On Queuing Lengths in On-Line Switching

Peter Damaschke*
School of Computer Science and Engineering
Chalmers University, 41296 Göteborg, Sweden
ptr@cs.chalmers.se

Zhen Zhou†
Department of Computer Science
Hong Kong University of Science and Technology, Hong Kong
cszz@cs.ust.hk

Abstract

Queues that temporarily store fixed-length packets are ubiquitous in network switches. Scheduling algorithms that prevent packet-loss are always desirable. LONGEST-QUEUE-FIRST (LQF) is an on-line greedy algorithm widely exploited because of its simplicity and efficiency. In this paper, we give improved bounds on the competitive ratio of LQF in terms of the worst-case queuing length, parameterized with respect to the optimal queuing length of a clairvoyant adversary. This gives a better picture of LQF's performance under heavy traffic than the usual (unparameterized) competitive ratio. We also discuss randomization, and we conclude with some intriguing open problems regarding a two-dimensional generalization of the problem.

Keywords: online switching, queuing, parameterized competitive ratio

*Corresponding author.
†This work has been done while the second author was visiting Chalmers.

1 Introduction

As the network expands, switching technology continues to be one of the bottlenecks in the development. Multiple packets arriving at the same time from different input ports may be destined to the same output port and cause contention. There are several major queuing technologies resolving this situation. Switches based on Input Queuing (IQ) are desirable for high speed switching, but their throughput is limited to 58.6% if a FIFO policy is used, due to the head-of-line (HOL) blocking phenomena [7, 9]. Output Queuing (OQ) switches have optimal delay-throughput performance for all traffic distributions, but the N-times speed-up in the fabric limits the scalability of this architecture [6]. Virtual Output Queuing (VOQ) [1] is used to overcome the drawbacks and combine the advantages of an IQ switch and an OQ switch. In a VOQ switch (see Figure 1), each input maintains n queues, one for each output, so that no additional speedup is required and HOL blocking can be eliminated. See also [8] for a statistical analysis of switches.

Scheduling algorithms are used to minimize packet loss, maximize throughput and minimize delay of VOQ switches. In the present paper we concentrate on one parameter: queue length. Assume that time is slotted. In each round, there are a number of fixed-length packets coming from each of the input ports. Then the scheduler selects one packet for each output port, and the remaining packets wait on the queues associated with their destined output ports at the input ports. The goal of the scheduler is to keep the length of *every* queue small all the time, i.e., to minimize the maximum of all queue lengths ever occurring, as this maximum length determines the size of memory that must be allocated to each port in order to avoid packet loss. As the scheduler is not aware of the future input sequence, it is natural to study this problem in the framework of on-line algorithms. We assume that the reader is familiar with the basic notions of competitive analysis of on-line algorithms as in [3].

We denote the number of input ports and output ports by m and n, respectively. A malicious off-line adversary will construct and also serve an input-packet sequence σ. We denote by $L = ADV(\sigma)$ the maximal queue length for the adversary after serving the sequence, and $X = ALG(\sigma)$ be that of an on-line algorithm ALG. The competitive ratio of ALG is defined to be $\mathrm{CR} = \max_\sigma \frac{X}{L}$. Informally speaking, the on-line player/adversary tries to minimize/maximize ratio $\frac{X}{L}$. Note that in the present problem, unlike most of the other usual on-line problems, the goal is to minimize the all-time high of some quantity, rather than the accumulated costs of actions.

From now on we consider the simplified case of one output port only ($n = 1$). We call this problem the on-line Balanced Scheduling Problem (BSP), following the terminology in [5]. Longest-Queue-First (LQF) is one of the most natural algorithms for BSP. LQF always sends a packet from one of the currently longest queues to the output port, ties are broken arbitrarily. Thus LQF is easy to implement (although there are problems to implement it in hardware) and guarantees its running time proportional to the size of the switch, in every round. LQF also does a good job in preventing packet loss, due to the results in [5]. In our paper we give more evidence that LQF is an excellent strategy.

