Scheduling in Buffered WDM Packet Switching Networks with Arbitrary Wavelength Conversion Capability

Zhenghao Zhang and Yuanyuan Yang
Department of Electrical & Computer Engineering
State University of New York, Stony Brook, NY 11794, USA

Abstract—**Optical networking is a promising candidate for high-speed communication networks because of its huge bandwidth. In this paper we study optimal scheduling in buffered WDM packet switching networks with arbitrary wavelength conversion ability. We focus on limited range wavelength conversion while considering full range wavelength conversion as a special case of it. We formalize the problem of maximizing network throughput and minimizing total delay in such a network as finding an optimal matching in a weighted bipartite graph. We then give a simple and fast algorithm called the Scan and Swap Algorithm that solves the problem in $O(kB^2)$ time, where k is the number of wavelengths per fiber and B is the buffer length, as compared to other existing algorithms that need at least $O(k^2B^2 + k^2BN)$ time where N is the number of input fibers.**

Index Terms—**Wavelength-division-multiplexing (WDM), packet scheduling, wavelength conversion, limited range wavelength conversion, optical packet switching, optical switches, optical switching networks, optical buffering, packet loss probability,**

I. Introduction and Background

All optical networking has been proposed as a promising candidate for high-speed communication networks [13], [8], [12] because of the huge bandwidth of optics: a single fiber has a bandwidth of nearly 50 THz [16]. To fully utilize the bandwidth, a fiber is further divided into a number of independent channels, with each channel on a different wavelength. This is referred to as *wavelength-division-multiplexing(WDM)*.

Several different technologies have been developed for transmitting data over WDM [13], such as broadcast-and-select, wavelength routing, optical packet switching, and optical burst switching. Broadcast-and-select networks and wavelength routing networks have been extensively studied. Optical packet switching and burst switching, especially optical packet switching, though still in research phase, are attracting more and more interests as it may offer better flexibility and better exploitations of the bandwidth [13]. In this paper, we will focus on WDM packet switching networks.

In a WDM optical packet switched network, data packet is modulated on a specific wavelength and may travel several hops before reaching the destination. In each hop, a switching network (or simply a switch) is used to direct the incoming packet to the correct output fiber link. There can be many input fiber links and output fiber links in a switching network and on each link there can be multiple wavelength channels. Output contention occurs when some packets on the same wavelength are destined for one output fiber. In general, there are three ways to combat output contention: deflection routing, buffering and exploiting the wavelength domain [13]. Deflection routing is to send the contending packet to some other output link (which may or may not have a route to the destination). By doing so the packet is not dropped. However, the end-to-end delay may be long and the packets arrived at the destination may not be in a correct order. In this paper we study the combination of the second and the third methods: buffering and exploiting the wavelength domain. By buffering, the contending packet will be delayed by fiber delay lines until it can be transmitted. Exploiting the wavelength domain means if there are more than one optical packets on the same wavelength destined for an output fiber, the wavelengths of some of these packets are converted to idle wavelengths (if there are any) on the output fiber, such that the packets can still be transmitted. Buffering in optical domain needs fiber delay lines and will be costly and bulky. However, it has been shown in [18] that by only exploiting the wavelength domain, the arrival rate to a network cannot be very high if the packet loss probability needs to be controlled under the generally accepted rate 10^{-10} for a typical switching network. Thus, some buffers may have to be used in a network with heavy traffic.

To translate a signal on one wavelength to another *wavelength converters* are needed. If a wavelength converter is capable of converting a wavelength to any other wavelength in the optical system, it is called *full range wavelength converter*. However, this type of wavelength conversion is quite difficult and expensive to implement due to technological limitations [14] [11]. A realistic all-optical wavelength converter may only be able to convert a given wavelength to a limited number of wavelengths, which is called *limited range wavelength converter*. It was shown [14], [11], [15] that limited range wavelength converters can achieve network performance similar to full range wavelength converters even when the conversion degree is very small. Thus, limited range converters are considered as a practical, cost-effective choice for providing wavelength conversion ability in WDM networks, which will be the main focus of this paper. It should be

This research was supported by the U.S. National Science Foundation under grant numbers CCR-0073085 and CCR-0207999.

mentioned that full range wavelength converters are also considered in this paper and will be treated as a special case of limited range wavelength converters.

We assume the duration of an optical packet arrived at the switching network is one time slot, as in [13] [8] [12]. The traffic pattern considered in this paper is unicast, i.e., each packet is destined for only one output fiber. The packet does not specify which wavelength channel on the destination fiber it should be connected to, and we can assign to it any free wavelength channel accessible to this packet. We will study scheduling algorithms in such WDM switching networks. Note that if the wavelength conversion in the WDM switching networks is full range, the scheduling is trivial. This is because that in this case we do not need to be concerned with the wavelength of the packets, since full range wavelength converters are capable of converting any incoming wavelength to any outgoing wavelength. When the wavelength conversion is limited ranged, the scheduling becomes complicated. We solved the problem of maximizing network throughput in an unbuffered WDM switching network with limited range wavelength conversion in [17]. In this paper we study the buffered case. We show that the problem of maximizing network throughput and minimizing total delay in such a network can be formalized as finding an optimal matching in a weighted bipartite graph called *request graph* and present an efficient algorithm called the *Scan and Swap Algorithm* that solves this problem in $O(kB^2)$ time, where k is the number of wavelengths per fiber and B is the buffer length.

In the past, buffered WDM switching networks with full range wavelength converters were studied and their performance were evaluated with analytical models [8] [19]. Buffered WDM switching networks with limited range wavelength conversion were studied in [12], in which the authors suggested to "store a packet in the output buffer that has smallest number of packets" when contention arises. However, no proof was given as to whether this will either achieve maximum throughput or minimum total delay. In this paper we consider the same network model as in [12] but will prove that our algorithm gives an optimal scheduling that both maximizes network throughput and minimizes total delay.

The rest of this paper is organized as follows. Section II describes the assumptions on limited range wavelength conversion and the network model. Section III gives the formalization of the problem. Section IV briefly reviews the algorithms for maximum matchings in request graphs. Section V presents the Scan and Swap Algorithm for finding optimal matchings in request graphs. Section VI presents the simulation results. Finally, Section VII concludes the paper.

II. Preliminaries

A. Wavelength Conversion

As mentioned earlier, with limited range wavelength conversion, an incoming wavelength may be converted to a set of adjacent outgoing wavelengths. We define the set of these outgoing wavelengths as the *adjacency set* of this input wavelength. The cardinality of the adjacency set is the *conversion degree* of this wavelength.

The wavelength conversion considered in this paper can be called "ordered interval" because the adjacency set of any wavelength can be represented by an interval of integers, and intervals for different wavelengths are "ordered". To be specific, we have the following assumptions:

Assumption 1: The wavelengths in the adjacency set can be represented by an interval of integers. The adjacency set of an input wavelength λ_i is denoted by $[begin(i), end(i)]$, where $begin(i)$ and $end(i)$ are positive integers and $begin(i) \leq end(i)$.

Assumption 2: For two wavelengths λ_i and λ_j, if $i < j$, then $begin(i) \leq begin(j)$ and $end(i) \leq end(j)$.

