Optimal On Demand Packet Scheduling in Single-Hop Multichannel Communication Systems

Maurizio A. Bonuccelli, Susanna Pelagatti

Dipartimento di Informatica, Università di Pisa, Corso Italia 40
56125 Pisa, Italy. Phone: +39-50-887255; Fax: +39-50-887226
e-mail: bonucce@di.unipi.it, susanna@di.unipi.it

Abstract

In this paper, we study the problem of on demand minimum length packet scheduling in single-hop multichannel systems. These systems allow parallel, simultaneous communication between a set of input users and a set of output users, provided that each input user and each output user is involved in at most one communication at a time. Each such communication must take place in a separate channel. Examples of these systems are those centered around switching networks, like crossbar switches, and WDM optical fiber networks. On demand scheduling requires that packets are scheduled upon receipt, and without changing the schedule of earlier packets.

On demand scheduling is performed by online algorithms. In the paper, we show that a large group of online scheduling algorithms, called maximal algorithms, are asymptotically optimal (in the worst case sense). A simple maximal algorithm, called First Fit, is used as an archetype of maximal algorithms. This result is established by first giving the competitive ratio of these algorithms (which is a little less than 3), and then by showing that no online algorithm can (asymptotically) perform better in the worst case. This last result is obtained by using the adversary technique. The above result is a worst case one. In order to get an insight of the average performance of maximal algorithms, we run a simulation experiment on randomly generated problem instances. The outcome of such an experiment is very encouraging: the ratio of the length of the schedules produced by maximal algorithms, and the lower bound on the length of the scheduling computed by the best off-line algorithm is less than 1.06. Thus, such algorithms can be usefully employed in practical cases.

KEY WORDS: single-hop multichannel systems, time division multiplexing, scheduling, online algorithms, complexity, competitive ratio.

1 Introduction

Single-hop communication systems are used to directly connect a given set of users. All the users so connected can exchange information without relaying it to intermediate entities (no store-and-forward is needed). When this is not the case, the system is a multi-hop one. Single-hop systems can be divided into two groups: single-channel and multichannel. Systems of the first group allow only one correct communication at a time, while in the second, it is possible to safely deliver information between several pairs of users simultaneously.

The above classification is an abstraction of communication systems, in which certain features of real systems, unessential when dealing with specific problems, are omitted. In particular, this abstraction is very useful for medium access control problems, since it captures the characteristics relevant to solve such problems in a wide variety of cases, allowing the attainment of results with a broad applicability.

In this paper, we shall consider single-hop multichannel communication systems. Many real systems fall into this category. For instance, switching networks, like crossbar switches or Benes networks, are single-hop multichannel systems. Other notable examples are purely optical networks, and some types of wireless systems, when operated with Wavelength Division Multiplexing (WDM) and Frequency Division Multiplexing (FDM), respectively.

In all these systems, uncoordinated exchange of information can cause the harmful phenomenon of collision. This happens when two or more entities simultaneously send information over the same channel, or to the same destination. Collided information is useless, and is therefore discarded. Obviously, it must be retransmitted later, with a waste of system resources, and an increase in the information delivery delay. This phenomenon is very well known since a long time, and its importance on the overall system performance is witnessed by the huge amount of work done to cope with it; even in the ISO/OSI protocol stack, one layer (called *medium access control*) is completely devoted to it [7, 20].

Protocols designed for medium access control can be divided into two groups, often called random access and deterministic. Under a random access protocol, an entity decides when to transmit according to some locally checked conditions. Examples of these protocols are Aloha (no condition is checked), slotted Aloha (timing condition is checked), and CSMA (channel occupancy condition is checked). These protocols, while simple, fast and truly distributed in nature, do not completely eliminate collisions, but only tend to reduce their number. Deterministic protocols are usually based on a coordination among the interested entities. Polling, token based (like token ring and token bus), as well as Time Division Multiple Access (TDMA) protocols fall into this group. These protocols completely avoid collisions, at the price of increased information delivery delay, or processing time.

The new technological advances have brought TDMA protocols into the forefront, and they have been intensively investigated recently [5, 9, 16, 19]. Such protocols are based on collision-free transmission time scheduling, with the objective of optimizing the system performance. Any collision-free schedule, however, does not lead to performance optimization, as the reader can easily realize. It is usually assumed that the performance measure to be optimized is the schedule length, also called makespan, since it represents the system throughput, and tends to minimize the delay, also.

TDMA based systems are assumed to be synchronous, and the information is packetized before being transmitted. Packetized communication is a de-facto standard, since it facilitates the system operations at negligible additional costs. Packet transmission is organized into successive *frames*, each divided into several *time slots*. Time slots size is identical to that of packets (which are assumed to all have equal length), and thus are the unitary pieces of information exchanged in the systems under investigation. In order to effectively deliver the scheduling information to all the involved parties, a control channel should be provided. Given the small amount of data carried by such a channel, it can be effectively accessed via a random access protocol, like CSMA/CA [7, 20].