Outline of our contributions: In [5] they proved that LQF has CR $= \log_2 m + O(1)$, and showed that this bound is tight for deterministic strategies for BSP. Their upper bound proof is long and quite technical. Here, we develop an alternative proof which is conceptually much simpler. More importantly, it provides improved competitive ratios: The result of [5] is tight regarding the overall competitive ratio, but the picture changes if we stratify the set of possible input sequences with respect to L. That is, for any fixed L let us consider those input sequences where the adversary has a strategy that guarantees her a maximum queue length L (but not $L-1$). Actually, our new upper bound shown in Section 2 is $(1+\frac{1}{L})\ln m + O(1) = (1+\frac{1}{L})\ln 2 \cdot \log_2 m + O(1)$. Since $\ln 2 \approx 0.7$, our analysis of LQF beats the general upper bound of [5] already for $L \geq 3$, and by up to 30 percent for large L. This is practically interesting in cases with heavy and irregular network traffic. (Remember that the maximum queue length determines how much buffer space must be allocated for loss-free queuing.) Competitive analysis parameterized with the optimal queue length has already been done in [2], however their upper bound is, in our notation, $(1+\frac{1}{\ln 2 \cdot L})\ln m$ for a certain variant of LQF suggested by a continuous version of BSP (where fractional packets can be removed, but at most a total of 1). Hence our result is a bit sharper for every $L \geq 2$ and large enough m, and it holds for generic LQF. We conjecture that the true upper bound is even better.

In Section 3 we consider lower bounds for deterministic BSP algorithms. The lower bound of $\log_2 m$ in [5] has been established for the case $L = 1$. Due to our upper bounds for LQF parameterized by L, it can no longer hold for large L. It is quite easy to prove an asymptotically tight lower bound of roughly $\ln m$ for any fixed L. (This has been observed also in [2], but for the sake of completeness we include a proof.) By some more technical efforts we can raise it to essentially $(1 + 1/2L - O(1/L^2))\ln m$, thus giving an explicit lower bound for each L. It would be nice to close the remaining gap, especially for small L. In Section 4 we argue that no randomized algorithm can be better than the best deterministic algorithm, as a consequence of the peculiar nature of BSP.

Early analysis of the performance of VOQ switches rely very much on queuing theory and the statistics of traffic models. There are a considerable number of literature concerning this topic, a particular example is [10]. Section 5 is devoted to some open problems around LQF in the VOQ model: We propose some greedy scheduling algorithms, and we conjecture that some can achieve competitive ratio $O(\log(mn))$, but we must leave it an open question. We briefly discuss the difficulty of applying our proof technique to this practically interesting case with n output ports.

We remark that a lemma in [4] deals with a game where, alternatingly, an adversary puts arbitrary fractions of one packet on m queues, and the player *empties* one queue of her choice. The player's goal is to minimize the maximal length ever occurring, and the length bound turns out to be $\ln m$. However these seem to be only superficial similarities to the BSP problem. We are interested in the competitive ratio rather than absolute lengths, and the player in BSP is much weaker.

2 Upper Bounds for LQF

We start proving upper bounds parameterized by L. Let H_k be the k-th Harmonic number, that is, $H_k = \sum_{i=1}^{k} 1/i$.

Define y_k inductively by $y_1 = 0$, and $y_{k+1} = y_k + \frac{L+1}{k+1}$ for $k > 0$. We could also write $y_k = (L+1)(H_k - 1)$, but the first form will be slightly more convenient. Note that the y_k are in general fractional numbers, however our argument below does not rely on integer thresholds.

For a set Q_k of k queues, let $a(Q_k)$ and $b(Q_k)$ be the total number of packets in the adversary's queues and LQF's queues, respectively, in Q_k, and $d(Q_k) := b(Q_k) + kL - a(Q_k)$. Recall that X was the largest queue length of LQF, given a fixed input sequence. In the following, a *configuration* means the pair of m-tuples of queue lengths for both players at some moment. With these denotations we prove:

Lemma 1 *Provided that $X - y_m > L$, there exists, for every $k \leq m$, a configuration containing a set Q_k of k queues with $d(Q_k) \geq k(X - y_k)$.*

Proof. We apply induction on k. The induction hypothesis is true for $k = 1$: By the definition of X, some queue reaches length X. For the set Q_1 consisting of this queue we have that $d(Q_1) \geq X + L - a(Q_1) \geq X = 1(X - 0)$.