The first assumption says that if a wavelength can be converted to λ_{i-1} and λ_{i+1} by a wavelength converter, then it can also be converted to λ_i by this wavelength converter. The second assumption says that if the wavelength converters are capable of converting, say, λ_1 to λ_5, then they should also be capable of converting λ_2 to λ_5, as λ_2 is "closer" to λ_5 than λ_1.

We can use a bipartite graph to visualize the wavelength conversion: the left side vertices represent input wavelengths and the right side vertices represent output wavelengths. If input wavelength λ_i can be converted to output wavelength λ_j, there is an edge between them. Fig. 1 shows a conversion graph for $k = 6$, e.g. the adjacency set of λ_1 can be represented as $[1, 2]$.

Note that under this type of wavelength conversion different wavelengths may have different conversion degrees. For example, in Fig. 1 the conversion degree of λ_1 is 2 but the conversion degree of λ_2 is 3. Thus, we introduce the *conversion distance* defined as the largest difference between the index of a wavelength and a wavelength that it can be converted to. In Fig. 1 the conversion distance for λ_1 is $2 - 1 = 1$. In fact, the conversion distance is 1 for all the wavelengths in this example. It should be mentioned that in our scheduling algorithms there is no need to assume that the conversion distances are the same for all wavelengths, as in real WDM networks wavelengths may have different conversion distances. The two assumptions we made are very general and can be applied to a wide variety of wavelength convertible WDM networks, and even for full range wavelength conversion, which can simply be considered as letting the conversion degrees for all wavelengths be k.

B. Network Model

The WDM switching network we consider is shown in Fig. 2. It is an $N \times N$ switch, i.e., there are N fibers on the input side of the switch and N fibers on the output side of the switch. On each fiber there are k wavelengths that carry independent data. There are limited range wavelength converters equipped for each input channel at the input side of the switch. There are optical buffers for each output fiber at the

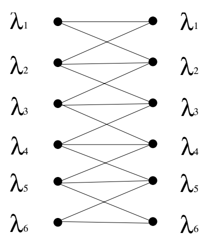


Fig. 1. Conversion graph on an optical fiber with 6 wavelengths.

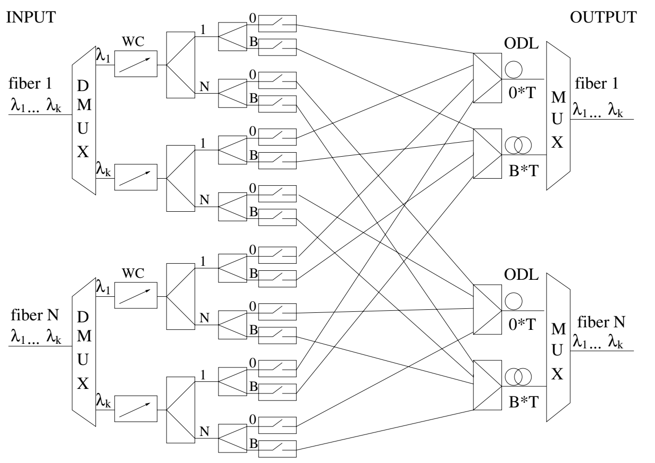


Fig. 2. A buffered wavelength convertible WDM optical switching network.

output side of the switch. As in most proposed WDM switching networks, we assume the optical buffers are implemented with Optical Fiber Delay Lines (ODL). A light signal that enters an ODL will come out of the ODL after a certain delay time proportional to the length of the ODL [13]. There are $B + 1$ ODLs for each output fiber, capable of delaying the optical packet for $0, 1, 2, \ldots, B$ time slots, respectively. As can be seen from the figure, the output of these $B + 1$ buffers are directly combined together and sent to the output fiber.

The switching fabric is capable of connecting any input wavelength channel to any of the $B + 1$ ODLs for any output fiber. With limited range wavelength conversion, an input wavelength channel may be assigned to any one of the wavelengths it can be converted to (in its adjacent set) on an ODL.

III. Problem formalization

In this section we show how the problem of maximizing network throughout and minimizing total delay in the WDM switching network can be formalized into a weighted bipartite graph matching problem.

First consider the optical packets arrived at the WDM switching network at a time slot. We can partition them into N subsets according to their destination fibers. The decision of accepting or rejecting a connection request in one subset does not affect the decisions for other subsets. Hence the scheduling for one output fiber can be done independently of other output fibers. The input to the scheduling algorithm is the packets destined to this fiber at this time slot. The output of the algorithm is the decision of whether a request is granted, and if granted, which wavelength on which ODL it is assigned to.

As in [13] [8], we assume that there are k wavelengths in an ODL, similar to input and output fibers. Since the packet is one time slot long, an ODL of length b can accommodate up to bk packets, k for each time slot. However, there may not be so many packets stored in the ODLs since there is another constraint. From Fig. 2 we can see that the outputs of the $B+1$ ODLs are directly combined into one signal and sent to the output fiber. Hence, it is required that no two packets on the same wavelength should come out of these ODLs at the same time slot, otherwise collision will occur. For example, at the beginning of a time slot, if there is a packet coming out of ODL 1 on wavelength λ_1, we cannot assign channel λ_1 in ODL 0 to any newly arrived packets, neither should there be a packet on λ_1 coming out of any other ODLs at this time slot.

More precisely, let $\{\lambda_1^0, \lambda_2^0, \ldots, \lambda_k^0\}$, $\{\lambda_1^1, \lambda_2^1, \ldots, \lambda_k^1\}$ ... $\{\lambda_1^B, \lambda_2^B, \ldots, \lambda_k^B\}$ denote the wavelength channels on ODL 0 to B where λ_i^b represents the wavelength channel λ_i on ODL b, $1 \leq i \leq k$ and $0 \leq b \leq B$. If the buffer is empty at the beginning of a time slot, all these $(B+1)k$ channels will be available for newly arrived packets. However, it is not the case if the buffer is not empty. Channel λ_i^b is available if and only if there is no packet on wavelength λ_i which will come out of an ODL after b time slots. For example, λ_i^{B-1} is not available if there is a packet on λ_i that was directed to ODL B at the previous time slot. λ_i^{B-2} is not available if there is a packet on λ_i that was directed to ODL $B-1$ at the previous time slot, or to ODL B two time slots ago. We can see that $\{\lambda_1^B, \lambda_2^B, \ldots, \lambda_k^B\}$ will always be available, since there will be no previously arrived packet coming out of any ODL after B time slots because the longest delay time is B.

Given the buffer occupancy state, we can find the set of available wavelength channels for newly arrived packets in linear time. After getting the available channels, we can draw a bipartite graph that will be referred to as *request graph* as follows. The right side vertices represent the arrived packets destined to this output fiber sorted according to their wavelength indexes, lower indexed first. There might be several packets on the same wavelength, in this case their orders are arbitrary. The left side of the vertices represent the available wavelength channels also sorted according to wavelength indexes. There may also be more than one available wavelength channels of the same wavelength. In this case, we put the wavelength channel with a shorter delay in a higher position (though it is not necessary). There is an edge connecting a left side vertex a and a right side vertex b if the wavelength of the packet represented by b can be converted to the output wavelength represented by a. Such a request graph is shown in Fig. 3.

