Computer Networks 56 (2012) 3274–3286

Contents lists available at SciVerse ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Optimal algorithms for the batch scheduling problem in OBS networks

Gustavo B. Figueiredo [a], Eduardo Candido Xavier [b], Nelson L.S. da Fonseca [b,*]

[a] Department of Computer Science, Federal University of Bahia, Brazil
[b] Institute of Computing, University of Campinas, Brazil

ARTICLE INFO

Article history:
Received 25 August 2011
Received in revised form 16 April 2012
Accepted 11 June 2012
Available online 19 June 2012

Keywords:
OBS networks
Channel scheduling
Batch scheduling

ABSTRACT

This paper introduces optimal batch scheduling algorithms for the problem of scheduling batches of bursts in optical burst switching networks. The problem is modelled as a job scheduling problem with identical machines. The consideration of previously scheduled bursts in the scheduling allows such modeling. Two optimal algorithms with polynomial time complexity are derived and evaluated. Results show that the algorithm that allows re-scheduling of previously scheduled bursts leads to preferred solutions.

Moreover, an extended version of the JET reservation protocol is proposed for efficient handling of batches of bursts. Results obtained via simulation prove the superior performance of the BATCHOPT algorithm.

1. Introduction

In Optical Burst Switching (OBS) networks, packets are aggregated at edge nodes to create transmission units called bursts. A control packet is transmitted out-of-band, ahead of the burst, so that bandwidth can be reserved for the data burst. The control packet carries information about the burst, such as size and offset time (defined as the time interval separating the arrival of the control packet and that of the data burst). A scheduling mechanism then reserves bandwidth of the output channels of nodes in the core network for incoming bursts, based on the information carried by their control packets. Since the nodes at the network border do not wait for the confirmation of bandwidth reservation to transmit a burst, the incoming burst will be discarded at a core node if bandwidth has not been reserved. [16–23,25]

The Just-Enough-Time (JET) [1] is a commonly used protocol for resource reservation in OBS networks. JET reserves the channel for the duration of the transmission of a burst, starting at its expected arrival time (given by the offset time minus the burst processing time). If this request for bandwidth reservation is granted, a new offset time is calculated, and this information is inserted into the control packet being forwarded to the next hop in the route.

Since bursts have different offset times, they may arrive in a different order than that of their control packets. This can lead to fragmentation of the occupancy of the output channels, since the occupancy pattern of output channels typically alternates between periods of occupancy and period of idleness, called void intervals. These void intervals can be used to accommodate the transmission of new bursts. Indeed, a void interval, I_j, defined by its starting, s_j, and its ending time, e_j, can be allocated to a burst with an arrival time, t', and departure time, t'' if and only if $s_j \leqslant t'$ and $t'' \leqslant e_j$.

Most of the existing algorithms for burst scheduling provide greedy processing to individual bursts [23,24,26–28]. However, this approach can lead to the loss of bursts which could be avoided if the arrival time were previously known. One way of ameliorating this type of loss is to gather reservation requests during a certain time interval, and then schedule them as a batch of requests. The objective of this process is to maximize the number of bursts transmitted, i.e., to minimize the loss of bursts. For this reason, the occupancy of the output channels by the

* Corresponding author. Tel.: +55 19 37885878; fax: +55 19 37885847.
E-mail addresses: gustavo@dcc.ufba.br (G.B. Figueiredo), ecx@ic.unicamp.br (E. Candido Xavier), nfonseca@ic.unicamp.br (N.L.S. da Fonseca).

1389-1286/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.comnet.2012.06.005

bursts can be modeled as an interval graph [32] with the solution of the burst assignment problem given by the solution of the coloring of an interval graph problem.

Previous work [10,11] on batch scheduling in OBS networks have addressed the channel reservation problem as a job scheduling problem with non-identical machines. However, such modeling leads to an NP-hard problem. As a consequence, heuristics have been proposed to solve the problem. In this paper, we show how the problem of batch scheduling in OBS networks can be formulated as a job scheduling problem with identical machines. We, then, introduce two optimal algorithms: one for networks with no prioritized requests and which has linear time complexity and the other for networks with prioritized requests which involve polynomial time complexity. In addition to low computational complexity, the results derived via simulation show that the proposed algorithms produce a lower probability of blocking than existing heuristics. These algorithms differ in respect to the re-scheduling of previously scheduled bursts. Moreover, the JET protocol is extended to operate efficiently with batch scheduling algorithms. Such an extension is compatible with the JET protocol, since its operation is exactly that of the JET protocol for scheduling individual bursts.

This paper is organized as follows. Section 2 presents concepts related to batch scheduling. Section 3 presents related background and notation. Section 4 reviews related work. Section 5 presents the new optimum batch scheduling algorithms. Section 6 presents an extended version of the JET protocol. Section 7 presents numerical examples and Section 8 draws some conclusions.

2. Batch scheduling in OBS networks

To minimize the chance of loss of burst, an algorithm should allocate bandwidth for a burst so that the chances of allocation to upcoming bursts is maximized. A channel is considered available for accommodation of a burst if there is a void large enough to accommodate the request. If no such channel is available, the request will be lost.

The scheduling algorithms proposed in [29–31,20] employ greedy strategies to reserve resources, but these strategies use only the information on individual control packets when they arrive without consideration of the overall demand during intervals between control packet arrivals.

Fig. 1a provides an example of a situation in which greedy algorithms fail to allocate resources for bursts: Let A, B and C be control packets arriving in this order and let their corresponding bursts arrive in that same order. These bursts can be accommodated only in channels 1 and 2 (dashed lines). Note that if channel 2 were used for burst A and channel 1 for burst B (as could happen if the algorithms proposed in [20,31,30] were used) there would be no way to accommodate burst C. On the other hand if channel 1 is used by burst A and channel 2 by bursts B and C, the result is a no loss scenario. This example illustrates the failure of scheduling due to lack of knowledge about future requests.

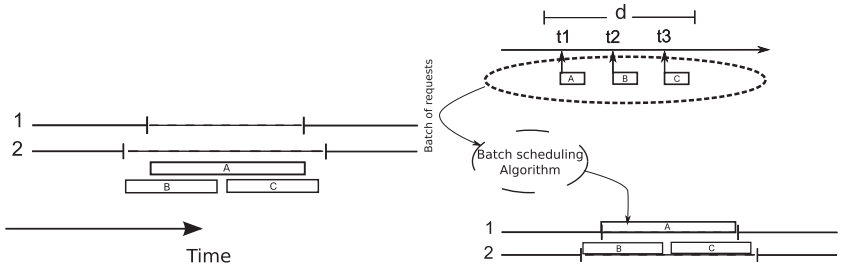
To decrease the loss of bursts, a new class of batch scheduling algorithms were proposed in [10,11]. The idea here is to group the largest possible number of control packets and process them together. After gathering information on the control packets, they are ordered and assigned to the channel with the smallest index value that has available wavelength to accommodate the burst (Fig. 1b). One main characteristic of the heuristics presented in [10,11] is that they collect requests arriving in time intervals of a fixed duration before processing them. This strategy, however, generates losses when the beginning of batch processing succeeds a request starting time.

3. Background and notation

The batch scheduling problem in OBS networks (denoted BS-OBS) can be stated as follows: Let M be a set of k channels, and let I be a batch of n requests formed by control packets. On each channel $m_i \in M$ there can be previously scheduled requests and voids resulting from lack of channel occupancy. These can be used to allocate the requests in order to maximize the number of requests granted. Each request in the control packet $J_x = (s_x, e_x, w_x) \in I$ is identified by a 3-tuple; the arrival time of the burst, (s_x), the finishing time of the burst, (e_x) and the request weight indicating the priority of the scheduling burst, (w_x). The objective is to find an allocation for a subset of requests such that the sum of the weights of the requests granted is maximized.

The offset time of a control packet (s_x, e_x, w_x) arriving at time t is given by $s_x - t$. The control packet must be processed during this offset time, otherwise its corresponding burst will be lost.

