{\text{OPT}}(T_{j+1}) = 1$ and $l_A(T_{j+1}) = 1 + \lfloor \log_2 m \rfloor$ at the start time T_{j+1} of phase $(j+1)$. Let `OPT` be the optimal off-line algorithm for BSP in the proof of Theorem 4. By using `OPT` as the off-line algorithm `Off` in Lemma 10, we can prove that no deterministic on-line algorithm is better than $((1 + \lfloor \log_2 m \rfloor) - 1)/1 = \lfloor \log_2 m \rfloor$-competitive for DBSP. □

7. Conclusions.

This paper investigates BSP to evaluate the power of scheduling algorithms in a router in terms of prevention of packet losses. We prove that a simple greedy algorithm is $\Theta(\log m)$-competitive and nearly optimal, where m is the number of queues.

There are many open problems with regard to BSP. One interesting open problem is to design an optimal off-line algorithm that finds the optimal solution directly without relying on binary search.

The following problems might also be worthwhile studying because they are more applied:

- Extending BSP to a dynamic buffer allocation policy. In this model a traffic stream with higher priority can also use the buffer memories provided for the streams with lower

priority. In this model BSP must be combined with the hierarchical server topology like in [6].
- Changing the amount of buffers assigned to each queue. In practical QoS networks it is common that each traffic is given a 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 provided for it.
- Discovering a competitive on-line algorithm for DBSP.

Acknowledgment. Helpful comments from Susanne Albers on an early version of this paper are greatly appreciated.

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(4):486–504, 1997.

[2] Y. Azar, A.Z. Broder, and A.R. Karlin. On-line load balancing. *Theoretical Computer Science*, 130(1):73–84, 1994.

[3] Y. Azar, B. Kalyanasundaram, S. Plotkin, K.R. Pruhs, and O. Waarts. On-line load balancing of temporary tasks. *Journal of Algorithms*, 22(1):93–110, 1997.

[4] Y. Azar, J. Naor, and R. Rom. The competitiveness of on-line assignments. *Journal of Algorithms*, 18(2):221–237, 1995.

[5] A. Bar-Noy, A. Freund, S. Landa, and J. Naor. Competitive on-line switching policies. In *Proceedings of the 13th ACM–SIAM Symposium on Discrete Algorithms*, pages 525–534, 2002.

[6] A. Bar-Noy, A. Freund, and J. Naor. On-line load balancing in a hierarchical server topology. In *Proceedings of the 7th Annual European Symposium on Algorithm*, pages 77–88. Lecture Notes in Computer Science 1643. Springer-Verlag, Berlin, 1999.

[7] P. Berman, M. Charikar, and M. Karpinski. On-line load balancing for related machines. *Journal of Algorithms*, 35(1):108–121, 2000.

[8] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, Cambridge, 1998.

[9] M. Chrobak, J. Csirik, C. Imreh, J. Noga, J. Sgall, and G.J. Woeginger. The buffer minimization problem for multiprocessor scheduling with conflicts. In *Proceedings of the 28th International Colloquium on Automata, Languages and Programming*, pages 862–874. Lecture Notes in Computer Science 2076. Springer-Verlag, Berlin, 2001.

[10] C. Dovrolis, D. Stiladis, and P. Ramanathan. Proportional differentiated services: delay differentiation and packet scheduling. In *Proceedings of ACM SIGCOMM* '99, pages 109–120, 1999.

[11] A. Fiat and G. Woeginger, editors. *Online Algorithms—The State of the Art*. Lecture Notes in Computer Science 1442. Springer-Verlag, Berlin, 1998.

[12] A. Fiat and G. Woeginger. On-line scheduling on a single machine: minimizing the total completion time. *Acta Informatica*, 36(4):287–293, 1999.

[13] A. Goel, A. Meyerson, and S. Plotkin. Distributed admission control, scheduling, and routing with stale information. In *Proceedings of the 12th ACM–SIAM Symposium on Discrete Algorithms*, pages 611–619, 2001.

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

[15] H. Hoogeveen and A. Vestjens. Optimal on-line algorithms for single-machine scheduling. In *Proceedings of the 5th International IPCO Conference*, pages 404–414. Lecture Notes in Computer Science 1084. Springer-Verlag, Berlin, 1996.

[16] 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.

[17] S. Jamin, P.B. Danzig, S. Shenker, and L. Zhang. A measurement-based admission control algorithm for integrated services packet networks. In *Proceedings of ACM SIGCOMM* '95, pages 2–13, 1995.