We could freely choose which side of vertices should represent the arrived packets and which side of vertices should represent the available wavelengths. Here we put the avail-

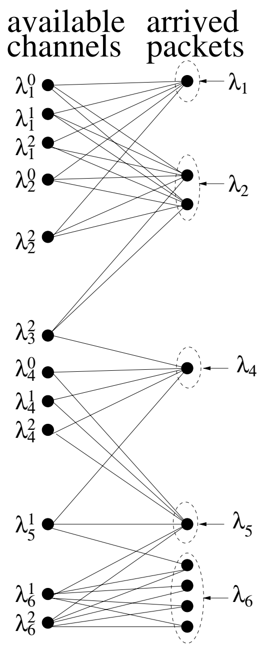


Fig. 3. A request graph with 6 wavelengths and 3 fiber delay lines and the conversion ability as defined in Fig. 1.

able wavelengths on the left side simply because the wavelengths will be assigned weights and it is more convenient to consider the weighted vertices are on the left side of the bipartite graph. In a request graph G, let E denote the set of edges. Any wavelength assignment can be represented by a subset of E, E', where edge $ab \in E'$ if wavelength channel a is assigned to packet b. Under unicast traffic, any packet needs only one output channel and an output channel can be assigned to only one packet. It follows that the edges in E' are vertex disjoint, since if two edges share a vertex, either one packet is assigned two wavelength channels or one wavelength channel is assigned to two packets. Thus, E' is a *matching* in G. For a given set of packets, to maximize network throughput, we should find a maximum cardinality matching in the request graph. This problem for a bufferless WDM switching network was studied and solved in [17].

However, for a buffered switching network, although a maximum cardinality matching maximizes network throughput, it may or may not give the best wavelength assignment, as we should also consider the delay time of the packets. Note that in [17] the network is bufferless and there is no packet delay since a packet is either transmitted immediately or rejected. In the buffered case, a packet that is granted may have several choices or can be assigned to several ODLs. It is naturally preferred to choose the one with shortest delay time. When considering all the arrived packets together, we want to minimize the total delay time of the granted packets. Note that this is under the pre-condition that we grant the maximum number of packets, since otherwise we can simply reject all arrived packets and the total delay time of the granted packets would be zero. As will be seen later, the solution we give will have the following two properties: (1) the number of granted packets is maximum, and (2) the total delay of the granted packets is minimum among all possible wavelength assignments when the same number of packets are granted.

To solve this problem, we introduce weights to the left side vertices and define the weight of λ_i^b for $1 \le i \le k$ and $0 \le i \le B$ to be $B - b$. Given a matching M of the request graph, the weight of M is the sum of the weights of the left side vertices covered by M. It is a large positive number minus the total delay time of the granted packets. Thus, the larger the weight, the smaller the total delay time. If we can find a matching that has both maximum cardinality and maximum weight, we have found a wavelength assignment that both maximizes network throughput and minimizes total delay.

IV. Maximum matchings in request graphs

Before we move onto solving the problem formalized in the previous section, we first briefly discuss maximum matchings in request graphs. Based on the definition of a request graph, it can be shown that the request graph has the following properties:

Property 1: The adjacency set of any left side vertex is an interval. Namely, if left side vertex a_i is adjacent to right side vertices b_u and b_v where $u < v$, a_i is adjacent to all the vertices between b_u and b_v, or, to all b_w where $u \le w \le v$. In the following we use interval $[begin(a_i), end(a_i)]$ to represent the adjacency set of any left side vertex a_i.

Property 2: Let $[begin(a_i), end(a_i)]$ be the adjacency set of left side vertex a_i, and $[begin(a_j), end(a_j)]$ be the adjacency set of left side vertex a_j. If $i < j$ then $begin(a_i) \le begin(a_j)$ and $end(a_i) \le end(a_j)$.

Property 3: In request graph G, if edge $a_ib_u \in E$, $a_jb_v \in E$ and $i < j$, $u > v$, then $a_ib_v \in E$, $a_jb_u \in E$.

Property 4: In a request graph, edges a_ib_u and a_jb_v are a pair of *crossing edges* if $i < j$ and $u > v$. There exists a maximum matching of a request graph with no crossing edges. As a result, in this matching, the i_{th} matched left side vertex is matched to the i_{th} matched right side vertex.

Property 5: Properties 1 and 2 also hold for right side vertices. Namely, the adjacency set of any right side vertex b_u is also an interval or can be represented by $[begin'(b_u), end'(b_u)]$, and for two right side vertices b_u and b_v, if $u < v$ then $begin'(b_u) \le begin'(b_v)$ and $end'(b_u) \le end'(b_v)$.

Property 6: Removing any vertex from the request graph, all the above properties still hold.

As in [17], we can use a simple algorithm called the First Available Algorithm described in Table 1 for finding a maximum cardinality matching in a request graph. The input to this algorithm is: (1) The left side vertex set A and the right side vertex set B; (2) For each left side vertex a, the set of vertices adjacent to it denoted by interval $[begin(a), end(a)]$. We call $begin(a)$ and $end(a)$ the begin

TABLE 1

FIRST AVAILABLE ALGORITHM

```
First Available Algorithm
 for $i := 1$ to $n$ do
     let $b_j$ be the first vertex in $B$
     adjacent to $a_i$ and is not matched yet
     if no such $b_j$ exists
         $MATCH[i] := \Lambda$
     else
         $MATCH[i] := j$
     end if
 end for
```

value and end value of a, respectively. The output of the algorithm is array $MATCH[]$. $MATCH[i] = j$ means that the i_{th} left side vertex is matched to the j_{th} right side vertex. $MATCH[i] = \Lambda$ if the i_{th} left side vertex is not matched to any left side vertex. In the description of the algorithm n is the number of left side vertices. In this algorithm, left side vertex a_i is matched to the first available right side vertex that is adjacent to it. We can image this as picking the "top" edge in the request graph and adding it to the matching in each iteration. [17] gave a proof for the following theorem.

Theorem 1: First Available Algorithm finds a maximum matching in a request graph.

Finding the first available vertex is easy and can be implemented in hardware. The time complexity of this algorithm is $O(n)$, since the loop is executed n times which is the number of left side vertices.

V. THE SCAN AND SWAP ALGORITHM

In this section we give the Scan and Swap Algorithm for finding an optimal scheduling for a buffered WDM switching network. Recall that the problem to be solved is: Given a request graph with weighted left side vertices, find a matching that maximizes both the number and the total weight of the covered left side vertices. (We will soon explain we can do both.) Such a matching is called an *optimal matching*. We will first consider a greedy algorithm called matroid algorithm.

A. Matroid Greedy Algorithm

Loosely speaking, a matroid is a structure on a set S and a family of subsets of S with properties such as independence defined for these subsets. Greedy algorithms can be used for finding optimal solutions for a matroid. In a graph, the set of vertices that can be covered by a matching is a matroid. Thus, an optimal matching of an arbitrary bipartite graph can be found by the following matroid greedy algorithm [1]: Start with an empty list. In step t, let a_i be the left side vertex with the t_{th} largest weight. Check if there is a matching covering a_i and all the previously selected vertices in the list. If yes, add a_i to the list. Otherwise leave a_i uncovered. Set $t \leftarrow t+1$ and repeat until all vertices have been checked.