Next, some definitions will be introduced to facilitate the understanding of the mathematical formulation of the BS-OBS problem.



(a) Example of bad instances for greedy strategies. (b) Batch scheduling of channels leading to no losses.

Fig. 1. Examples of how batch scheduling can avoid losses of bursts.

Let $G = (V,E)$ be a graph, where $V(G)$ represents the set of vertices of G and $E(G)$ the set of edges. Given a vertex $u \in V(G)$, we define the adjacency/neighborhood of u by $Adj(u) = \{v \in V(G); (u,v) \in E(G)\}$. A subgraph H of G is a graph such that $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. The degree of vertex u in the graph G, denoted by $d(u|G)$, is the size of the set $Adj(u)$. The subgraph H of G is induced by $V(H)$ if for every pair $u, v \in V(H)$ we have that $(u,v) \in E(H)$ if and only if $(u,v) \in E(G)$. A clique of a graph G is a set $C \subseteq V(G)$ such that $\forall u, v \in C$; $(u,v) \in E(G)$. A clique C is *maximal* if there is no other clique C' in G such that $C \subset C'$ [32].

The graph G is called an interval graph if there is a correspondence/bijection between the set of vertices and a set of intervals on the real line, such that there is an edge between two vertices if and only if the correspondent intervals intersect, i.e., $(u,v) \in E(G) \Leftrightarrow I_v \cap I_u \neq \emptyset$ [32]. Interval graphs have several properties that can be used to solve problems in combinatorics. The fact that interval graphs can be recognizable and colorable in linear time [5,9] will be explored in this paper.

Interval graphs are typically used in the solution of the job scheduling problem, which can be stated as follows: let $I = \{J_1 = (s_1, e_1, w_1), \ldots, J_n = (s_n, e_n, w_n)\}$ be a list of n jobs. Moreover, there are k machines with the same processing capacity that are used to process these jobs. Initially, all machines are free starting at time 0. The problem is to select a sub-list $I' \subseteq I$ with maximum total weight such that no pair of jobs allocated to the same machine intersect their processing intervals.

This problem is known as the job scheduling problem with identical machines (denoted by S-IM) since there is no restriction on which machine each job can be processed; otherwise it is called job scheduling problem with non-identical machines (denoted by S-NIM). In S-NIM problem, there are restrictions on the assignment of jobs to machines, i.e., for each job J_i there is a list N_i of machines on which that job cannot be scheduled. The solution of the BS-OBS problem introduced here is based on an S-IM formulation in which jobs are represented by the requests for bandwidth allocation and machines are represented by the output channels of core nodes.

4. Related work

In [10], four heuristics for the solution of the OBS scheduling problem were introduced. All the heuristics involved time complexity $O(nk\log(N))$, where n is the number of requests being processed, k the number of channels and N the number of previously allocated requests. In these heuristics, requests are modeled as interval graphs G. They are briefly described below:

Smallest Vertex Ordering (SLV): In SLV, requests are allocated in the smallest last order, i.e., the first request allocated is the one corresponding to vertex v_1. The vertices $v_1, \ldots, v_n$ of a graph G are considered to be ordered in a smallest last fashion if v_i has the smallest degree in the subgraph induced by the vertices $v_1, \ldots, v_i$. If a request cannot be allocated in any channel, it is discarded and the process is repeated for all other vertices on that ordering.

The idea behind the SLV algorithm is that if a graph has only a few vertices with large degree, then if these requests are allocated first, a small number of channels is used for them. Nonetheless, SLV can generate great loss of requests as reported in [10]. The problem with this heuristic is illustrated in Fig. 2. The requests are shown in Fig. 2a and the corresponding interval graph is presented in Fig. 2b. Vertex "A" is the one with the highest degree in the smallest last ordering, and consequently the first one to be processed. However, scheduling "A" generates the loss of all the other requests.

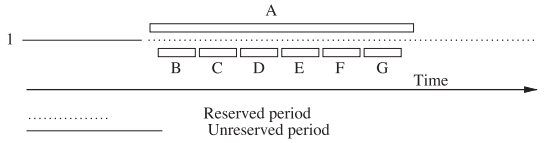
Maximal Cliques First (MCF): The MCF heuristic computes the order in which the requests are going to be processed and also the requests that will be discarded. This heuristic computes all the maximal cliques of G and then sorts them chronologically. Let $\{C_1, C_2, \ldots, C_m,\}$ be the set of maximal cliques of G ordered in a way such that $C_i \prec C_j$ for $i < j$. The algorithm processes the requests of cliques in increasing order. If the size of a clique C_j exceeds the number of channels k, then the $|C_j| - k$ requests with smallest finishing time are discarded, since necessarily $M - k$ requests are discarded if there is a clique of size M in the interval graph such that $k < M$.

This heuristic can also produce poor results. Considering the example presented in Fig. 2, when the heuristic MCF is executed, the first clique to be processed (see Fig. 2) is the clique with vertices "A" and "B". Since there is only one channel, request "A" is scheduled and all the remaining requests are discarded.

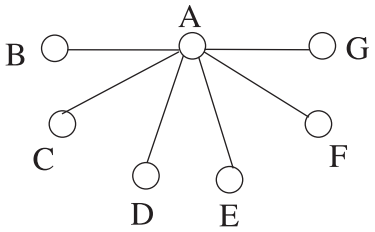
Smallest Start-time First Ordering (SSF): with this heuristic, requests are ordered and processed according to their starting time. Poor results can also be produced as illustrated by the example in Fig. 2. In this example the first request to be processed is request "A" and all other requests are discarded.

Largest Interval First Ordering (LIF): with this heuristic, requests are ordered and processed in non-increasing order of duration. Considering again the example presented in Fig. 2, it is possible to see the poor behavior of this heuristic. Once "A" is allocated, all other requests are discarded. It is interesting to note that even when the total length of the other requests is greater than the size of "A", all other requests are discarded.

These heuristics depend on the structure of the interval graphs generated; for some graphs good solutions can be



(a) *Requests over the time*



(b) Correspondent interval graph

Fig. 2. Example where heuristic in [10] is inadequate.

found, but not for others. Thus, the main problem with all of the proposed approaches is that there is no guarantee that the best solution will be achieved.

In [11], Charcranoon et al. present a heuristic that schedules the maximum number of requests, considering only one channel. This heuristic *Max Stable Set algorithm (Max-SS)* [11] is presented next. It tries to find a maximum independent set to maximize the number of disjoint requests that can be scheduled on a channel. Max-SS finds one independent set for each channel, with a time complexity of $O(n \log n)$ which is the complexity needed to find the maximum independent set in an interval graph.

5. Optimum algorithms for the BS-OBS problem

The job scheduling formulation of the BS-OBS problem relies on the list I of reservation requests as well as the list S of requests already allocated to the channels. The problem is to find a subset of requests of I with a maximum total weight that can be scheduled on the channels, since two intersecting requests cannot be scheduled on the same channel.

The heuristics in [10] were based on the formulation of the job scheduling problem with non-identical machines. Thus, if some incoming request intersects a request already scheduled to a machine m, then the second request cannot be scheduled for this machine.

In [12], it was introduced an optimal algorithm for the job scheduling problem with non-identical machines which has computational complexity of $O(n^{k+1})$. However, its computational complexity is prohibitive for the solution of the BS-OBS problem because the exponential dependency on the number of channels (k).

The approach proposed here is the formulation of the BS-OBS problem as a job scheduling with identical machines problem. For that, instead of trying to accommodate requests of I into the existing voids generated by previously allocated requests S, all requests in the set $I \cup S$ are considered for scheduling. Moreover, all previous allocated requests in S are rescheduled. In this way, the BS-OBS problem can be formulated as a S-IM problem.

Fig. 3 illustrates this idea. At a given moment, channel 1 is reserved for the time period [6,11] and channel 2, for the time period [2,7] and [11,15] then a set of incoming requests A ([1,5]), B ([8,11]), C ([12,16]) and D ([5,9]) arrived. These incoming requests are grouped with those previously allocated in a single batch, and both channels become elegible for allocation by any of the requests of this newly formed batch.