The above packet scheduling problem has received much attention in the past, sometimes under the name of *time slot assignment*. The basic problem was investigated in the late 70's and early 80's [8, 15, 10] and was originally motivated by SS/TDMA satellite systems. Later on, several different side constraints were introduced, in order to consider specific system characteristics and special traffic features. Just to make a few examples, systems with different port capacities [4], or interconnected through low capacity links [6], or with communication set-up times (also called tuning latencies) [9] have been investigated, as well as multicast [12] or real time [21] messages. Very recently, the same model has been used to represent and study the scheduling of I/O operations [16].

Only off-line packet scheduling in single-hop multichannel communication systems has been studied up to now. Off-line algorithms need to know the entire set of requested communications before producing the schedule. Thus, a data gathering phase must precede the algorithm processing, which in turn precedes the actual schedule. So, an entity that wants to send packets to another entity, issues its request first, then waits till a certain quantity of requests are gathered (or a timeout is elapsed), before sending its demand to be processed. This waiting time can be very large. Besides, the entity may want to receive an answer to its request quickly, without waiting for other requests to be issued. Because of this, on-demand scheduling is becoming increasingly important, for data packets [1], and for communication traffic of other nature like video [18, 2]. An early attempt to investigate the on-demand feature of our problem can be found in [22, 14], where constrained parallel traffic streams are scheduled on demand, with the objective of minimizing the mean delay. There, a limited amount of collisions are allowed.

On-demand scheduling is performed by the so-called *online algorithms*. Such algorithms process requests as soon as they arrive. The scheduling decision taken by an online algorithm is solely based on the schedule already generated at the time a request is received, and cannot use information about requests not arrived yet. Besides, it cannot modify the already produced schedule. The performance of online algorithms is measured in terms of *competitive ratio* [13]. Let us consider an optimal off-line algorithm, OFLA, namely an algorithm always producing an optimal (i.e., minimum length) schedule based on the perfect knowledge of all past, present and future requests. The competitive ratio of a given online algorithm for the same problem is the maximum ratio between the length of the schedule that it produces, and the length of the schedule output by OFLA, taken over all possible request sequences. This is a worst case measure, and can be used as a performance guarantee.

Online algorithms have been proposed for a wide variety of problems (see [13] for a large survey). Online algorithms for problems related to the one considered in this paper have been recently proposed. In [11], two-stage shop scheduling is considered. In [3], an optimal online algorithm is given for edge coloring of graphs. The edge coloring problem solved in [3] can be seen as a special case of the problem investigated in this paper. In particular, it coincides with the case of no channel constraint (namely, when the number of channels is not smaller than the number of communicating entities) and of single packet communication. When multi-packet communication is allowed, the algorithm presented in [3] is no more polynomial (becoming pseudopolynomial). The channel constraint assumed in the present paper (again under the edge coloring formulation) has been considered in [17], where approximation heuristics have been proposed and evaluated. These heuristics, however, do not deal with the multi-packet communication (and in such cases they would have a pseudopolynomial time complexity), and are off-line.

The online nature of the algorithms considered here, as well as their independence on the actual number of users, makes them attractive for dynamic groups also, in which users join and leave the system when it is operational. Thus, these algorithms can be used in mobile networks, like indoor ad-hoc ones, where each room can be viewed as a single-hop system.

In Section 2, the problem is formally stated, and the technique used to assess the lower bounds on competitive ratios of online algorithms is introduced. Section 3 contains a description of an online algorithm for the above problem, called *first fit scheduling* (FF for short) as well as its competitive ratio. FF always produces a maximal schedule: it is introduced as an archetype of maximal algorithms. The established competitive ratio, as well as the optimality result given in the following, hold for any maximal algorithm, and are not a prerogative of first fit policy. In Section 4, the asymptotic optimality of maximal algorithms is established, by showing a set of requests for which no online algorithm can do better than their competitive ratio (asymptotically). The above results are of worst-case nature, and can be achieved in very special cases only. In order to have an insight on the average algorithms performance, we set up a simulation experiment performed on FF and on other maximal algorithms. The description of the experiment, of the involved algorithms, and of its outcome, are presented in Section 5. Finally, conclusions terminate the paper.

2 Problem Formulation

This section introduces the *single hop multichannel packet scheduling* problem along with some basic definitions and techniques.

We assume that there are n input users and n output users. Furthermore, we assume that there are c communication channels, with $c \leq n$. Input users issue one demand for each message they want to transmit. Messages are made of one or more packets. All packets have equal length. There is no preemption on packet transmission, while the packets that are part of the same message must not (but can) be transmitted at consecutive times. The system is synchronous, and packets transmission must start at times multiples of a given value. Thus, the communication is time slot based.