Assume the lemma holds for some $k \leq m - 1$. Before the game begun, all queue lengths were 0, hence $d(Q_k) = kL < k(X - y_m) < k(X - y_k)$. Consider the last round before the configuration for k was reached the first time, i.e., where still $d(Q_k) < k(X - y_k)$ after both players' actions. In the following, symbols $a(Q_k)$, $b(Q_k)$, and $d(Q_k)$ refer to the beginning of this round.

The next packet arrival does not promote $d(Q_k)$ yet, since the same number c of packets is added to both the adversary's queues and LQF's queues in Q_k. In order to raise $d(Q_k)$ to the threshold $k(X - y_k)$, the adversary must now remove one packet from Q_k while LQF does not. By the rule of LQF, some queue outside Q_k, in the following called the *supporting queue*, must be one of the longest queues at this moment, i.e., just before LQF reduces it. Since a maximum of numbers is at least their average, the supporting queue has length at least

$$\frac{b(Q_k)}{k} + \frac{c}{k} - 1 = \frac{d(Q_k)}{k} + \frac{a(Q_k)}{k} - L + \frac{c}{k} - 1$$

after reduction. Since $a(Q_k) \geq 0$ and $c \geq 1$, the latter expression is at least

$$\alpha := \frac{d(Q_k)}{k} - L + \frac{1}{k} - 1.$$

We define Q_{k+1} as Q_k plus the supporting queue at this moment. By definition of l we have

$$d(Q_{k+1}) = b(Q_{k+1}) + (k+1)L - a(Q_{k+1}).$$

The changes to the queue lengths discussed in the previous paragraph yield

$$d(Q_{k+1}) \geq (b(Q_k) + c + \alpha) + (k+1)L - (a(Q_k) + c - 1 + L).$$

Applying the definition of l again, we get

$$d(Q_{k+1}) \geq d(Q_k) + 1 + \alpha.$$

Since symbol $d(Q_k)$ refers to the previous round, the induction hypothesis must be rephrased as

$$d(Q_k) \geq k(X - y_k) - 1.$$

Altogether this yields:

$$d(Q_{k+1}) \geq d(Q_k) + 1 + \frac{d(Q_k)}{k} - L + \frac{1}{k} - 1 \geq (k+1)(X - y_k) - L - 1,$$

hence

$$d(Q_{k+1}) \geq (k+1)\left(X - y_k - \frac{L+1}{k+1}\right).$$

This completes the induction step. ◇

Theorem 2 *The competitive ratio CR of LQF is less than*

$$\left(1 + \frac{1}{L}\right)\left(\ln m + C + \frac{1}{2m}\right),$$

where $C \approx 0.577$ is the Euler-Mascheroni constant.

Proof. Assume $X \leq L + y_m$. Then

$$X \leq L + y_m \leq L + (L+1)\sum_{k=2}^{m} \frac{1}{k} < (L+1)\left(\ln m + C + \frac{1}{2m}\right),$$

and division by L gives CR.

Assume the other case, $X > L + y_m$. Applying Lemma 1 to $k = m$, there is a configuration with $d(Q_m) \geq m(X - y_m)$. On the other hand, $d(Q_m) = b(Q_m) + mL - a(Q_m) = mL$, since LQF and the adversary have always the same total number of packets. This yields $m(X - y_m) \leq mL$, a contradiction. ◇

3 Lower Bounds

Not surprisingly, our general lower bound proof has basically the same idea as in [5], however the details differ in that we change the number of queues decrementally.

Theorem 3 *For any fixed L, no deterministic algorithm for BSP can achieve a competitive ratio below $\ln m$.*

Proof. Assume we have fixed some online algorithm. Define $x_m = 1$ and $x_{k-1} = x_k + 1/k$ for $2 \leq k \leq m$. We show by downwards induction on $k \leq m$ the existence of a configuration with k queues of total length at least kLx_k.