5.1. Optimum algorithm with linear time complexity

A formulation for the batch scheduling problem with linear time complexity is proposed here. The algorithm, called GreedyOPT, is suitable for networks in which all requests have the same priority. It involves two major steps. In the first, the problem is transformed into a job scheduling problem with identical machines, thus allowing the employment of fast exact algorithms.

In the second step, the algorithm in [6], proposed for job scheduling with identical machines, with time complexity

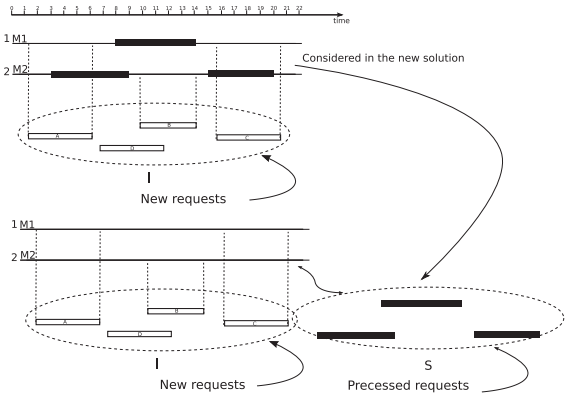


Fig. 3. Batch scheduling with identical machines.

$O(n \max\{\log(n), k\})$ is used. This algorithm process sequentially the requests of the newly formed batch. It tries to schedule the requests on one of the channels. If a given request cannot be scheduled, the algorithm tries to replace it with one of the requests already scheduled that has latest ending time. The correctness of the algorithm is given by Theorem 1.

Theorem 1. *If one of the requests in a set has its duration shortened by the anticipation of its ending time, the number of requests granted is equal to or higher than the number of requests in the original allocation for that set of requests.*

Proof. See p. 147 in [2]. □

GreedyOPT considers both the already allocated requests and the incoming new ones for the new schedule. GreedyOPT processes the new requests in chronological order to accommodate them into one of the k channels. If this is not possible, the requests already allocated but with a data burst yet to arrive can be deallocated if a new assignment decision will improve the burst loss ratio. A heap is used to sort the requests, which leads to a $O(S \log(S))$ complexity for this step. The GreedyOPT is presented in Algorithm 1.

Algorithm 1. GreedyOPT

INPUT
The set of the output channels of a node i, (k), a set of new requests (I), and a set of requests already allocated for which data bursts have not yet arrived (S).
OUTPUT
A maximum cardinality set I'.
GreedyOPT
1: $N \leftarrow |I| + |S|$
2: Sort all the requests in $I \cup S$ in chronological order of starting time
3: Add the requests sequentially into the set I'
4: If it is not feasible to allocate all the requests, remove from I' that request with latest ending time.

The GreedyOPT algorithm optimally solves the batch scheduling problem with identical machines when all requests have the same priority. This is shown in Theorem 2.

Theorem 2. *Let t_0 be the earliest time at which more than k requests in J intersect each other and $e_y = max_{j\in J} e_j$. Then, the optimal schedule excludes this yth request.*

Proof. See Proposition 1 in [6]. □

The next theorem establishes the computational complexity of GreedyOPT.

Theorem 3. *The computational complexity of the GreedyOPT algorithm is $O(N\log(N) + Nk) = O(N\max(\log(N), k))$, where k is the number of output channels of an OBS node, n the number of incoming requests, S the set of requests already allocated with the data burst yet to arrive $s = |S|$, the cardinality of the set S and $N = n + s$.*

Proof. See Appendix A. □

5.2. Optimal algorithm with polynomial time for prioritized requests

This section presents an algorithm with polynomial time complexity, entitled BATCHOPT, for networks in which requests have different priorities. As with the GreedyOPT algorithm, it is implemented in two steps. In the first step, the problem is transformed into a problem of job scheduling with identical machines problem. In the second step, an adaptation of the algorithm presented below is used to schedule the requests.

Arkin and Silverberg [12] proposed an optimal algorithm (denoted here *AS*) for the job scheduling problem with identical machines which will be described next. Our algorithm extends the *AS* algorithm to include the set *S* of requests already scheduled in the solution.

AS builds an interval graph *G* considering incoming requests and it computes all the maximal cliques of this graph; it then sorts the cliques into an increasing order for the starting time, $C_1,\ldots,C_r$. Fig. 4 exemplifies this construction.

A flow-graph G' is then constructed as follows: first create a vertex v_0 and for each clique C_j ($j = 1,\ldots,r$) create a vertex v_j, then, create directed arcs (v_j, v_{j-1}) for each C_j, with cost 0 and infinite capacity. Let *M* be the maximum size of a clique among the cliques $C_1,\ldots,C_r$. For each clique C_j, create a directed arc (v_{j-1}, v_j) with cost 0 and capacity equal to $M - |C_j|$that represents a dummy job. For each job J_i belonging to cliques $C_j,\ldots,C_{j+l}$, create a directed arc (v_{j-1}, v_{j+l}) with capacity 1 and cost w_i. This arc represents all cliques J_i belongs to (C_j to C_{j+l}). The aim is to find a flow from v_0 to v_r in G' of $(M - k)$ units and with minimum cost. In the solution to this minimum cost flow problem in G', the arcs along the flow correspond to jobs that must be discarded whereas arcs with zero flow correspond to jobs that should be scheduled.

Fig. 5 exemplifies the construction of the flow-graph G' for the interval graph *G* in Fig. 4.

Four vertexes are created, an initial vertex, v_0, and three vertexes associated with cliques of the interval graph: v_1 corresponding to C_1, v_2 corresponding to C_2 and v_3 corresponding to C_3. Then, it is created the bottom arcs (v_3, v_2), (v_2, v_1), (v_1, v_0) with cost 0 and infinite capacity. The maximum size of a clique in this example is $M = 3$ (cliques C_1 and C_3). Notice that $M = |C_1| = |C_3|$. No dummy arc was created for these cliques, since the capacity of the arcs should be $M - |C_1| = M - |C_3| = 0$. It is created only one dummy arc (v_1, v_2) for C_2, with capacity 1 and cost 0. Then it is created the arcs associated with jobs. Since job *a* belongs only to C_1, arc (v_0, v_1) with capacity 1 and cost w_a is created. Similarly, it is created arcs associated to jobs *b*, *d*, *e*, and *f*, since each one of these jobs belongs to only one clique. The exception is job *c*, that belongs to C_1, C_2 and C_3, and in this case the arc (v_0, v_3) is created, but also with capacity 1 and cost w_c. The construction of the flow graph finishes and the problem that remains is to find a minimum cost flow from v_0 to v_3 with $3 - k$ units of flow.

Theorem 4. *The algorithm AS [12] optimally solves the job scheduling problem with identical machines.*

Proof. See [12] for the proof. □

To include already scheduled jobs in the *AS* algorithm, it is necessary to have a formulation that can be applied to BS-OBS.

The set *S* of previously scheduled requests can be easily introduced in the S-IM problem by assuming that they have a weight of infinity for each request $J_i \in S$. This guarantees that the already scheduled requests will remain scheduled in the final solution. The formulation of the proposed algorithm, entitled BATCHOPT, considers both the requests to be scheduled and those already scheduled in the search for an optimal solution, as in [12]. The pseudo-code of the BATCHOPT algorithm is presented next (Algorithm 2).

Algorithm 2. BATCHOPT

Input
k channels of a node *i*, a set *I* of requests and a set *S* of previously scheduled requests that intersects with some of the requests in *I*.
Output
A subset $I' \subseteq I$ of maximum weight with a feasible schedule.
BATCHOPT
1: Set infinity weights for each request in *S*.
2: Construct an interval graph *G* representing $I \cup S$.
3: Order the maximal cliques of *G* chronologically.
4: Construct a flow graph G'.
5: Compute a flow of size $M - k$ and minimum cost.
6: Allocate all requests that correspond to arcs in G' that have a flow equal to zero.