We assume that channel tuning latencies are negligible. Besides being time scheduled, the packets must be assigned a channel. As the reader will see later, this is not an issue, as it can be easily done after the scheduling (for instance, assigning the channels in the order packets are scheduled).

Packet scheduling in a time slot is represented by an $n \times n$ *time slot matrix* $T = (t_{ij})$ with 0-1 entries. The rows of T represent the input users, and the columns represent the output users. If $t_{ij} = 1$ in T, then one packet from input user i to output user j is scheduled for transmission in the time slot associated to T. In order to meet the collision-freeness requirement, at most one nonzero entry must be assigned to each row and column of any matrix T. Besides, a total of at most c nonzero entries must be present in T, one for each available channel. The computation of a schedule can be expedited by computing several identical time slot matrices at once. Such identical matrices are represented by *switching matrices* $S_k = (s_{ij}^{(k)})$. Switching matrices differ from time slot matrices in the value of the nonzero entries, that can be greater than one. Such a value is the number of (usually consecutive) time slots needed to transmit the packets scheduled in the matrix, and is called the matrix *length* l_k .

A *schedule* for a set of requests is a collection of switching matrices $S_1, S_2, \ldots, S_k$, such that $s_{ij}^{(1)} + s_{ij}^{(2)} + \ldots + s_{ij}^{(k)}$ is equal to the number of packets that input i demanded to transmit to output j. The length of such a schedule is $l_1 + l_2 + \ldots + l_k$. We say that a row or a column of a switching matrix with only zero entries is *exposed*, and is *covered* otherwise.

We are interested in finding minimum length schedules for a given set of messages that input users requested to transmit to output users.

An obvious lower bound LB to such a minimum schedule length is given by

$$\max\left\{\rho_i, \gamma_j, \frac{\tau}{c}, \forall i, j, 1 \leq i, j \leq n\right\}$$

where ρ_i is the total number of packets that input i demanded to transmit, γ_j is the total number of packets that the input users requested to send to output user j, and τ is the global number of packets demanded by all the input users, namely $\tau = \sum_{1 \leq i \leq n} \rho_i$. A polynomial time off-line algorithm always producing schedules with a length equal to LB exists [8, 10].

In this paper, we consider on demand packet scheduling. Messages are requested to be scheduled either one by one, or in groups, in an arbitrary order. In such requests, the number of packets forming each message is given, as well as the input and output users involved. The proposed algorithms must schedule the messages as soon as they are requested, and such a scheduling must be done without modifying the schedule of the previously demanded messages. Thus, such algorithms are online ones. Involved input and output users are immediately notified of the message schedule time.

A scheduling is *maximal* if none of the packets scheduled in its last matrix can be placed in an earlier matrix without violating the collision-freeness constraint (namely, at most one nonzero entry per row or column), or exceeding the channel capacity. A scheduling algorithm is called *maximal* if it produces maximal schedules at each step.

The performance of online algorithms is measured in terms of their *competitive ratio*, namely how poorly they can behave compared to the best off-line algorithm [13]. For our scheduling problem, the competitive ratio of an online algorithm, OLA, is the maximum ratio between the length of the schedule it produces, and that of the optimal off-line algorithm (which in turn is equal to the lower bound LB [8, 10]), taken over all possible request sequences.

Sometimes, it is possible to establish a lower bound on the competitive ratio of *any* online algorithm. An online algorithm attaining such a lower bound is called *optimal*. In order to compute competitive ratio lower bounds, a technique based on an imaginary adversary is usually employed. The adversary knows the behaviour of the algorithm. Based on such knowledge, he submits the worst possible sequence of requests to the algorithm, which is in turn used to establish the lower bound.

3 Maximal Algorithms and their Performance

We now present a class of online algorithms for the single hop multichannel packet scheduling problem. For the sake of simplicity, such a class (termed maximal algorithms) is introduced by describing a specific member of it, called First Fit (FF for short). The algorithm builds the schedule in terms of switching matrices. It starts with an empty switching matrix, and places the first request in it. When a new request arrives, it places such a request in the first possible matrix, namely the first matrix it considers with less than c nonzero entries, and with the row and column relative to such a request both exposed. The matrices are scanned in the order of their generation, which is the order of scheduling (earliest first). Then, it eventually truncates the request to the value of the nonzero entries already present in the matrix, and the process is repeated until the request is entirely scheduled. If the request (or one of its fractions) does not fit in any switching matrix produced so far, then it creates a new empty (i.e. zero) matrix, placing the remaining fraction of the request in it, and putting the matrix at the end of the schedule. If more messages are requested at once, they are considered for scheduling one at a time, in an arbitrary order. Because of its behaviour, the algorithm is called *First Fit*. A more accurate description follows. Let us assume that a request from i, with destination j, and a_{ij} packets long, arrives.