The resulting matching is optimal in a strong sense:

1. It is a maximum cardinality matching.
2. The sum of the weights covered by the matching is maximum.
3. The matching is also lexicographically maximum: Let the matching found by the greedy algorithm be M and let $a_1, a_2, \ldots, a_{|M|}$ represent the covered left side vertices by M sorted in an non-increasing order according to their weights. Let M' be any other matching and $a'_1, a'_2, \ldots, a'_{|M'|}$ be the covered left side vertices by M' sorted in an non-increasing order according to their weights. Then $w(a'_i) \leq w(a_i)$ for all $1 \leq i \leq | M' |$ where $w()$ is the weight of a vertex.

The key operation of the matroid algorithm is to check if a matching covering the new vertex and all the previously selected vertices can be found. Suppose we are checking vertex a_i. Let the matching covering all the vertices in the list in this step be M_i. If there is an M_i alternating path with one end being a_i and the other end being an unmatched right side vertex, we can find such a matching. Otherwise no such matching exists and a_i should be left unmatched.

It has been shown in [4] how to find an optimal matching in a convex bipartite graph using the above algorithm. A bipartite graph is convex if it satisfies Property 1. [4] showed that in convex bipartite graphs the optimal matching can be found in $O(n(m+n))$ time, where n is the number of left side vertices and m is the number of right side vertices. The task of finding M_i alternating paths can be greatly simplified by using the interval property of adjacent sets. This algorithm can be directly applied to finding optimal scheduling in our request graph as well. However, since there can be up to Nk vertices on the right and $(B+1)k$ vertices on the left, the running time will be $O(k^2B^2 + k^2BN)$, which may be too long and not suitable for a WDM switch. Next we will give a simpler and faster algorithm by using some special properties of the request graph.

B. Scan and Swap Algorithm

In the problem we are considering, it may be the case that more than one left side vertices are of the same weight. This is because the weight corresponds to the length of the ODL and there may be more than one available channels on the same ODL. As a result, these vertices will be of the same weight. Thus, if we can consider these vertices together at a time instead of checking them one by one, the total amount of work can be reduced. This observation leads to our Scan and Swap Algorithm.

Given the set of weighted left side vertices, we partition them into different groups according to their weights. The vertices in the same group are of the same weight. There are W groups, where $W = B+1$. For any optimal matching M of the request graph, let m_i be the number of vertices covered by M of weight i, $1 \leq i \leq W$. We say the matroid greedy algorithm has "finished stage i" when it has checked all the

TABLE 2
SCAN AND SWAP ALGORITHM

```
Scan and Swap Algorithm
 Set all the right side vertices unmarked.
 $List \leftarrow \emptyset . i \leftarrow 1$
 for $i = 1$ to $n$
     Find the first adjacent right side vertex
     to $a_i$ that has not been marked.
     if there is such a vertex
         $List \leftarrow List \cup \{a_i\}$
         Mark this right side vertex.
     else
         if $a_i$ is a compulsory vertex
             $List \leftarrow List \cup \{a_i\}$
             Let $a_t$ be the nearest non-compulsory
             vertex to $a_i$ in the list
             $List \leftarrow List - \{a_t\}$
         end if
     end if
 end for
```

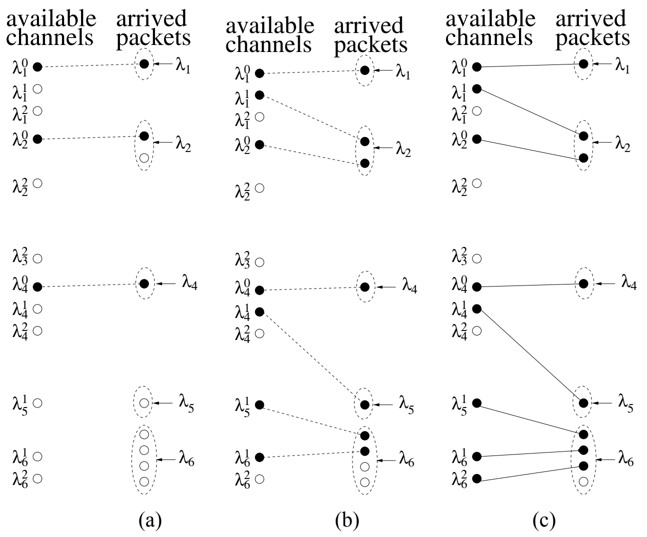


Fig. 4. Outputs of the Scan and Swap Algorithm when applied to the request graph in Fig. 3 at stages 1, 2 and 3. (a) Stage 1. (b) Stage 2. (c) Stage 3.

vertices of weight $W - i + 1$ (weight W will be check first at stage 1). After finishing stage i, it should have found a matching that covers maximum number of vertices of weight $W - i + 1$ while keeping all previously selected vertices with weight more than $W - i + 1$ covered. Otherwise we can find another matching M' which covers more vertices of weight $W - i + 1$, and as a result the matching found by the matroid greedy algorithm will not be lexicographically maximum. Furthermore, note that the order by which the matroid greedy algorithm checks the vertices within a group of the same weight does not change the total weight of the optimal matching, or more precisely, the number of covered vertices within each group. Thus, if we can find the maximum number of vertices of weight $W - i + 1$ that can be covered by a matching along with all previously selected vertices in any stage i (not by the matroid greedy algorithm but by some other algorithms), we have done an equivalent job as the matroid greedy algorithm.

From now on, we will focus on finding such an algorithm. Let the original graph be G^0. The input to this algorithm is: (1) A set of *compulsory* vertices which is a subset of the left side vertices in G^0 that we know can be covered by a matching. (2) A set of *non-compulsory* vertices which is also a subset of the left side vertices in G^0. The algorithm should find the maximum number of non-compulsory vertices that can be covered by a matching under the constraint that all the compulsory vertices are still covered. The algorithm is called the *Scan and Swap Algorithm* and is described in Table 2. The output of the algorithm is stored in $List$.

We now explain how to use this algorithm for finding an optimal matching in the original graph G^0. We will need to run this algorithm W times, one for each stage (each weight). In stage i, the set of input compulsory vertices to this algorithm is those left side vertices with weight higher than $W - i + 1$ that have been selected previously. We express the compulsory vertices at stage i as $CMP(i-1)$. At stage 1 there is no compulsory vertex and $CMP(0)$ is empty. The set of non-compulsory vertices at stage i is those left side vertices with weight $W - i + 1$. (It is interesting to notice that when there is no compulsory vertex, the Scan and Swap Algorithm is equivalent to the First Available Algorithm and will find the maximum number of vertices that can be covered by a matching.) When stage i terminates the output $List$ contains the compulsory vertices for the next stage. The process is continued until $i = W$. If this algorithm is correct (which will soon be proved), at each stage, say, stage i, when the algorithm finishes, it finds the maximum number of vertices with weight $W - i + 1$ that can be covered by a matching along with all the compulsory vertices (vertices selected in the previous stages). Thus when we finish stage W we find an optimal matching of G^0.