The optimality and feasibility of the schedule produced by the algorithm can be proved using the following theorem.

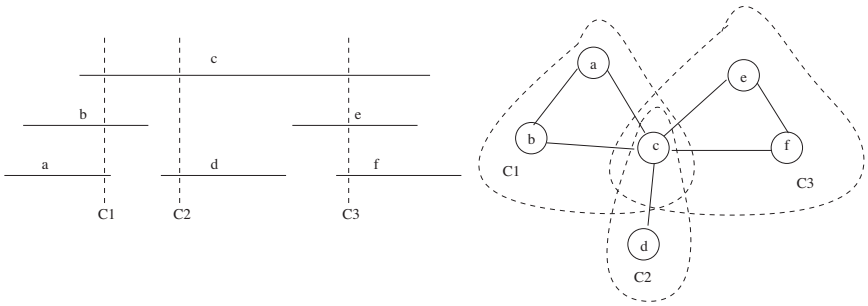


Fig. 4. Requests are on the left. On the right is the corresponding interval graph and its maximal cliques numbered according to the ordering of time.

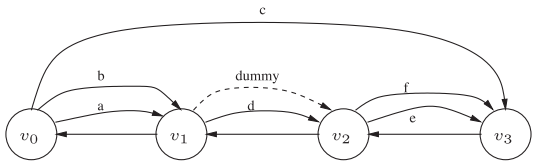


Fig. 5. An example of construction of a flow graph.

Theorem 5. *The BATCHOPT algorithm optimally solves the OBS scheduling problem. Moreover, each request already being scheduled remains scheduled on its previously scheduled channel.*

Proof. See Appendix B. □

We now establish the time complexity of the proposed algorithm.

Theorem 6. *The BATCHOPT algorithm has time complexity $O(N^2 \log(N) + N)$, where $N = |I \cup S|$, i.e., N is the number of requests in the batch plus the number of requests in S.*

Proof. See Appendix B. □

6. Adaptation of the JET protocol

In this section, a variation of the JET protocol adapted for batch scheduling is proposed.

The batch scheduling algorithms previously proposed collect requests during a time window, called acceptance window, and then employ various heuristics to schedule the requests. It was assumed that each intermediary node have a fixed acceptance window Δ used to gather requests to form a batch. However, such an approach leads to unnecessary loss of bursts, since the fact that requests can arrive at a network node with different offset times is ignored. Fig. 6 illustrates this type of loss. In this figure, two control packets arrive during the fixed size time window. Burst A arrive at time t_A and the burst B arrive at time t_B. While burst A will be processed by the batch scheduling policy and will be potentially transmitted, burst B will not be considered by the scheduling policy and it will be lost, because it arrived during the fixed time window.

To overcome this problem, we propose an extension of the JET protocol, called JET-Δ, which is employed jointly with batch scheduling algorithms. In this variation, the acceptance window is added to the offset time, as illustrated in Fig. 7. Furthermore, the instant of time that ends the period for collecting requests and begins the processing period, called the processing threshold, is determined in a way that it occurs prior to the arrival of the next arriving burst, ensuring that all requests are processed before the arrival of their burst.

In the JET protocol, the source node computes the offset time based on the estimated processing time of all intermediary nodes along the route to the destination node. The offset time computed at the source node s with destination d, along a route with intermediary nodes H is

$$T_d^s = \left(\sum_{i \in H} P_i\right) + P_d,$$

where P_i is the estimated processing time at an intermediary node i and P_d is the processing time at the destination.

When an intermediary node $i \in H$ receives a control packet, it processes the request. If the request is scheduled then it computes a new offset time decreasing its processing time, and sends this information in the control packet to the next node in the route. Let T_d^i be the offset time of the control packet received at time t_d by node i which has d as a destination. Node i computes, at time t_i, a new offset time as:

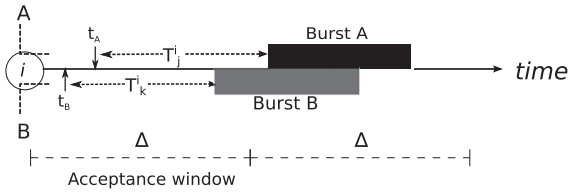


Fig. 6. Example of data burst loss due to the use of fixed size acceptance window.

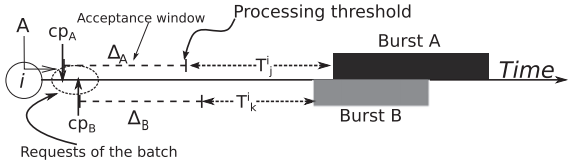


Fig. 7. Time diagram of the JET-Δ reservation protocol.

$T_d^{i+1} = T_d^i - (t_i - t_d)$.

The processing time $(t_i - t_d)$, at node i is then decreased.

A worst case estimation of the computation of the offset time at the source node is assumed, which leads to an offset time at the source node computed os:

$$T_d^s = \left(\sum_{i \in I} P_i\right) + |I|\Delta + P_d.$$

T_d^s includes the value $|I|\Delta$, which accounts for all acceptance time windows of all intermediary nodes. This offset time should be enough to avoid losses, since no node can take more time than $(P_i + \Delta)$ to process one control packet. In general, the processing time of each node is of the order of microseconds, and the acceptance window is of the order of milliseconds. In this case, the processing times are almost negligible, and an upper bound P_{max} can be established.

Since at any given OBS node there can be several control packets from distinct source–destination pairs waiting to be processed and these control packets can have different offset times, it is important to ensure that each control packet will be processed before the arrival of its corresponding burst. Thus, a processing threshold (L) should be determined by the data burst arriving first and this should be computed for every arrival of a control packet. The threshold is given by $L = (t_R + \Delta) - \delta$, where R is the $min_r\left\{t_r + T_j^i + \Delta\right\}$, δ is the batch processing time, and t_r the arrival time of control packet r with offset time T_j^i at node i on its way to node j. The request which determines the processing threshold is the request for the burst which arrives first.

It is important to note that the JET-Δ extension can be used by batch scheduling algorithms, as well as by greedy scheduling algorithms, by making $\Delta = 0$.

When the processing threshold is reached, the batch is processed. All bursts going to the same destination are grouped and the information about the new batch goes into the same control packet to decrease the overhead [8].

The inclusion of the acceptance window into the offset time can increase the end-to-end delay experienced by the packets assembled in a burst. However, this can be ameliorated if the maximum tolerable end-to-end delay is considered when computing the acceptance window as:

$$\Delta = \frac{D - T_j^s}{H}, \quad (1)$$

where D is the maximum tolerable end-to-end delay, T_j^s the offset time of a burst j at the network border and H is the number of hops from source to destination.

An alternative way of reducing the introduction of delay is to use prediction of burst size at the assembly edge node, as proposed in [15,14].

7. Numerical examples

In this section, the proposed algorithms are compared to existing algorithms, although the algorithm MAX-SS is not considered since it considers only OBS networks with a single data channel. To evaluate the performance of the algorithms, simulations were carry out using the simulator OB2S (Optical Burst Switching Simulator) [13]. The algorithms were implemented in C, using the library available in [4].

Each simulation run consisted of the allocation of 10.000 requests to the available channels. Each experiment was executed 20 times, with different seeds and the confidence interval for the mean value was computed using a confidence level of 95%.

Two distinct scenarios were considered in the simulations: in the first scenario, the experiments reported in [10] were reproduced in order to evaluate the performance of the algorithms and compare the results with those in the literature. In the second scenario, topologies of real networks with a more realistic set of parameters were employed. The results are reported in the following subsections.

7.1. Topology with one bottleneck node

Fig. 8 shows the topology used in the first scenario. In this topology, there is a single OBS node connected to four sources and a single destination node. Each input link has two wavelengths (one for data and one for control signaling). The link connecting the OBS node to the destination has five wavelengths (four for data and one for control signaling). Each wavelength has 2.5 Gbps capacity (OC-48).