Algorithm: First Fit

STEP 1: If no matrix with less than c nonzero entries, and with both row i and column j exposed in the schedule exists, then create a new one, filled with zeroes. Assign the value a_{ij} to the entry in row i and column j, and place the matrix at the end of the schedule.

STEP 2: If at least one such matrix exists, then find the smallest index h of those matrices. Assign to its (i, j) entry the value $l'_h = min\{a_{ij}, l_h\}$.

STEP 3: If $a_{ij} > l'_h$, then set $a_{ij} = a_{ij} - l'_h$, and go to step 1.

STEP 4: If $a_{ij} < l'_h$, then split S_h into two matrices S'_h and S''_h, placing them where S_h was. The nonzero entries of S'_h will have a value equal to a_{ij}, and those of S''_h, equal to $l_h - a_{ij}$. Set the (i, j) entry of S'_h to a_{ij}.

The question of how many times the algorithm cycles when the requested transmission does not immediately fit into a switching matrix (i.e. when step 3 is performed) arises. Let us consider the first request. Obviously, it does not perform step 3, and so the schedule contains exactly one switching matrix, S_1, after it. If the second request can be placed into S_1, and its value is identical to l_1, then no new switching matrix is added to the schedule. Otherwise, exactly one matrix, S_2, is added. It is easy to see that each new request can cause the generation of at most one new switching matrix in the schedule, and such a new matrix is created either in step 1, or in step 4. Thus, at most k matrices are in the schedule when the $(k+1)$-th request is issued. Hence, at most k matrices must be inspected for scheduling such a new request. Since each inspection takes constant time (provided the generation of a zero matrix does), the overall time complexity of scheduling the k-th request is $O(k)$.

Observe that the matrix representation of the schedule is not the only possible one, and has been introduced for the sake of simplicity: for instance, a list of the scheduled pairs can effectively be used, instead of the switching matrices. This, together with the fact that only the number of available channels is actually used in scheduling the requests, make the algorithm applicable in mobile settings too, like indoor ad-hoc networks, where each room is a single-hop system, and where the number of users is unknown, being highly dynamic.

Finally, the available channels must be assigned to the scheduled packets. This is not a problem, since the channels are all alike, and comes for free with the scheduling: just assign the first channel to the communicating pair with the lowest indexed input user, the next channel to the pair with the second lowest indexed input user, and so on.

Theorem 1 *Algorithm FF is a maximal algorithm*

Proof: The maximal schedule property is proved by induction. Obviously, the schedule produced by first fit after the first demand is maximal. Let us assume that such a schedule is maximal after the first $k-1$ demands, and suppose that the k-th demand is issued by input user i, asking to transmit $a_{i,j}$ packets to output user j. If a new switching matrix must be created in step 1 for this entry, then it did not fit into the already generated switching matrices unless some constraint is violated, and so the schedule is maximal. If the entry exactly fits into a matrix already in the schedule, then the schedule obviously remains maximal. Thus, we are left with one case: the splitting of a switching matrix in the current schedule (step 4). Let us assume that the split matrix is S_h. If we simply split it, without placing $a_{i,j}$ in such matrix, we have not actually changed the schedule, but only its representation (instead of an l_h long matrix, we have two identical matrices with length, respectively, $l_{h'}$ and $l_{h''}$, such that $l_{h'} + l_{h''} = l_h$). So, if $a_{i,j} = l_{h'} < l_h$, we add $a_{i,j}$ to $S_{h'}$, and we are in the previously considered case of an entry exactly fitting into a switching matrix. Finally, observe that the actual schedule of any entry is done by repeatedly performing the above cases, one at a time. Since each of them maintain the maximal property, such a property is still retained after the k-th demand is completely scheduled. •

We are now in a position to establish the algorithm competitive ratio.

Theorem 2 *The competitive Ratio of maximal algorithms is 3-2/LB*

Proof: Let us consider a generic schedule produced by a maximal algorithm, let S_h be the last switching matrix of such schedule, and let $s^h_{i,j} > 0$. This means that entry

$a_{i,j}$ has been (partly) scheduled in the last matrix of the schedule. Since the schedule is maximal, $s^h_{i,j}$ has not been scheduled in earlier matrices because they either already had c nonzero entries, or they had a nonzero entry in row i or in column j. Thus, the schedule can be divided in two parts: one containing the switching matrices with c nonzero entries, and the other with the remaining switching matrices. Let l' and l'' be the total length of these two parts of the schedule. Obviously, $l'' \leq \rho_i + \gamma_j - s^h_{i,j}$ (since otherwise $s^h_{i,j}$ is counted twice, once in ρ_i and once in γ_j). Let $\beta = cl'$. Then, β represents the total traffic scheduled in the switching matrices with c nonzero entries. So, $\beta \leq \tau - l''$, and $l' \leq \frac{\tau - l''}{c} = \frac{\tau}{c} - \frac{l''}{c}$.