Therefore, x_1 is the lower bound of CR. This claim is obvious for $k = m$, since the adversary can immediately put L packets on each queue.

Suppose the claim is true for k. Let Q_k be a set of player's queues of total length at least kLx_k. The adversary does not deliver new packets and empties $k-1$ of her queues in Q_k, which takes at most $(k-1)L$ rounds. After this time, the sum of lengths in Q_k is still at least $kLx_k - (k-1)L = kL(x_k - 1 + 1/k)$. It follows that the $k-1$ longest queues in Q_k now have a total length of at least $(k-1)L(x_k - 1 + 1/k)$. Since the on-line player's strategy is deterministic, the adversary knows in advance which $k-1$ queues in Q_k will be the longest, so that she can empty her corresponding queues. Finally she adds a total of $(k-1)L$ packets to them. This yields $k-1$ queues of total length at least $(k-1)L(x_k + 1/k) = (k-1)Lx_{k-1}$.

Finally apply the claim to $k = 1$ and note that $x_1 \approx \ln m$. $\diamond$

The simplicity of this proof is also its weakness. The argument does not exploit the fact that queue lengths are integer. A combination with the technique applied in [5] to $L = 1$ will give a sharper lower bound. First we need an elementary lemma.

Lemma 4 *In any partition of the positive integer s (or larger) into k nonnegative integer summands, the h largest summands, where $h \leq k$, have the sum*

$$t \geq (s - s \bmod k)h/k + \min(h, s \bmod k).$$

Proof. This is obvious for the *equalized partition* where s is the exact sum, $s \bmod k$ summands are equal, and the remaining $k - s \bmod k$ summands are smaller by 1. Any other partition can be transformed into the equalized partition by successively subtracting 1 from the largest summand and adding 1 to the smallest. Since t can only decrease in each step, the lemma follows. $\diamond$

The formulation of the improved lower bound is somewhat complicated by the technicalities of rounding.

Theorem 5 *For any fixed L and $\epsilon > 0$, there is an m_0 such that, for any $m > m_0$, no deterministic algorithm for BSP can achieve a competitive ratio below $\frac{1}{L\ln(1+1/L+\epsilon)} \ln m$.*

Proof. In the beginning, the adversary can produce a configuration with m queues, each of length L. In general, suppose that there exists a player's configuration with a set Q of k queues, each of length at least L. In each queue in Q, we only consider the top L packets that have to be removed first. Let Q' be the set of these pruned queues of length L in Q.

The adversary does not deliver new packets and empties $h < k$ of her queues in Q', which takes at most hL rounds. After this time, the sum of lengths in Q' is still at least $s := kL - hL$. By Lemma 4, the h longest queues in Q' have total length at least $t = (s - s \bmod k)h/k + \min(h, s \bmod k)$. Since the on-line player's strategy is deterministic, the adversary knows in advance which h queues in Q' will be the longest, so that she can empty her corresponding queues. Finally she adds a total of hL packets to them. Since these h queues in Q' have had hL packets before, the net increase of their total length is at least t.

The adversary chooses $h := \lfloor kL/(L+1) \rfloor$. For a moment suppose that $kL/(L+1)$ is integer, so that $s = kL - hL = h$. Then we have $s \bmod k = h \bmod k = h$, hence $t = h$. That is, the average length of these queues increased by 1. The conclusion still holds for $h < kL/(L+1)$, since the adversary has to promote a smaller number h of queues.