As in [10], we defined the constant τ as the time required to transmit 1024 bits on one of the wavelengths, i.e., $\tau = \frac{1024}{2377728000} = 4.3e-7$ s. Bursts were generated according to a Poisson distribution with mean size, $\overline{b} = 81920$ bits. The offset time was generated according to a uniform distribution in the interval $[130\tau, 150\tau]$. The acceptance window had an arbitrary value of 100τ, which was approximately 40 μs.

Fig. 9a and b plots the blocking probability as a function of the load. For the sake of visual interpretation, the blocking probability values were plotted for different ranges of loads in these two figures. The lowest blocking probability is produced by the BATCHOPT algorithm, followed by the GreedyOPT algorithm. Although both BATCHOPT and GreedyOPT optimally schedule requests for a given batch, only the BATCHOPT guarantees that requests already scheduled in previous batches remain scheduled in the current computation. As a consequence, only the new requests which do not overlap with already scheduled

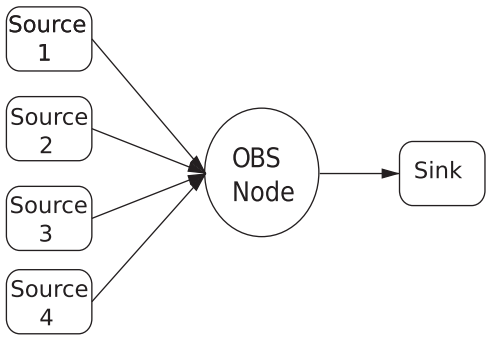
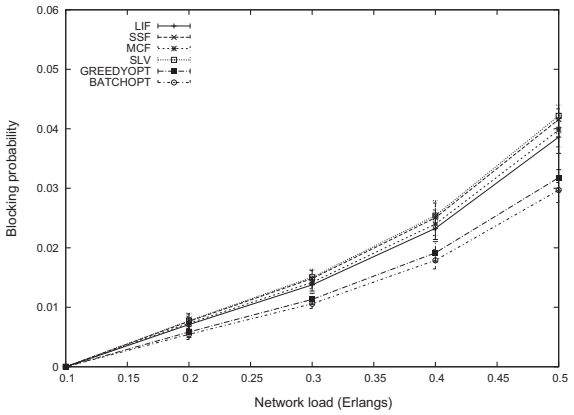
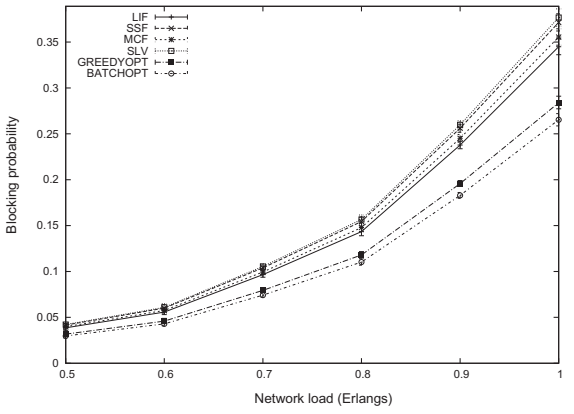


Fig. 8. Topology used in the first simulation scenario.



(a) Load varying from 0.1 to 0.5 Erlangs.



(b) Load varying from 0.5 to 1 Erlangs.

Fig. 9. Blocking probability for each algorithm.

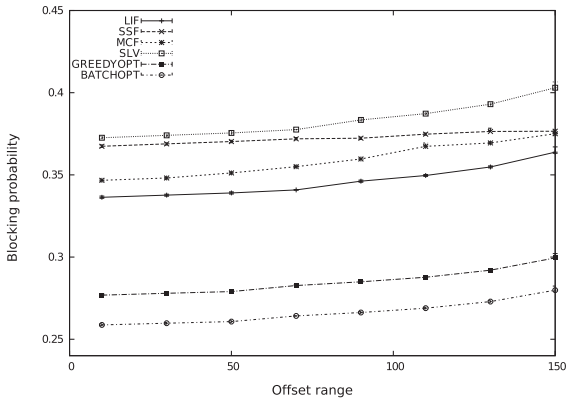


Fig. 10. Blocking probability as a function of the difference in offset time.

requests are added to the final solution. With GreedyOPT algorithm, requests already scheduled in previous batches may be dropped during the computation of future batches, thus increasing the blocking probability.

Another experiment conducted using this simulation scenario aimed to evaluate the effect of the offset range in the probability of blocking. The offset range is defined as the difference between the highest and lowest value of

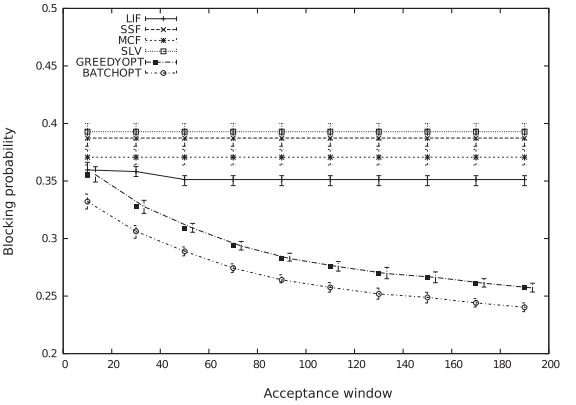


Fig. 11. Blocking probability as a function of the size of the acceptance window.

the offset time, $T_{max} - T_{min}$. The value of T_{max} was fixed at 200τ and the value of T_{min} was gradually increased. In this experiment, the network load was of 99% of the link capacity.

Fig. 10 shows that, as observed in the experiments reported in [10], there is an increase in blocking probability as the offset range is increased due to the retro-blocking phenomenon of the JET signaling protocol, in which a reservation request can be blocked by another reservation starting after it.

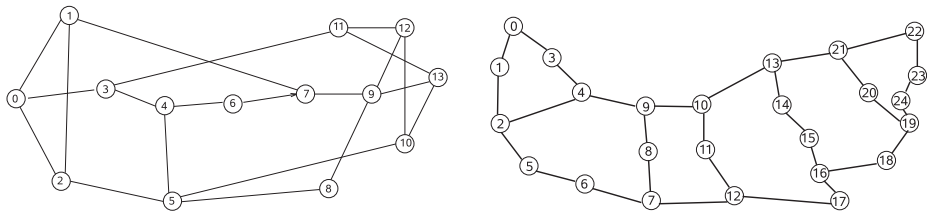
Fig. 11 shows the blocking probability as a function of the size of the acceptance window. In this experiment, as in [10], the difference between the highest and lowest offset time is adjusted to 50τ and the network load to 50τ with the acceptance window varying from 10τ to 190τ.

It can be seen that, except for the BATCHOPT and GreedyOPT algorithms, the blocking probabilities produced do not depend on the increasing size of the acceptance window. In fact, the success of allocation by heuristics depends on the pattern of the starting and ending times of the requests rather than the number of requests.

Under BATCHOPT and GreedyOPT, the blocking probability decreases as the acceptance window increases, since increasing the acceptance window also increases the number of requests composing each batch. The optimal algorithms schedule the maximum number of requests in each batch resulting in lower blocking probabilities than those given by heuristics.

7.2. Simulations using real network topologies

In these experiments, the simulations used real topology networks such as NSFNet and Abilene, shown in Fig. 12. Each link represents a fiber with 32 wavelengths with 2.5 Gbps of capacity. The processing time for a control packet and the time required to configure the switching fabric was 50 μs. Each node at the edge can be either source or destination; for each request, source and destination were drawn from a uniform distribution. Traffic was generated according to a Poisson process, and the burst size followed a negative exponential distribution.



(a) *Backbone* NSFNet (b) *Backbone* Abilene

Fig. 12. Real topologies used in the simulations.