Thus, $SL = l' + l'' \leq \frac{\tau}{c} - \frac{l''}{c} + l'' = \frac{\tau}{c} + \frac{c-1}{c}l'' \leq \frac{\tau}{c} + \frac{c-1}{c}(\rho_i + \gamma_j - s^h_{i,j})$.

Since $LB \geq max(\frac{\tau}{c}, \rho_i, \gamma_j)$, we have :

$SL \leq LB + 2\frac{c-1}{c}LB - \frac{c-1}{c}s^h_{i,j} = 3LB - \frac{LB}{c} - \frac{c-1}{c}s^h_{i,j} = 3LB - \frac{LB+(c-1)s^h_{i,j}}{c}$. Let us now consider the term $\frac{LB+(c-1)s^h_{i,j}}{c}$. We shall show that if such a term is not larger than one, the maximal schedule is optimal. If $\frac{LB+(c-1)s^h_{i,j}}{c} \leq 1$, then $LB + cs^h_{i,j} \leq c + s^h_{i,j}$. If $s^h_{i,j} > 1$, this inequality can hold only when $c = 1$. But in this case all the entries must be sequentially scheduled, and the maximal schedule is optimal (any schedule is optimal). So, let us assume that $s^h_{i,j} = 1$. In this case, the inequality becomes: $LB + c = c + 1$, which holds (with the equal sign) only when $LB = 1$, namely when at most c entries with value one must be scheduled, and no two such entries are in the same row or column. But in this case, any maximal algorithm will produce a schedule with length one, and is therefore optimal (when a new demand is issued, less than c entries are in the unique switching matrix built so far, and its row and column are both exposed. Hence, it is placed in that matrix). Summarizing, $SL \leq 3LB - \frac{LB+(c-1)s^h_{i,j}}{c} < 3LB - 1$. Since SL must be integer, and LB is integer, we have that $SL \leq 3LB - 2$. •

4 A Lower Bound on Competitive Ratios

In this section, we shall show that no online algorithm can perform asymptotically better than any maximal algorithm, as far as the worst case concerns.

The result presented in this section is based on the adversary technique, described in Section 2. Let us recall that the adversary submits one (or more) demands at a time, waits for the schedule output by the algorithm, and then decides whether to continue submitting demands or not, and in the affirmative case, chooses the next demand to submit. The purpose of the demands generated by the adversary is to force the worst possible performance for the algorithm.

In the following, we shall show that the competitive ratio of any online algorithm cannot be smaller than $3 - 3/LB$. This result, together with the one stated in Theorem 2, establish the worst case optimality of any maximal algorithm, for $LB \rightarrow \infty$.

Theorem 3 *The competitive ratio of any online single hop multichannel packet scheduling algorithm is at least 3-3/LB*

Proof: The proof is based on the adversary technique. The adversary strategy consists in requesting the scheduling of packets according to a template matrix t, adapting the specific requests issued, to the schedule generated by the generic algorithm under consideration.

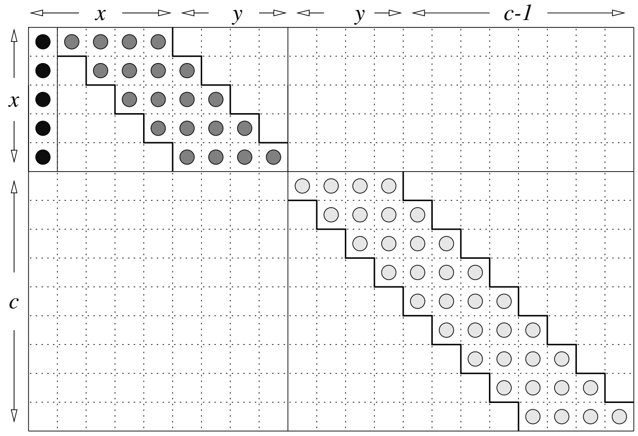

Figure 1: General structure of a template matrix. In this case, $1 < y = 4 < x = 5$.

The structure of a template matrix submitted by the adversary is shown in Fig. 1. The nonzero entries of the template matrix, $t_{i,j}$, are all set to 1, and are shown as circles in the picture. Nonzero elements have the following indices:

1. $x + 1 \leq i \leq x + c$ and $y + i \leq j \leq 2y + i - 1$ (light gray circles)
2. $1 \leq i \leq x$ and $i + 1 \leq j \leq y + i$ (dark gray circles)
3. $1 \leq i \leq x$ and $j = 1$. (black circles)

The bound holds for any y such that $1 < y < x$. Proper values for x and c, achieving the target bound, will be discussed later. Not all these entries will be requested by the adversary, which adapts its strategy according to the generated schedule. The adversary can requests several entries at once, and will start with those of the above first group of indices. In particular, it starts requesting the entries $t_{i,j}$ with all i such that $x+1 \leq i \leq x + c$ and, for each i, $j = y + i$ Observe that these entries form a sub-diagonal, namely have different row and column indices, and so can be scheduled in the same switching matrix, since they are exactly c in number. If the algorithm schedules these entries in one switching matrix, then the adversary requests the next sub-diagonal, namely those $t_{i,j}$ entries with $x + 1 \leq i \leq x + c$ and, for each i, $j = y + i + 1$. The above observation obviously holds for this sub-diagonal, too. The adversary continues this way, requesting one sub-diagonal of the first group at a time, with increasing column index, until either all these sub-diagonals are requested, or one sub-diagonal is scheduled in more than one switching matrix. Let d be the number of sub-diagonals scheduled so far.