Fig. 4 shows the outputs of the Scan and Swap Algorithm when applied to the request graph in Fig. 3 at stages 1, 2 and 3. The black left side vertices are in the list. The black right side vertices are marked. The dashed lines in (a) and (b) indicate that there can be a perfect matching between the black left and right side vertices, but they are subject to change in the following stages. The solid lines in (c) are the final selections.

Fig. 5 shows step 3 and step 4 in stage 3 of the Scan and Swap Algorithm when applied to the request graph in Fig. 3. We can see that after step 3, vertex λ_1^2 was in the list. However, it is swapped out at step 4, since compulsory vertex λ_2^0 finds all its adjacent vertices marked, and λ_1^2 is the nearest non-compulsory vertex in the list.

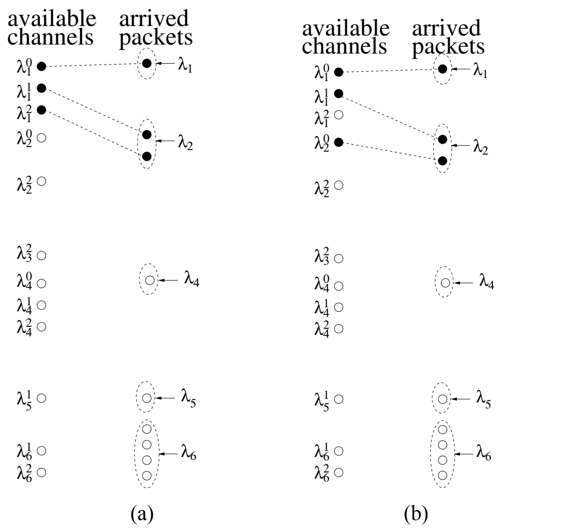


Fig. 5. Step 3 and step 4 in stage 3 of the Scan and Swap Algorithm when applied to the request graph in Fig. 3. (a) After step 3. (b) After step 4.

C. Correctness Proof of the Scan and Swap Algorithm

We now prove that the Scan and Swap Algorithm is correct, i.e., it indeed finds the maximum number of non-compulsory vertices that can be covered along with all the compulsory vertices. We claim that at any step i of the algorithm, the following three invariants hold for the left side vertices added to the list and the marked right side vertices:

1. There is a perfect matching between the vertices in the list and the marked vertices.
2. The list contains all the compulsory vertices scanned from a_1 to a_i and the maximum number of non-compulsory vertices from a_1 to a_i that can be covered by a matching along with these compulsory vertices.
3. Let b_u be the marked vertex with the largest index. For any other matching M' that covers all the compulsory vertices from a_1 to a_i and the same number of non-compulsory vertices from a_1 to a_i (not necessarily the same vertices), let $b_{u'}$ be the right side vertex covered by M' with the largest index. We have $u \leq u'$.

Note that if the first left side vertex cannot be added to the list, it is an isolated vertex. We can remove it from G until we find a non-isolated vertex. Hence from now on we only work on request graphs where the first left side vertex is not isolated.

We introduce some notions here. Let G be the subgraph of the request graph G^0, with left side vertices being all the compulsory vertices and the non-compulsory vertices and the right side vertices being all the right side vertices of G^0. The list at step h is denoted by $List(h)$. At step h there are $p(h)$ left side vertices in the list and $p(h)$ marked right side vertices (the numbers are the same, which will be proved later). Let the set of marked right side vertices be $Marked(h)$. Let G_h be the subgraph of G with vertex set of $List(h) \cup Marked(h)$. Let $L(h)$ be the number of non-compulsory vertices in $List(h)$. For a left side vertex a, we refer to the first right side vertex adjacent to a as the "begin vertex" of a and the last right side vertex adjacent to a as the "end vertex" of a.

We now prove the first Invariant.

Lemma 1: The first invariant of the Scan and Swap Algorithm is true.

Proof. By induction. Consider the first step. Since a right side vertex adjacent to the first left side vertex, a_1, can be found, a_1 can be added to the list and it will mark exactly one right side vertex adjacent to it. Thus the claim is true when $h = 1$.

Suppose it is true for all the steps before $h+1$. When a_{h+1} is checked, if it can be added to the list, then there is a right side vertex adjacent to it, and it marks exactly one new right side vertex. Hence the claim is still true. Now if the adjacent vertices to a_{h+1} are all marked, if a_{h+1} is a non-compulsory vertex, it will not be added to the list, and the there is still a perfect matching from the vertices in the list to the marked vertices on the left side. Thus we only need to consider the case that a_{h+1} is a compulsory vertex and all its adjacent vertices are marked.

Consider when we have finished the h_{th} step. By the induction hypothesis, G_h has a perfect (also maximum) matching. By Property 4 of the request graph, there is a maximum matching M_h that has no crossing edges, that is, the i_{th} left side vertex is matched to the i_{th} right side vertex (in G_h). Let a_y be the vertex in $List(h)$ with the largest index that is matched to its begin vertex. (The begin vertex is still defined in G and not in G_h, i.e., y is still the index in G). There must be such a vertex since the first vertex in the list will always be matched to its begin vertex. We define the *end-segment* of G_h, G_{h-es}, as follows. G_{h-es} is a subgraph of G_h that contains vertex a_y, all the left side vertices in $List(h)$ with larger indexes than a_y, and all the right side vertices in $Marked(h)$ that are matched to them in M_h. We have the following claim regarding G_{h-es}:

Claim: The right side vertices in G_{h-es} are an interval in G. In other words, there is no such a right side vertex b_u which is in G but not in G_{h-es} while there exist two vertices b_w and b_v in G_{h-es} and $w < u < v$.

Proof of the Claim. By contradiction. If the right side vertices in G_{h-es} are not an interval in G, we can find a vertex b_u in G but not in G_{h-es} and two other vertices b_w and b_v in G_{h-es} such that $w < u < v$. Let b_v, $u < v$, be the first such right side vertex (with the smallest index) in G_{h-es}. It is shown in Fig. 6(a). Consider the case that b_v was first marked by some vertex a_t. We claim that a_t must have a begin value of v. Since if it is not the case, $begin[a_t] < v$. Since all the vertices between b_u and b_{v-1} are not marked, and since we do not unmark any marked vertices, they must be also unmarked when the algorithm checks a_t. Then there must be a vertex adjacent to a_t and unmarked with a smaller index than b_v. Then a_t would not have marked b_v.

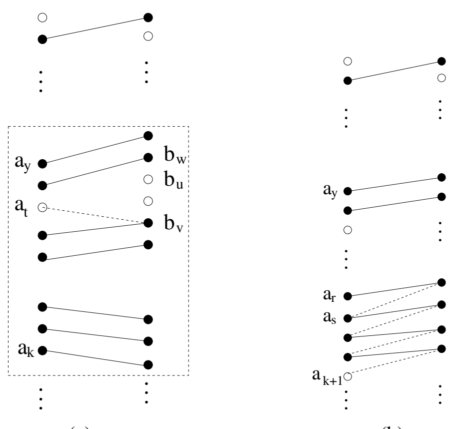


Fig. 6. Auxiliary figures for proving Lemma 1. (a) End-segment of G_h. (b) a_{h+1} can be matched by finding an M_h-alternating path starting at a_{h+1} and ending at the nearest non-compulsory vertex.