7.2.1. Comparison of grouping strategies

As previously discussed, the dimensioning of the acceptance window is challenging in complex topologies with several source–destination pairs. The scheduling algorithm used in these simulations was the BATCHOPT since it produced the best results in the evaluation reported in Section 7.1. This algorithm was employed with JET-Δ strategy, as well as with the FIXED strategy proposed in [10].

Fig. 13 displays the blocking probability as a function of the size of the acceptance window. While the acceptance window is smaller than the smallest offset time, the blocking probability produced by BATCHOPT is constant, regardless of the grouping strategy adopted. When the acceptance window becomes larger than the smallest offset time, the performance of FIXED deteriorates. Since the acceptance window is larger than the smallest possible offset time, the bursts corresponding to the requests in the batch arrive at the network nodes even before their requests have been processed, causing burst loss.

7.2.2. Comparison of the batch scheduling algorithms

In this section, the two algorithms are evaluated using topologies of real network. In these experiments, JET-Δ was employed as grouping strategy.

First, the impact of the acceptance window size (Δ) was evaluated. Each request had a unit cost and the network load was 1000 Erlangs. Fig. 14 shows the blocking probability as a function of Δ and Fig. 15 shows the mean number of requests in each batch as a function of Δ. The heuristics LIF, MCF, SSF and SLV were not sensitive to changes in the acceptance window size due to the fixed order in which the requests were processed, since the acceptance of requests depends more on the adjacency structure of the interval graph representing the requests than on the number of requests processed.

Both, BATCHOPT and GreedyOPT can benefit from an increase in the mean number of requests in each batch. The larger the acceptance window for requests, the better is the performance. This is due to the fact that when more requests are processed at one time, more information about the intersections between requests is available. The algorithms can then make the best choice for scheduling the requests in the batch.

In practice, the window size should be adjusted as a function of the timing requirements of the traffic, given by Eq. (1). In order to avoid impacting the performance of applications with time constraints, Δ was set to 1*ms* [3,7].

Fig. 16 reveals the blocking probability as a function of network load in a scenario where all requests have the same priority. Increasing the network load also increases the blocking probability. With the increase in network load, the size of maximal cliques of the interval graph associated with the requests increases, which explains the growth in the blocking probability. This is a common behavior in all algorithms evaluated. However, the increase in blocking probability is less when the BATCHOPT algorithm is used. Compared to the SLV algorithm (the algorithm that produced the worst results), the BATCHOPT algorithm produced gains of 42% in blocking probability. Since all requests had the same priority, the algorithm always select the set of requests of maximum cardinality

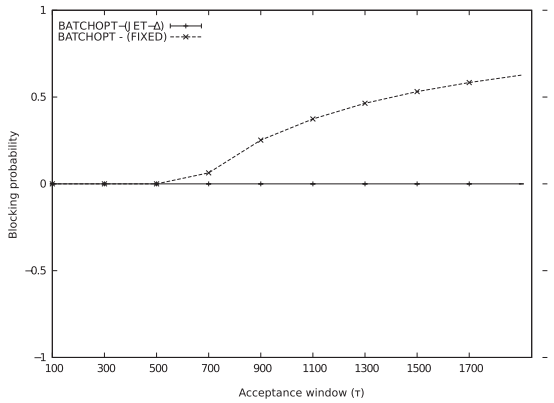


Fig. 13. Blocking probability for the strategies of JET-Δ and FIXED as a function of acceptance window size.

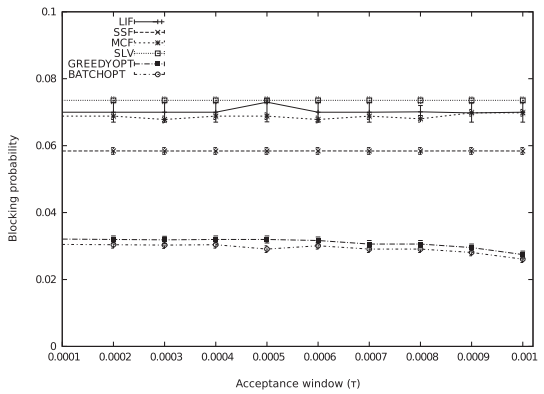


Fig. 14. Blocking probability as a function of the window acceptance size.

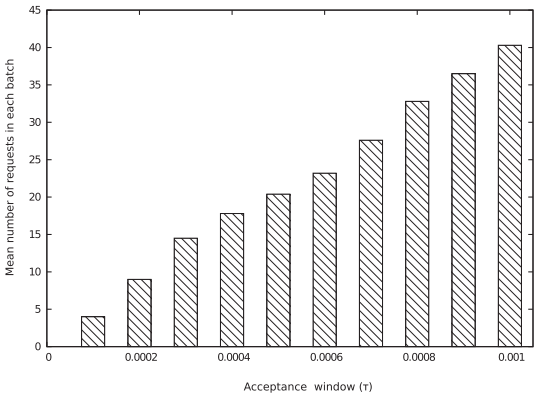


Fig. 15. Average number of requests per batch as a function of acceptance window size.

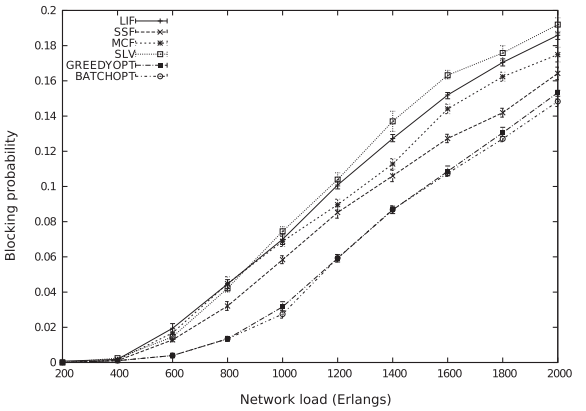


Fig. 17. Blocking probability as a function of network load (considering QoS).

and, thus, discards those with least number of request. The gains in the blocking probability of LIF, MCF and SSF in relation to SLV are 12%, 8% and 2%, respectively.

The algorithm that produced the second lowest blocking probability was GreedyOPT, with a gain of 35% when compared to SLV. Although the GreedyOPT algorithm is optimum it does not guarantee that already scheduled requests remain scheduled, i.e. a requests already scheduled can be blocked by those being processed in the current batch.

Simulations considering requests with different priorities (Quality of Service) were also conducted. In these simulations, five classes were considered with the class associated with a request being randomly picked from a uniform distribution. The weight of the classes involved follow the pattern: $w_1 = 1$, $w_2 = 2$, $w_3 = 4$, $w_4 = 8$, $w_5 = 16$, where w_i is the weight of class i.

Fig. 17 shows the blocking probability resulted from the algorithms when the requests have different priorities. Again, the algorithm that produced the lowest blocking probability was BATCHOPT algorithm followed by GreedyOPT algorithm. Moreover, the difference in the blocking probability among all evaluated algorithms is lower when compared to Fig. 16. This happens because BATCHOPT and GreedyOPT try to maximize the sum of weights of the requests in the batch. Thus, while trying to do this, a higher number of requests with low weight can be discarded by those algorithms.

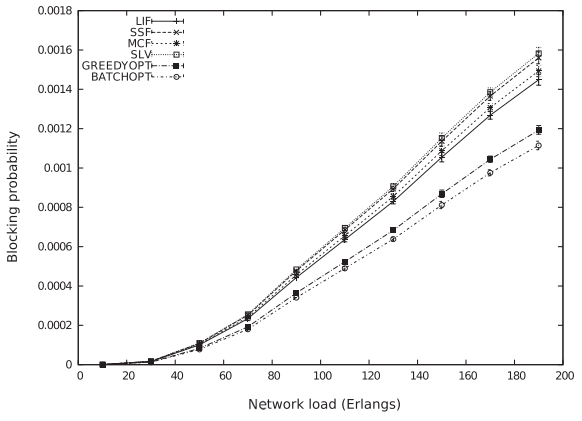


Fig. 16. Blocking probability as a function of the network load.