At this point, the adversary starts requesting entries with indices in the second group, resetting to zero the entries of the first group not requested yet. Note that none of the entries not yet scheduled can be placed in the first $d-1$ (d if $d = y$) switching matrices, since each of them already contains c non zero entries. Like for the previous group, some of the entries of this second group could be not requested by the adversary. In particular, if $d < y$, then only the first d sub-diagonals will be requested, and the other entries of this group will be reset to 0 in the template matrix. So, the second group of nonzero entries will become those $t_{i,j}$ such that $1 \leq i \leq x$ and $i+1 \leq j \leq d+i$. These entries can be requested in any order. At the end of this second part of the schedule, the adversary checks the length of the schedule generated by the algorithm. If such a length is not smaller than $3d$, then the adversary resets to zero all the entries in the third group, and stops requesting entries.

Observe that the schedule produced so far is at least $2d-1$ long, since the entries of the second group require at least d switching matrices because in each row there are d non zero entries, and none of these entries can be scheduled in the first $d-1$ switching matrices, which contain c entries each.

Finally, the adversary picks the largest set of rows whose indices are in $\{1, 2, 3, \ldots, x\}$ (namely, row indices of entries in the second group) and have an identical schedule, that is their non zero entries are scheduled in the same set of d switching matrices. If the cardinality of the above set of rows is larger than $d+1$, only $d+1$ of them are considered. Let A denote such a set, and let its cardinality be b. Now, the adversary sets to zero the entries of the third group whose row indices are not in A. Now, the non zero entries remaining after the last resetting are requested, in any order, and one at a time.

The length of the schedule generated by the algorithm is $2d-1+b$ at least, since there are $d-1$ switching matrices with c entries, other d switching matrices with entries in the second group covering all the rows of entries in the third group, and b switching matrices with the entries of the third group (which must be scheduled in b different switching matrices because all are in column 1).

Let us now compute the lower bound on the schedule length. The rows with entries in both group 2 and group 3 of indices have sum $d+1$, while all other rows have sum equal to d. The first column has sum $b \leq d+1$, and all the other columns have sum equal to d. Finally, the total sum of template matrix entries is equal to $dc + dx + b$, where the first addendum represents the entries in the first group of indices, the second represents the entries of the second group, and the third those of the last group. So, $LB = max\left\{d+1, \lceil\frac{dc+dx+b}{c}\rceil\right\}$. We want that $LB = d+1$. Thus, we must have $b = d+1$, and $dx + b \leq c$.

The equality $b = d+1$ must hold for any online scheduling algorithm. This is true if we have a sufficiently large number of rows, namely when x is sufficiently large. In other words, we are looking for the minimum number x of rows with exactly d entries equal to 1 (and all the other entries equal to 0), and such that in any schedule of length λ, with $d \leq \lambda \leq 2d+2$, at least z such rows have an identical schedule. In particular, we are interested to the case $z = d+1$. Our goal is reached when $x \geq \left[\frac{d}{2}(d+1)(d+2)\ldots\lambda\right]+1$.

Let us now turn our attention to the other constraint, namely $dx + b \leq c$. This inequality holds when $d(x+1)+1 \leq c$. Observe that it is always possible to find proper values for x and c meeting the above constraints, even though such values are very large.

So, with the proper values for x and c, the adversary forces any online algorithm to produce a schedule with length $3d$ while the lower bound on the schedule length is $d+1$. Hence, the competitive ratio of any online algorithm is $3 - 3/LB$. •

Observe that the template matrix used by the adversary to get a worst case behaviour of any online algorithm has a rather special structure. In particular, it is a 0-1 matrix. This makes the results rather strong, since it rules out the possibility that message sizes can be influential in forcing online algorithms to have a bad performance. On the other hand, we are left with the question that worst case behaviour can be very uncommon. In the next section, we present the results of a simulation experiment set up for establishing the average performance behaviour of several maximal algorithms. The results presented show that maximal algorithms perform quite well on the average, and so worst case matrices are very rare indeed.

5 Heuristics and their Average Performance

In this section, we introduce three maximal algorithms. Two of them are based on the Best Fit policy, while the third randomly schedules the requests in the already generated switching matrices. The worst case results derived in the previous sections hold for these heuristics, too. Then, we describe a simulation experiment set up for getting insights into the average performance of the above algorithms, as well as First Fit. The results of the simulation experiment are then commented.