Iterating this procedure, the adversary obtains a sequence of sets of queues, such that the number of queues therein decreases by a factor not larger than $h/k = L/(L+1)$. At the same time, the average lengths increase by summand 1. The number of iterations would be $\frac{\ln m}{\ln(1+1/L)}$ if the factor were exactly $L/(L+1)$ in every step. Due to the rounding it is smaller, but the error remains limited by ϵ (arbitrarily small but fixed) as long as k is large enough compared to L, and the number of remaining iterations depends on the fixed L only. This eventually creates a queue of length $\frac{1}{\ln(1+1/L+\epsilon)} \ln m$, giving a competitive ratio of at least $\frac{1}{L\ln(1+1/L+\epsilon)} \ln m$. $\diamond$

Using the first three terms of the Taylor series of ln, one can easily give this lower bound the more comprehensive form $(1 + 1/2L - O(1/L^2)) \ln m$. The following table displays our lower and upper bounds on the factors of $\ln m$ for the smallest L's.

L	1	2	3	4	5
L.B.	1.43	1.22	1.16	1.11	1.08
U.B.	1.43	1.43	1.33	1.25	1.20

We conjecture that the true upper bounds for LQF are slightly better than we could prove in Section 2. The obstacle for an improvement is that the (basic) supporting queue argument implies only the existence of one more sufficiently long queue in each step. We must leave the complete solution as an open problem.

4 Randomization is Useless (Asymptotically)

Another result in [5] was an $\ln m$ lower bound for randomized strategies against an oblivious adversary (who must fix the request sequence in advance, without the ability to inspect the player's random bits). The construction worked for $L = 1$. However no randomized algorithm that beats the deterministic bound has been presented, thus it was not even clear whether the true randomized lower bound could be $\log_2 m$. In this section we argue that, in fact, randomized algorithms cannot be better than the best deterministic algorithm.

In the following, let $dcr(m, L)$ be a lower bound on the competitive ratio of deterministic algorithms for fixed m and L. We also need that the adversary can achieve this lower bound within a number $t(m, L)$ of rounds that depends on m and L only. In simpler words, the adversary can produce a queue of length

$dcr(m,L)L$ within time $t(m,L)$, whatever the deterministic algorithm applied by the online player is. This condition holds for the deterministic lower bounds in [5] and the present paper.

Theorem 6 *No randomized algorithm for BSP can achieve a competitive ratio below $dcr(m,L)$ against an oblivious adversary.*

Proof. The adversary defines a probability distribution on the set of possible input sequences of length $t(m,L)$ which allow an optimal maximum queue length L. Since at most mL packets can be delivered in each round (otherwise the adversary would have to exceed her limit L), the number of such sequences is finite, for any fixed m and L. The adversary could take the uniform distribution, however the only important thing is that every input gets a positive probability.

First suppose that the adversary presents a random input sequence composed of epochs. Every epoch begins with a sequence of length $t(m,L)$ sampled from the mentioned distribution, followed by mL rounds where no new packets are delivered and all queues are emptied.

Any randomized algorithm RA can be considered as a probability distribution on all possible deterministic algorithms, because all random decisions (depending on the input) can be made in advance. In particular, with every new epoch RA behaves like a randomly selected but otherwise deterministic algorithm starting from scratch. By the prerequisites, for every deterministic algorithm there exists an input sequence of length $t(m,L)$ that causes a queue of length $dcr(m,L)L$. Since all sequences are presented with at least some constant positive probability, sometimes a queue will get length $dcr(m,L)L$. More precisely, this happens with probability at least $1-\delta$ after a finite number of epochs depending on δ only, where $\delta > 0$ can be arbitrarily small.

We have shown for any randomized algorithm RA that a random input sequence (as specified above) causes a expected competitive ratio of at least $(1-\delta)dcr(m,L)$, where expectation refers to the randomness in both input and algorithm. By linearity of expectation and the pigeonhole principle, there also exists a particular input sequence that gives at least this expected competitive ratio. Now, expectation merely refers to the randomness on the algorithm's side. The adversary may present this sequence to RA, knowing RA but not the random bits that will be actually chosen, i.e., this adversary is in fact oblivious. $\diamond$

Now it follows from the deterministic bounds in [2, 5] and Section 3 of the present paper:

Corollary 7 *No randomized algorithm for BSP can achieve competitive ratio below $\log_2 m$ for $L = 1$. For any fixed L, no oblivious randomized algorithm can achieve competitive ratio below $\frac{1}{L\ln(1+1/L+\epsilon)}\ln m$.* $\diamond$

Remark 1. Obviously, the proof does not use very much of the structure of BSP, thus the result is not specific for BSP. It holds similarly for any "minmax" online problem where the input can be "reset to zero" arbitrarily often.