Fig. 18 shows the distribution of loss per class. Note that BATCHOPT provides the lowest blocking probabilities for the classes with higher priorities, concentrating most of the losses in those with lowest priority. In other words, BATCHOPT was capable of providing differentiated services for distinct classes of service. This does not happen with the other policies, which leads to an "almost uniform" distribution of losses.

The time for execution of the algorithms was also assessed. Simulations were run on a Intel Pentium Core 2 Duo machine with 2.8 Ghz clock, 4 GB RAM, and OpenSuse 11.1 operating system. The execution time was measured using the time command. Five sets of 500 requests were randomly generated. Ten simulations were executed for each set with the mean execution time for each set computed as well as the overall mean execution time. Gain was defined as the percentage difference in relation to the lowest execution time. They are displayed in Table 1. The slowest algorithm is the SLV algorithm while the SSF is the fastest. GreedyOPT was at most 6% slower than SSF. On an average it was 32% faster than LIF algorithm and 3% slower than the SSF. BATCHOPT was at most 10% slower than SSF. On an average it was 28% faster than LIF algorithm and 7% slower than SSF.

8. Conclusions and future work

This paper has introduced two optimum algorithms for the batch scheduling problem in OBS networks, they are based on a formulation of the problem of scheduling jobs with identical machines. Such a formulation considers not only the unprocessed requests but also those scheduled in the problem formulation. Although GreedyOPT has linear computational complexity, BATCHOPT produces the lowest blocking probability ratio. The blocking ratio given by these two algorithms are lower than those given by

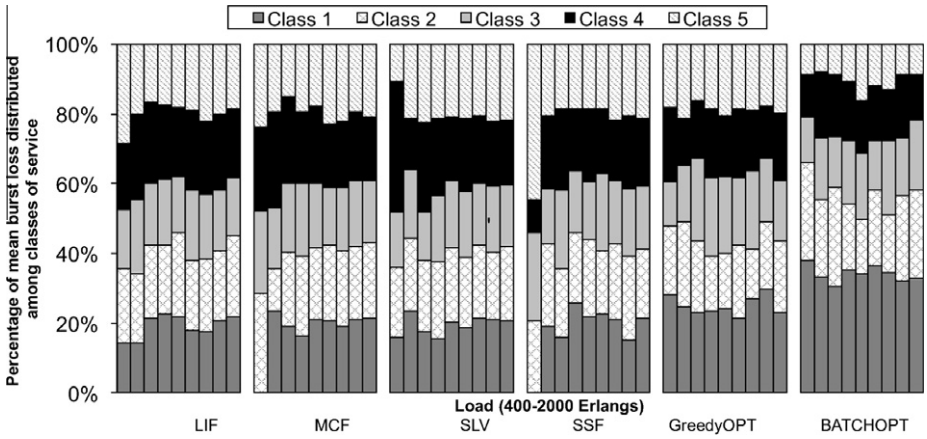


Fig. 18. Blocking probability per class as a function of load.

Table 1
Relative gain in execution time.

	Batch 1	Batch 2	Batch 3	Batch 4	Batch 5	Mean
Reference value	7.2×10^{-2}	8.0×10^{-2}	6.8×10^{-2}	7.6×10^{-2}	7.2×10^{-2}	7.3×10^{-2}
LIF	37%	27%	22%	40%	33%	32%
SSF	41%	30%	25%	42%	37%	35%
MCF	33%	26%	26%	34%	25%	28%
SLV	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
GreedyOPT	36%	27%	25%	40%	31%	32%
BATCHOPT	33%	25%	23%	36%	27%	28%

other existing heuristics. The difference between the proposed algorithms is that one (BATCHOPT) maintains the original scheduling of the already scheduled requests while the other (GreedyOPT) does not. Results derived via simulation show that this proposed BATCHOPT scheduling algorithm outperforms those previously proposed ones. BATCHOPT algorithm produces blocking probability 42% lower than that SLV, the algorithm with the highest blocking probability, and 30% lower than LIF, the third lowest blocking probability. Moreover, the BACTHOPT algorithm is only 7% slower than the SSF algorithm, which is the fastest heuristic. Besides that, BATCHOPT penalizes less the high priority classes in prioritized networks than do other existing heuristics. Furthermore, this paper has extended the JET reservation protocol for the consideration the scheduling of bursts although it remains compatible to the original JET.

The use of BATCHOPT is then recommended for networks which support prioritized traffic. However, if the execution time is a critical requisite and the network does not support prioritized traffic, GreedyOPT would be the best option.

As future work, it might be interesting to investigate the impact of different batch creation strategies as well as the impact of acceptance window size for each class of service, on the network performance. Another interesting point is to investigate how to use the proposed optimal batch scheduling algorithms jointly with proportional QoS policies in order to obtain an integrated solution for end-to-end proportional QoS.

Appendix A

This appendix presents proofs related for the GreedyOPT algorithm.

Theorem 3. *Let k be the number of output channels of an OBS node, n the number of incoming requests, S the set of requests already allocated but with data bursts yet to arrive, and $s = |S|$, the cardinality of the set S. The computational complexity of the GreedyOPT algorithm is $O(N\log(N) + Nk) = O(N\max(\log(N), k))$, where $N = n + s$.*

Proof. It takes $O(s\log(s))$ to determine which unprocessed requests should be organized in a heap. It takes $O(N\log(N))$ to order the requests, as well as $O(Nk)$ to verify the availability of a channel to accommodate the new request (line 4 of the GreedyOPT). Therefore, the complexity of GreedyOPT is $O(s\log(s) + N\log(N) + Nk)$. Making $s = N$ in $s\log(s) \leqslant N\log(N)$, the complexity of GreedyOPT is thus $O(2N\log(N) + Nk) = O(N\log(N) + Nk) = O(N(\log(N) + k))$. Considering the maximum value of the $(\log(N) + k)$, we have $O(N\max(\log(N), k))$. □

Appendix B

This appendix provides proofs related to the BATCHOPT algorithm.

Theorem 5. *The BATCHOPT algorithm optimally solves the OBS scheduling problem. Moreover, each already scheduled request remains scheduled on the scheduled channel.*

Proof. Since each request in S has a weight equal to infinity, and the algorithm computes a minimum cost flow, the algorithm will not use any of the arcs corresponding to requests in S. Therefore, the requests in S will necessarily remain scheduled. Moreover, from the result of Theorem 4 it is known that among the requests in I, the algorithm will schedule a subset I' of requests of maximum weight.

Indeed, it is not necessary to change the channels for it is assigned to requests that have already been scheduled. Suppose the algorithm generates the schedule starting at some time t. Let $S' \subseteq S$ be the set of requests with starting time prior to t. This means that each request in S' is already scheduled, and the scheduled channel cannot be changed. But notice that for the requests in $S \backslash S'$ it is indeed possible to change their channel without problems. The BATCHOPT algorithm selects a set I' of new requests such that for any time $t' \geqslant t$ there are at most k requests from $I' \cup S$ that will be transmitted at time t'. The requests in $I' \cup S$ can be sorted by their starting times and scheduled in this order on an available channel. This generates a feasible schedule since at any time t' there are at most k requests intersecting it, and will thus be there an available channel each time a request is assigned. Notice that the requests in S were previously scheduled in a feasible schedule. Since the requests in S' have the earliest starting times among the requests in $S \cup I'$, they are processed first so that each request in S' can be scheduled on the previously scheduled channel. □

Theorem 6. *The BATCHOPT algorithm has time complexity $O(N^2 \log(N) + N)$, where $N = |I \cup S|$, i.e., N is the number of requests in the batch plus the number of requests in S.*

Proof. Let n be the size of the incoming batch ($n = |I|$) and let and s be the size of previously scheduled requests that intersects with I ($s = |S|$).