5.1 Heuristics

The first two heuristics, are based on the Best Fit policy. Contrary to First Fit, where the requested entry is scheduled into the earliest switching matrix that can host it without violating the constraints, Best Fit considers the set A of all the matrices in which the demanded entry can fit (without violating the problem constraints, obviously). Algorithm *Best Fit minimum* (BFmin), schedules the entry in the switching matrix with the smallest number of nonzero entries among those in A, while *Best Fit Maximum* (BFMax), schedules it in the switching matrix with the largest number of nonzero entries among those in A. The third heuristic, called RAND for *Random*, places the new entry in a matrix randomly chosen in A. Obviously, a truncation of the entry, like in FF, is performed when the request is larger than the nonzero entries already in the selected matrix. In this case, the algorithm is repeated on the unscheduled part of the entry. A more formal description of the algorithms can be easily obtained from that of FF (see Section 3), by properly changing Step 2.

The computational complexity of RAND is the same as FF, while BFmin and BFMax can have a higher complexity because of the matrix search phase. Such a search can increase the complexity from $O(k)$ to $O(kh)$, when A contains $h, (h < k)$ matrices.

Finally, it is easy to see that the above three heuristics are maximal algorithms: a proof similar to that of Theorem 1 suffices.

5.2 Average Performance

To evaluate the average performance of the proposed maximal algorithms (FF, BFmin, BFMax, RAND), we simulated their behaviour for different values of the parameters, and computed their competitive ratio. In the experiment, we set the number of input/output users to n = 10, 100, 1000, and for each value of n we selected two values of c ($c = n/2$ and $c = n$). In each experiment, we set the length of the message stream to 1000 and repeated the experiment 1000 times, on different data. The source and destination of each message were taken with uniform distribution in the interval $[1, n]$, while the message length was taken from a uniform distribution in the integer interval $[1, 100]$.

Tab. 1 and Tab. 2 shows the mean and variance (with respect to the lower bound LB) of each experiment.

		First fit		*Random*	
c	n	mean	variance	mean	variance
2	10	1.000962	0.000773	1.000979	0.000795
5	10	1.010054	0.005324	1.008722	0.004393
8	10	1.049447	0.023014	1.045925	0.022177
5	100	1.003701	0.001781	1.003459	0.001686
50	100	1.141723	0.075613	1.114763	0.069276
95	100	1.002149	0.009490	1.011174	0.018916
5	1000	1.003244	0.001359	1.003230	0.001459
500	1000	1.004056	0.021790	1.019787	0.032343
990	1000	1.003533	0.019649	1.022655	0.035293

Table 1: Simulation results: mean and variance of the measured competitive ratio.

		Best fit min		*Best fit max*	
c	n	mean	variance	mean	variance
2	10	1.000980	0.000762	1.000986	0.000783
5	10	1.008785	0.004910	1.010336	0.005572
8	10	1.041410	0.019203	1.048217	0.022300
5	100	1.003237	0.001952	1.003728	0.001844
50	100	1.053354	0.055059	1.136553	0.075932
95	100	1.029259	0.030833	1.001498	0.005729
5	1000	1.002958	0.001637	1.003245	0.001301
500	1000	1.035986	0.049658	1.002576	0.014694
990	1000	1.034962	0.048420	1.003291	0.016095

Table 2: Simulation results: mean and variance of the measured competitive ratio (best fit).

As the reader immediately realizes, the average performance of the proposed maximal algorithms is extremely good, and is almost always smaller than $1.1LB$. This means that these maximal online algorithms produce (on the average) schedules less than 10% longer than those produced by optimal off-line algorithms. So, the instances leading to a 200% schedule length increase on the lower bound (20 times the average increase) are rare, and online algorithms are a viable alternative to off-line ones.

The proposed heuristics do not show notable differences in their average performance. However, RAND and BFmin are slightly better than the other two. The effectiveness of a random choice in on demand computation is relatively common [13], and so this outcome of the simulation experiment is not very surprising.

6 Conclusions

In this paper, we investigated the problem of packet scheduling in single-hop multichannel networks. The scheduling must be performed on demand, and the produced timetable must be collision free. We introduced a broad class of scheduling algorithms, called *maximal*, and derived their competitive ratio, namely $3-2/LB$ (LB is the lower bound on the minimum schedule length). Then, we established that the competitive ratio of *any* on demand scheduling algorithm is $3-3/LB$ at least, thus assessing the (asymptotic) optimality of maximal algorithms. These results are worst-case ones. Finally, we described a simulation experiment, set up for getting an insight in the average performance of on demand scheduling algorithms. The experiment was run on four heuristics, and showed that they perform extremely well (within 10% the lower bound) in practice.