If we run the First Available Algorithm on G_h, b_v will be still matched to a vertex with a begin value of v. Consider the case that b_v was first marked. Suppose at step t, a_t marked b_v and was added to the list (it may be swapped out later). Assume $p(t-1)$ vertices were marked before b_v. From the inductive hypothesis, at this step (the t_{th} step), there is a perfect matching in G_t, and by Property 4 of the request graph, the i_{th} vertex in the list is matched to the i_{th} marked vertex. Thus there are exactly $p(t-1)$ vertices in the list before a_t. Now consider running the First Available Algorithm on G_h. The $p(t-1)$ first marked right side vertices will still be matched to the first $p(t-1)$ vertices in the list, and b_v will be matched to the $(p(t-1)+1)_{th}$ vertex in $List(h)$. By the Scan and Swap Algorithm, the index of the $((p(t-1)+1))_{th}$ vertex in $List(h)$ is no less than that of the $(p(t-1)+1)_{th}$ vertex in $List(t)$, since from step $t+1$ to step h we only add a vertex with a larger index than t to the list. It follows that b_v will be matched to some vertex with an index no less than t. By Property 2 of the request graph, b_v will be matched to a vertex with a begin value of v. Note that b_v cannot be matched to a_y in M_h, since otherwise b_w will be matched to some vertex with a larger index than a_y and will cause edge crossing. This contradicts with the assumption that a_y is the first vertex matched to its begin vertex. Thus the claim is true. **End of Proof for the Claim.**

Now consider the $(h+1)_{th}$ step. a_{h+1} is a compulsory vertex and cannot be added to the list directly. It must be the case that all the vertices adjacent to it have been marked. Given the end-segment G_{h-es}, at step h, we add a_{h+1} to G_{h-es} (also add all the edges) and call it G'_{h-es}. We claim that there must be at least one non-compulsory vertex in G'_{h-es}. Since the right side vertices in G'_{h-es} are all the adjacent vertices of the left side in G'_{h-es}, as they form an interval that begins with the begin value of the smallest left side vertex and ends with the end value of the largest left side vertex. If all the left side vertices are compulsory vertices, then a matching that covers all of them cannot be found, since there is one more vertex on the left than on the right. Let a_r be the non-compulsory vertex with the largest index (closest to a_{h+1}). In M_h, we know that the next vertex in the list, say, a_s, is not matched to its begin vertex. We also know that its begin vertex is matched to some left side vertex with a smaller index. Then a_s must be adjacent to the vertex matched to a_r. For the same reason, the next vertex to a_s, say, a_t, is adjacent to the right side vertex matched to a_s. This argument can be carried on till the last vertex in $List(h)$. All of them are adjacent to the right side vertex matched to the previous one in the list. Now we can draw an M_h-alternating path that begins at a_{h+1} and ends at a_r, as shown in Fig. 6(b). It follows that by removing a_r, a_{h+1} can be matched, and all other vertices in the list are still matched. ■

Next we show that the Invariants 2 and 3 are also true.

Lemma 2: Invariants 2 and 3 of the Scan and Swap Algorithm are also true.

Proof. By induction. The claim is obviously true in the first step. Now suppose it is true in all the steps before $h+1$. We first show that Invariant 2 is true in the $(h+1)_{th}$ step. Consider the case that in the $(h+1)_{th}$ step, a_{h+1} can be added to the list. If a_{h+1} is a non-compulsory vertex, Invariant 2 is true, since otherwise we can find another matching that covers more than $L(h)+1$ non-compulsory vertices from a_1 to a_{h+1}, where $L(h)$ is the number of non-compulsory vertices in $List(h)$. It follows that this matching covers more than $L(h)$ non-compulsory vertices from a_1 to a_{h+1}, regardless it covers a_{h+1} or not. This is a contradiction to our inductive hypothesis that Invariant 2 is true in the h_{th} step.

If a_{h+1} is a compulsory vertex and can be added to the list, Invariant 2 is obviously true. This is because otherwise Invariant 2 is not true in the h_{th} step, since there must be a matching covering more than $L(h)$ non-compulsory vertices from a_1 to a_h and all the compulsory vertices. Now consider the case that a_{h+1} cannot be directly added to the list. If it is a non-compulsory vertex and Invariant 2 is not true, then there exists a matching M' that covers all compulsory vertices from a_1 to a_{h+1} and covers more than $L(h)$ non-compulsory vertices from a_1 to a_{h+1}. If it dose not cover a_{h+1}, then Invariant 2 is not true in the h_{th} step. Hence it covers a_{h+1}. Consider the subgraph of G that contains all the vertices covered by M' from a_1 to a_{h+1} and all the right side vertices from b_1 to b_u where b_u is the end vertex of a_{h+1}. Call this graph G'. All edges in M' covering vertices from a_1 to a_{h+1} are in the edge set of G'.

If a_{h+1} is not matched to b_u in M', we can transform M' to M'' where a_{h+1} is matched to b_u, since the request graph has Property 3. Thus, all the other left side vertices are matched to right vertices with smaller indexes than b_u, since they cannot access larger indexed vertices by Property 2 of the request graph. It follows that M'' covers no less than $L(h)$ non-compulsory vertices and all the compulsory vertices. Moreover, the most recently used right side vertex has a smaller index than b_u, which contradicts with our assumption that Invariant 3 is true in the h_{th} step. Finally, if this vertex is a compulsory vertex and Invariant 2 is not true, then following the same argument, there is a matching where a_{h+1} is matched to its end vertex b_u while all the compulsory vertices from a_1 to a_h are covered and at least $L(h)$ non-compulsory vertices are covered. This again contradicts with the assumption that Invariant 3 is true in step h. Thus, Invariant 2 is true in the $(h+1)_{th}$ step.

We now prove that Invariant 3 is true in the $(h+1)_{th}$ step. We have seen that Invariant 2 is true in the $(h+1)_{th}$ step. Thus the Scan and Swap Algorithm finds a matching M that covers the maximum number of non-compulsory vertices from a_1 to a_{h+1} and all the compulsory vertices from a_1 to a_{h+1}. Let b_u be the highest indexed right side vertex covered by M (in $Marked(h+1)$). Suppose another matching M' also covers $L(h+1)$ non-compulsory vertices from a_1 to a_{h+1} and all the compulsory vertices from a_1 to a_{h+1}. Let the highest indexed right side vertex covered by M' be $b_{u'}$ and $u' < u$.