The time complexity to find the set S is $O(s \log s)$ if previously scheduled requests are stored using a heap. The AS algorithm has time complexity $O(N^2 \log N)$ (see [12]), and since $N = s + n$ the overall complexity to find S and to schedule $I \cup S$ is $O(N^2 \log N)$. □

References

[1] C. Qiao, M. Yoo, Choices, features and issues in optical burst swirching (OBS), Optical Network Magazine 1 (2000) 36–44.
[2] G.B. Figueiredo, Control Mechanisms for Optical Burst Switched Networks, PhD Thesis, Institute of Computing, University of Campinas, 2009.
[3] Network QoS Needs of Advanced Internet Applications: A Survey, Internet2 QoS Working Group, 2001. <http://qos.internet2.edu/wg/apps/fellowship/Docs/Internet2AppsQoSNeeds.pdf>.
[4] Robert. Sedgewick, Algorithms in C, Part 5: Graph Algorithms, Addison-Wesley Professional, 2001.
[5] Stephan Olariu, An optimal greedy heuristic to color interval graphs, Information Processing Letters 37 (1) (1991) 21–25.
[6] Khalid I. Bouzina, Hamilton Emmons, Interval scheduling on identical machines, Journal of Global Optimization 9 (3-4) (1996) 379–393.
[7] Joel J.P.C. Rodrigues, Mario M. Freire, Pascal Lorenz, Impact of setup message processing and optical switch configuration times on the performance of IP over optical burst switching networks, Lecture Notes in Computer Science 3733 (20) (2005) 264–273.
[8] M. Elhaddad, R. Melhem, T. Znati, D. Basak, Traffic shaping and scheduling for OBS-based IP/WDM backbones, IEEE Opticom 5285 (2003) 336–345.
[9] Donald J. Rose, Robert Endre Tarjan, George S. Leuker, Algorithmic aspects of vertex eliminations on graphs, SIAM Journal on Computing 5 (1976) 266–283.
[10] A. Kaheel, H. Alnuweiri, Batch scheduling algorithms for optical burst switching networks, Lecture Notes in Computer Science 3462 (2005) 90–101.
[11] S. Charcranoon, T.S. El-Bawab, H.C. Cankaya, J.D. Shin, Group scheduling for optical burst switched (OBS) networks, IEEE Globecom (2003) 2745–2749.
[12] E.M. Arkin, E.B. Silverberg, Scheduling jobs with fixed start and end times, Discrete Applied Mathematics 18 (1987) 1–8.
[13] J. Maranhao, A. Soares, W.F. Giozza, A study on architectures of OBS networks, in: XXV Brazilian Symposium on Computer Networks, 2007, pp. 133–146.
[14] J. Liu, N. Ansari, T.J. Ott, FRR for latency reduction and QoS provisioning in OBS networks, Journal on Selected Areas in Communications 21 (2003) 1210–1219.
[15] D. Morato, J. Aracil, L.A. Diez, On linear prediction of internet traffic for packet and burst switchin networks, IEEE ICC (2001) 138–143.
[16] J. Li, C. Qiao, Schedule bursts proactively for optical burst switched networks, Computer Networks 44 (2004) 617–629.
[17] X. Wang, H. Morikawa, T. Aoyama, Priority-based wavelength assignment algorithm for optical burst switched photonic networks, in: Optical Fiber Communications Conference, 2002, pp. 765–766.
[18] J. Chang, C. Park, Efficient channel scheduling algorithm in optical burst switching architecture, IEEE Workshop on High Performance Switching and Routing, 2002, pp. 194–198.
[19] V.M. Vokkarane, J.P. Jue, Prioritized burst segmentation and composite burst-assembly techniques for QoS support in optical burst-switched networks, IEEE Journal on Selected Areas in Communications 21 (2003) 1198–1209.
[20] J. Xu, C. Qiao, J. Li, G. Xu, Efficient channel scheduling algorithms in optical burst switched networks, IEEE INFOCOM 3 (2003) 2268–2278.
[21] J. Xu, C. Qiao, J. li, G. Xu, Efficient burst scheduling algorithms in optical burst-switched networks using geometric techniques, IEEE Journal on Selected Areas in Communications 22 (2004) 1796–1811.
[22] J. Li, C. Qiao, Y. Chen, Recent progress in the scheduling algorithms in optical-burst-switched networks, OSA Journal of Optical Networking 3 (2004) 229–241.
[23] Y. Chen, C. Qiao, Y. Xiang, Optical burst switching (OBS): a new area in optical networking research, IEEE Network 18 (2004) 16–23.
[24] S. Oh, M. Kang, A burst assembly algorithm in optical burst switching networks, in: Optical Fiber Communications Conference, 2002, pp. 771–773.
[25] K. Dolzer, Assured Horizon – A New Combined Framework for Burst Assembly and Reservation in Optical Burst Switched Networks, NOC, 2002.
[26] An Ge, Franco Callegati, Lakshman S. Tamil, On optical burst switching and self-similar traffic, IEEE Communications Letters 4 (2000) 98–100.
[27] Xiaojun Cao, Jikai Li, Yang Chen, Chumming Qiao, Assembling TCP/IP packets in optical burst switched networks, IEEE GLOBECOM (2002) 2808–2812.
[28] Xiang Yu, Jikai Li, Xiaojun Cao, Yang Chen, Chumming Qiao, Traffic statistics and performance evaluation in optical burst switched networks, Journal of Lightwave Technology 22 (12) (2004) 2722–2738.
[29] J. Turner, Terabit burst switching, Journal of High Speed Networking (1999) 3–16.
[30] Y. Xiong, M. Vandenhoute, C. Cankaya, Design and analysis of optical burst-switched networks, in: SPIE'99 Conf. All Optical Networking: Architecture, Control and Management Issues, vol. 3843, 1999, pp. 112–119.
[31] Y. Xiong, M. Vandenhoute, C. Cankaya, Control architecture in optical burst-switched WDM networks, IEEE Journal of Selected Areas on Communications (2000) 1838–1851.
[32] D. West, Introduction to Graph Theory, Prentice Hall, Inc., Upper Saddle River, NJ, 1996.

Gustavo B. Figueiredo received his B.Sc. degree in Computer Science from Salvador University (2001), and the M.Sc. (2003) and PhD (2009) degrees in Computer Science from the State University of Campinas. Since 2010, he has been affiliated with the Department of Computer Science of the Federal University of Bahia, Bahia – Brazil, where is currently an Associate Professor. His main research interest includes problems involving Network Performance Evaluation, Planning, Dimensioning and Optimization of Optical Burst Switched Networks.

Eduardo Candido Xavier obtained a Bachelor degree in Computer Science from the Federal University of Parana, the MSc. and Ph.D. (2006) from the State University of Campinas. He worked at the University of São Paulo (2007–2008) and then joined the State University of Campinas in 2008. He was a postdoc at the State University of New York at Stony Brook (2009–2010). His areas of interest are algorithms, and combinatorial optimization.

Nelson L.S. da Fonseca received his Electrical Engineer (1984) and M.Sc. in Computer Science (1987) degrees from The Pontificial Catholic University at Rio de Janeiro, Brazil, and the M.Sc. (1993) and Ph.D. (1994) degrees in Computer Engineering from The University of Southern California, USA. Since 1995, he has been affiliated with the Institute of Computing of The State University of Campinas, Campinas – Brazil where is currently a Full Professor. He published over 250 refereed papers and supervised over 50 graduate students. He served as Editor-in-Chief of the IEEE Communications Surveys and Tutorials and Editor-in-Chief of the IEEE Communications Society Electronic Newsletter and Editor of the Global Communications Newsletter. He is a member of the editorial board of: Computer Networks, IEEE Communications Surveys and Tutorials, IEEE Communications Magazine, Peer-to-Peer Networking and Applications, International Journal of Communication Systems and Journal of Internet Service and Applications. He served on the editorial board of the IEEE Transactions on Multimedia, Brazilian Journal of Computer Science, and on the board of the Brazilian Journal on Telecommunications. He is the recipient of Elsevier Computer Networks Editor of the Year 2001. He served as ComSoc Director for Latin America. He served as ComSoc Director of On-line Services and served as technical chair for over 15 ComSoc symposia and workshops.