Several problems remain open. Among them, we cite a worst-case analysis of randomized algorithms, and on demand packet scheduling for specific single-hop multichannel systems, like those with non-negligible tuning latencies, or with variable channel bandwidth, and for particular traffic, like periodic real time, or multicast.

References

[1] S. Acharya and S. Muthukrishnan. Scheduling on-demand broadcasts: new metrics and algorithms. In *Proc. of 3rd Annual European Symp. on Algorithms*, volume 979 of *LNCS*, pages 538–553, Corfu (Greece), Sept. 1995. Springer: Berlin.

[2] S. Aggarwal, J. A. Garay, and A. Herzberg. Adaptive video on demand. In *Proc. of the 4th ACM/IEEE Conference on Mobile Coputing and Networking - Mobicom'98*, pages 43–54, Oct. 1998.

[3] A. Bar-Noy, R. Motwani, and J. Naor. The greedy algorithm is optimal for on-line edge coloring. *Information Processing Letters*, 44:251–253, 1992.

[4] P. Barcaccia and M. A. Bonuccelli. A polynomial time optimal algorithm for time slot assignment in variable bandwidth systems. *ACM/IEEE Transactions on Networking*, 2(3):247–251, 1994.

[5] P. Barcaccia, M. A. Bonuccelli, and M. Di Ianni. Minimum length scheduling of precedence constrained messages in distributed systems. In L. Bouge, P. Fraigniaud, A. Mignotte, and Y. Robert, editors, *Proc. of EuroPar '96*, volume 1024 of *LNCS*, pages 594–601, Lyon (France), Aug. 1996. Springer: Berlin.

[6] A. A. Bertossi, G. Bongiovanni, and M. A. Bonuccelli. Time slots assignment in SS/TDMA systems with intersatellite links. *IEEE Transactions on Communication*, 35:602–608, 1987.

[7] D. P. Bertsekas and R. Gallagher. *Data Networks*. Prentice-Hall: Englewood Cliffs, NJ, second edition, 1992.

[8] G. Bongiovanni, D. Coppersmith, and C. K. Wong. An optimal time slot assignment for a SS/TDMA system with variable number of trasponders. *IEEE Transactions on Communication*, 29:721–726, 1981.

[9] M. S. Borella and B. Mukherjee. Efficient scheduling of nonuniform packet traffic in a WDM/TDM local ligthwave network with arbitrary transceiver tuning latencies. *IEEE J. on Selected Areas on Communication*, 14:923–934, June 1996.

[10] R. E. Burkard. Time-slot assignment for TDMA-systems. *Computing*, 35:99–112, 1985.

[11] B. Chen and G. J. Woeginger. A study of on-line scheduling two-stage shops. In D. Z. Du and M. Pardalos, editors, *Minimax and Applications*, pages 97–107. Kluwier Academic Publ.: Norwell, MA, 1995.

[12] W. T. Chen, P. R. Sheu, and J. H. Yu. Time slot assignment in TDM multicast switching systems. *IEEE Transactions on Communication*, 42:149–165, Jan. 1994.

[13] A. Fiat and G. J. Woeginger, editors. *Online Algorithms: the state of the art*, volume 1442 of *LNCS*. Springer: Berlin, 1998.

[14] B. Hajek and T. Weller. Scheduling non uniform traffic in a packet switching system with large propagation delay. *IEEE Transactions on Information Theory*, 41(2):358–365, Mar. 1995.

[15] T. Inukai. An efficient SS/TDMA satellite assignment algorithm. *IEEE Transactions on Communication*, 27:1449–1455, 1979.

[16] R. Jain, K. Somalwar, J. Werth, and J. C. Browne. Heuristics for the scheduling I/O operations. *IEEE Transactions on Parallel and Distributed Systems*, 8(3):310–320, Mar. 1997.

[17] R. Jain and J. Werth. Analysis of approximate algorithms for edge-coloring bipartite graphs. *Information Processing Letters*, 54:163–168, July 1995.

[18] J. McManus and K. Ross. Video on demand over ATM: constant rate transmission and transport. In *Proc. of IEEE Infocom '96*, volume 3, pages 1357–1362, 1996.

[19] G. N. Rouskas and M. H. Ammar. Analysis and optimization of transmission schedules for single-hop WDM networks. *ACM/IEEE Transactions on Networking*, 3:211–221, Apr. 1995.

[20] A. S. Tanenbaum. *Computer Networks.* Prentice-Hall: Englewood Cliffs, NJ, third edition, 1996.

[21] H. Y. Tyan, C. J. Hou, B. Wang, and C. C. Han. On supporting time-constrained communications in WDMA-based star-coupled optical networks. In *Proc. of the IEEE Real Time Systems Symposium*, pages 175–184, 1996.

[22] T. Weller and B. Hajek. Scheduling non uniform traffic in a packet switching system with small propagation delay. In *Proc. of IEEE Infocom '94*, June 1994.