Starting with the most recently marked vertex in $Marked(h+1)$, b_u, we find b_v such that all the vertices between b_v and b_u (in G) are marked but b_{v-1} is not marked. We know that b_v was first marked by a left side vertex a_t whose begin value is v, as in the proof for Invariant 1. a_t is also the first such a vertex. Since if not, there is $a_{t'}$ where $t' < t$ also has begin value of v but $a_{t'}$ does not mark b_v. When checking $a_{t'}$, b_v is available since only in later steps when a_t was checked would b_v be marked by our assumption. This is a contradiction. We claim that the vertices added to the list prior to a_t will not be removed by later operations. (These are the vertices in $List(t-1)$.) This is because that we only swap out vertices in the same end-segment as a to-be-added compulsory vertex, and a_t and b_v form an end-segment. a_t might later be removed, but b_v will always be matched to some vertex with begin value of v. After finishing executing the $(h+1)_{th}$ step, all vertices from b_v to b_u are marked and thus can be matched to some left side vertex from a_t to a_{h+1}. Suppose there are X matched non-compulsory vertices in M from a_t to a_{h+1}, where $X = u - v$. We know that in M', from a_t to a_{h+1} there must be fewer than X matched non-compulsory vertices, as M' cannot use vertices with the same or a larger index than b_u and b_v is the begin vertex of a_t. Thus from a_t to a_{h+1} M' covers at most $u-1-v$ vertices. It follows that from a_1 to a_{t-1} there are more vertices covered by M' than by M. This contradicts with the fact that after finishing executing the $(t-1)_{th}$ step, $List(t-1)$ contains the maximum number of non-compulsory vertices from a_1 to a_{t-1} and they will not be removed from the list later. That completes our proof. ■

Combining these two lemmas, we have

Theorem 2: The Scan and Swap Algorithm finds the maximum number of non-compulsory vertices that can be covered along with all the compulsory vertices in a request graph.

D. Complexity and Implementation Issues

We now discuss the complexity and implementation of the Scan and Swap Algorithm. In the algorithm, left side vertices are scanned exactly once. For a left side vertex a_i, we need to check if we can find an unmarked right side vertex adjacent to it, and if yes, mark the first (with the smallest index) such a vertex. This can be done in constant time by maintaining a pointer $current$ which points to the right side vertex that immediately follows the most recently marked right side vertex.

If the adjacency set of this vertex is all marked, if it is non-compulsory we can simply move on to next vertex, otherwise we need to swap out the nearest non-compulsory vertex. We can use a stack structure to store the non-compulsory vertices added to the list, with the newly added non-compulsory vertex at the top of the stack. Once a non-compulsory vertex needs to be swapped out, we can perform a "pop" operation on the stack which can be done in constant time. Thus, the overall complexity of the Scan and Swap Algorithm is $O(n)$ where n is the number of left side vertices.

As mentioned earlier, we would need to run the Scan and Swap Algorithm W times, where W is the number of weights. If there are A_i vertices with weight i, $0 \leq i \leq W-1$, then in the worst case, all the vertices can be matched and the number of left side vertices to be scanned in stage j will be $O(\sum_{i=j}^{W-1} A_i)$. Thus, to find the optimal matching we need $O(\sum_{i=0}^{W-1} iA_i)$ time.

In our applications there are $B+1$ weights and for each weight there are up to k vertices. Thus, when using the Scan and Swap Algorithm for finding the optimal matching, we need $O(kB^2)$ time, where B is the longest delay line length and k is the number of wavelengths per fiber, as compared to the existing algorithm for weighted general convex bipartite graphs with $O(k^2B^2 + k^2BN)$ time complexity in [4].

VI. Simulation Results

We implemented Scan and Swap Algorithm in software and conducted simulations. The network in simulations has 16 input fibers and 16 output fibers with 16 wavelengths on each fiber. The arrivals of connection requests at input channels are bursty: an input channel alternates between two states, the "busy" state and the "idle" state. When it is in the "busy" state it continuously receives packets and all the packets go to the same destination; otherwise the input channel is in "idle" state and does not receive any packets. The length of the busy and idle periods follows geometric distribution. The network performance is measured by the *packet loss probability* which is defined as the ratio of the total number of successfully transmitted packets over the total number of arrived packets. The durations of the packets are all one time slot and for each experiment the simulation program is run for 100,000 time slots.

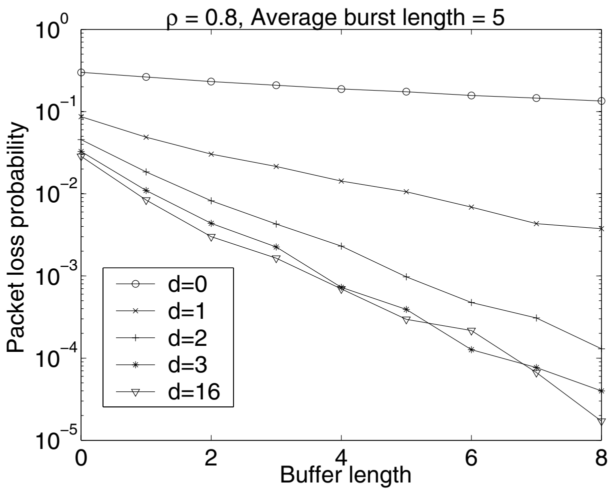
In Fig. 7 we show the packet loss probability of the network as a function of number of fiber delay lines. The traffic is bursty. In Fig. 7(a) the average burst length is 5 time slots and the average idle period is 1.25 time slots. In Fig. 7(b) the average burst length is 40 time slots and the average idle period is 10 time slots. In both cases the traffic load is $\rho = 0.8$. As expected, packet loss probability decreases as the number of delay lines increases. For example, in Fig. 7(a), when the conversion distance $d = 2$, when $B = 0$ (no buffer), the packet loss probability is about $10^{-1.3}$. However, when $B = 4$, it is reduced to about 10^{-3}. As the traffic becomes more bursty, i.e., as the average burst length increases, the packet loss probability decreases much more slowly, as can be observed in Fig. 7(b) where the curves are almost flat. This is because that the burst is too long and always exceeds the buffer capacity.

We can also notice that with the same buffer length, a larger conversion distance always results in a smaller packet loss probability. Also, when the burst is too long, increasing buffer length does not yield much benefit, while increasing conversion distance always does. For example, in Fig. 7(b), when $d = 1$, increasing buffer length does not decrease much of the packet loss probability, but when we increase d to 2, the packet loss probability almost drops by $10^{-0.4}$. This suggests that wavelength conversion ability is more important than buffering in a WDM switching network. However, we can observe that only a relatively small conversion distance is needed to achieve good performances. As can be seen in Fig. 7, the packet loss probability for $d = 3$ is already very close to that for $d = 16$ (full range conversion). This is exactly the reason to use limited range wavelength converters instead of full range wavelength converters.

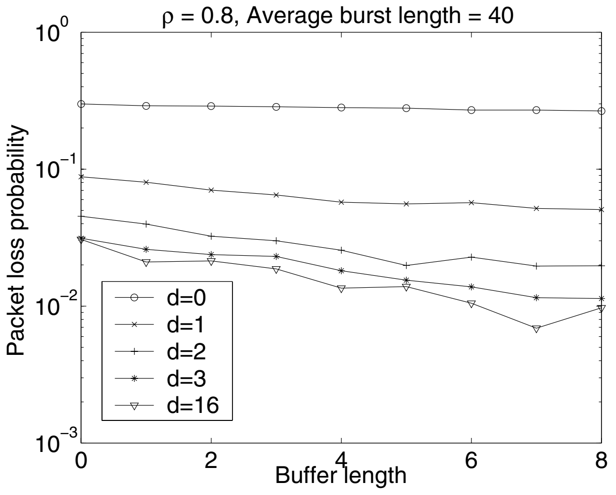
In Fig. 8 we show the average delay of a packet as a function of number of fiber delay lines. The traffic are the same as in Fig. 7. We can see that as the buffer length increases the average packet delay also increases, since fewer packets are dropped and thus more are directed to a buffer before being actually transmitted. For the same buffer size, a larger conversion distance results in a shorter average delay. As in Fig. 8(a), when $B = 4$, the average delay for $d = 1$ is about 0.9 time slots and the average delay for $d = 3$ is only about 0.3 time slots.