Remark 2. Theorem 6 is a demonstration of *von Neumann/ Yao principle* [11, 12], which asserts that, given some problem, if one shows a distribution over problem instances such that all deterministic algorithms incur an expected cost of at least x, then x is a lower bound on the cost incurred by the best possible randomized algorithm.

Remark 3. The result does not mean that the weaker $\ln m$ bound from [5] is entirely pointless. Our proof only shows that the worst case is reached in exponential time, with high probability. Appropriate randomized algorithms might still work better than deterministic ones most of the time. Thus, randomization could still reduce the "typical" maximum queue length in applications where the workload varies over time and rare cases of packet loss are acceptable. It may be interesting to explore the possibilities and tradeoffs. A specific question is: How does LQF behave when ties are broken randomly?

5 Greedy Algorithms in the VOQ Model

Because of the nature of VOQ switches, we will have queues that are blocked by others. If there is a queue chosen from an input port, all other queues in that port become *unavailable*. Likewise, if there is a queue chosen for an output port, all other queues targeting that output port become unavailable. However, an *available* queue could be of zero length. Any scheduling algorithm can output an *independent set* of μ available queues in one round, where $\mu = \min\{m, n\}$.

It is obvious that VOQ switches give rise to a two-dimensional generalization of BSP problem. This setting can also be viewed as a game on a matrix Γ of size $m \times n$, where each entry $\gamma_{i,j}$ specifies the queue length of $q_{i,j}$ – the queue at input i for output j. The rule for the scheduler can be translated as: The scheduler chooses at most one entry from each row and from each column. That is, if the entry $\gamma_{i,j}$ has been selected to output, no one from row i and column j can be a choice for the scheduler in this round. An immediate generalization of LQF is:

Input-Iterative LQF (I-LQF): In each iteration i from 1 to m, input port i will choose its longest available queue to output (breaking ties arbitrarily).

Output-Iterative LQF (O-LQF): In each iteration i from 1 to n, output i will choose a longest available queue targeting it to output (breaking ties arbitrarily).

By the matrix view, I-LQF selects queues row-by-row; O-LQF selects queues column-by-column. They both suffer from scenarios similar to HOL blocking in FIFO queues. W.l.o.g. we show for I-LQF:

Theorem 8 *For any fixed $L > 0$, I-LQF has a competitive ratio of at least m.*

Proof. The worst case is constructed as follows. In the beginning, the adversary delivers L packets to each of m queues in any column i, after which she delivers one packet to $q_{m,i}$ per round.

An optimal algorithm keeps on outputting $q_{m,i}$. This makes the optimal queue length L throughout. On the other hand, I-LQF tries to empty the queues from $q_{1,i}$ to $q_{m,i}$ one-by-one. It takes $(m-1)L$ rounds for I-LQF to reach $q_{m,i}$, when its length is mL. Hence, I-LQF is at least m-competitive. $\diamond$

It is apparently bad to stick to some fixed ordering of rows or columns. Therefore we propose two other, more promising greedy algorithms. The first one is

Global LQF (G-LQF): Choose the longest queue to output (breaking ties arbitrarily), then successively take the longest available queue, until all queues are blocked.

The results for BSP immediately imply that a "lazy" G-LQF algorithm that only outputs a packet from one queue at a time (instead of μ queues) has competitive ratio $(1+1/L)\ln(mn)+O(1)$.

G-LQF can work on μ queues simultaneously. Now imagine there are μ instances of "lazy" G-LQF's running one-by-one in one round. Due to the restriction in VOQ switches, some of these instances of "lazy" G-LQF cannot operate on a longest queue because it is blocked. Therefore, we expect a speedup by μ "lazy" G-LQF's no more than a factor of μ. We conjecture that the competitive ratio of G-LQF is between $(1+1/L)\ln\nu+O(1)$ and $(1+1/L)\ln(mn)+O(1)$, where $\nu=\max\{m,n\}$. The simplicity of the supporting queue argument (Section 2) gives hope for an upper bound proof, however the difficulty in two dimensions is that supporting queues can form different patterns that are hard to control in a case analysis.

Matrix LQF (M-LQF): Given the queue-length matrix $\Gamma(t)$ at time t. Find a zero-one $m\times n$ matrix $\Phi(t)$, in which each row/column has at most one 1, that maximizes $\Gamma(t)\cdot\Phi(t)$. We then output $q_{i,j}$ if corresponding $\phi_{i,j}(t)\in\Phi(t)$ is 1.

Apparently, M-LQF is a bit different from the other three although it is also greedy. Intuitively, it seeks for an independent set of queues with largest total length to output. Moreover, the adversary can simply deliver packets to only one row or column of length ν, and then the 1-dimensional lower bound $(1+1/L)\ln\nu+O(1)$ applies trivially. Although the algorithm is seemingly more efficient in eliminating long queues, we have to leave the upper bound of its competitive ratio an open problem.

Acknowledgment

The work 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).

References

[1] T. Anderson, S. Owicki, J. Saxe, and C. Thacker. High-speed switch scheduling for local area networks. *ACM Transactions on Computer Systems*, 11(4):319–352, 1993.

[2] A. Bar-Noy, A. Freund, S. Landa, and J. Naor. Competitive on-line switching policies. *Algorithmica*, 36:225–247, 2003.

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

[4] P. F. Dietz and D. D. Sleator. Two algorithms for maintaining order in a list. *19th ACM Symposium on Theory of Computing STOC'87*, pp. 365-372.

[5] R. Fleischer and H. Koga. Balanced scheduling toward loss-free packet queuing and delay fairness. *Algorithmica*, 38:363–376, 2004.

[6] M. G. Hluchyj and M. J. Karol. Queueing in high-performance packet switching. *IEEE Selected Areas in Commun.*, 6(9):1587–1597, 1988.

[7] M. J. Karol, M. G. Hluchyj, and S. P. Morgan. Input versus output queueing on a space-division packet switch. *IEEE Trans. Commun.*, 35(12):1347–1356, 1987.

[8] E. Leonardi, M. Mellia, F. Neri, and M. A. Marsan. Bounds on delays and queue lengths in input-queued cell switches. *Journal of the ACM* 50(4):520–550, 2003.

[9] N. McKeown, V. Anantharam, and J. Walrand. Achieving 100% throughput in an input-queued switch. In *INFOCOM'96*, pp. 296–302.

[10] A. Mekkittikul and N. McKeown. Scheduling VOQ switches under non-uniform traffic. Technical Report CSL-TR-97-747, Stanford University, 1997.

[11] J. von Neumann and O. Morgenstern. *Theory of Games and Economic Behavior.* Princeton University Press, Princeton, New Jersey, first edition, 1944.

[12] A. C. Yao. Probabilistic computations: Toward a unified measure of complexity. In *18th IEEE Symposium on Foundations of Computer Science, FOCS'77*, pp. 222–227.

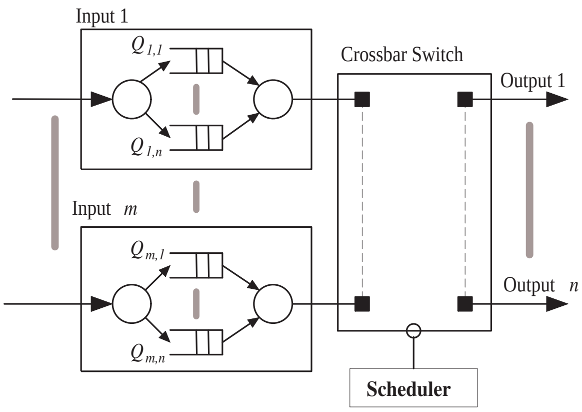

Figure 1: A VOQ switch with m input ports and n output ports.