VII. Conclusions

In this paper we have studied optimal scheduling in buffered WDM optical switching networks with arbitrary wavelength conversion ability. We formalized the problem as a weighted bipartite graph matching problem and showed that maximum network throughput and minimum delay can be found by finding an optimal matching of the bipartite graph. We utilized the fact that there are many vertices of the same weight in this special bipartite graph and presented the Scan and Swap Algorithm for finding the optimal matching in $O(kB^2)$ time, where k is the number of wavelengths per fiber and B is the buffer length, as compared to $O(k^2B^2 + k^2BN)$ time by directly adopting other existing algorithms for more general weighted bipartite graphs where N is the number of input fibers.



(a)



(b)

Fig. 7. Packet loss probability of the WDM switching network under bursty traffic. The load is $\rho = 0.8$. (a) Average burst length is 5 time slots. (b) Average burst length is 40 time slots.

References

[1] Eugene L. Lawler, "Combinatorial Optimization:Networks and Matroids," *Holt, Rinehart and Winston*, 1976.

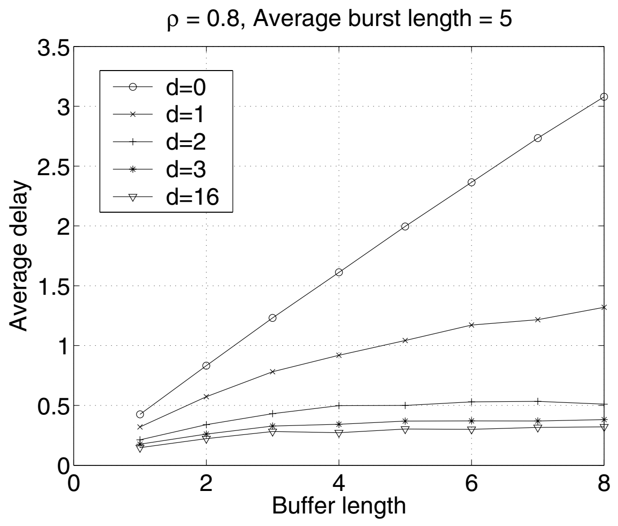
[2] J. Hopcroft and R. Karp "An $n^{\frac{5}{2}}$ algorithm for maximum matchings in bipartite graph," *SIAM J. Comput.*, 2(4):225-231, Dec. 1973.

[3] F. Glover "Maximum matching in convex bipartite graph," *Naval Res. Logist. Quart.*,14, pp. 313-316, 1967.

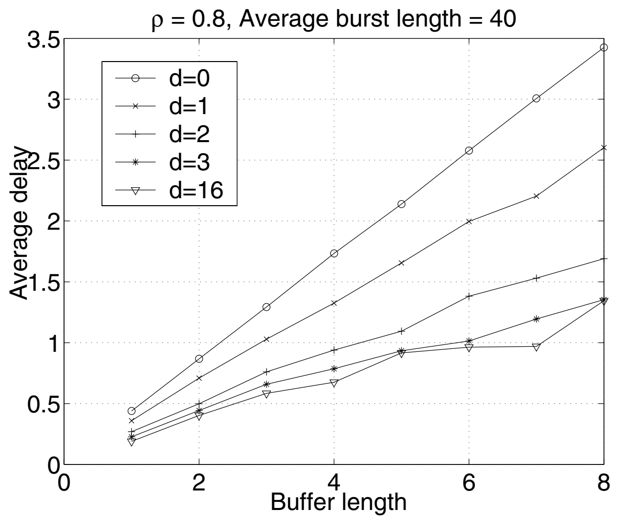
[4] W. Lipski Jr and F.P. Preparata "Algorithms for maximum matchings in bipartite graphs," *Naval Res. Logist. Quart.*,14, pp. 313-316, 1981.

[5] B. Mukherjee, "WDM optical communication networks:progress and challenges," *IEEE JSAC*, vol. 18, no. 10, pp. 1810-1824, Oct. 2000.

[6] D. K. Hunter, M. C. Chia and I. Andonovic "Buffering in



(a)



(b)

Fig. 8. Average delay of the WDM switching network under bursty traffic. The load is $\rho = 0.8$. (a) Average burst length is 5 time slots. (b) Average burst length is 40 time slots.

optical packet switches," *J. Lightwave Technology*, vol. 16 no. 12, pp. 2081-2094, 1998.

[7] M. Kovacevic and A. Acampora, "Benefits of wavelength translation in all-optical clear-channel networks," *IEEE JSAC*, vol. 14, no. 5, pp. 868 -880, June 1996.

[8] S.L. Danielsen, et. al, "Analysis of a WDM packet switch with improved performance under bursty traffic conditions due to tuneable wavelength converters," *J. Lightwave Technology*, vol. 16, no. 5, pp. 729-735, May 1998.

[9] H. J. Chao, C. H. Lam and E. Oki, *Broadband Packet Switching Technologies*, Wiley-Interscience, 2001.

[10] W. J. Goralski, *Optical Networking and WDM*, McGraw-Hill, 2001.

[11] T. Tripathi and K. N. Sivarajan, "Computing approximate blocking probabilities in wavelength routed all-optical networks with limited-range wavelength conversion,"*IEEE JSAC*, vol. 18, pp. 2123–2129, Oct. 2000.

[12] G. Shen, et. al, "Performance study on a WDM packet switch with limited-range wavelength converters," *IEEE Comm. Letters* , vol. 5, no. 10, pp. 432-434, Oct. 2001.

[13] L. Xu, H. G. Perros and G. Rouskas, "Techniques for optical packet switching and optical burst switching," *IEEE Comm. Magazine*, pp. 136 - 142, Jan. 2001.

[14] R. Ramaswami and G. Sasaki, "Multiwavelength optical networks with limited wavelength conversion," *IEEE/ACM Trans. Networking*, vol. 6, pp. 744–754, Dec. 1998.

[15] X. Qin and Y. Yang, "Nonblocking WDM switching networks with full and limited wavelength conversion," *IEEE Trans. Comm.*, vol. 50, no. 12, pp. 2032-2041, Dec. 2002.

[16] Y. Yang, J. Wang and C. Qiao "Nonblocking WDM multicast switching networks," *IEEE Trans. Parallel and Distributed Systems*, vol. 11, no. 12, pp. 1274-1287, 2000.

[17] Z. Zhang and Y. Yang, "Distributed scheduling algorithms for wavelength convertible WDM optical interconnects," *Proc. of 17th IEEE International Parallel and Distributed Processing Symposium,* Nice, France, April, 2003.

[18] Z. Zhang and Y. Yang, "Performance modeling of bufferless WDM packet switching networks with wavelength conversion," *Proc. of IEEE GLOBECOM 2003*.

[19] S.L. Danielsen, et. al, "WDM packet switch architectures and analysis of the influence of tunable wavelength converters on the performance," *J. Lightwave Technology*, vol. 15, no. 2, pp. 219-227, Feb. 1998.