Prioritized Scheduling in WDM Packet Switching Networks with Limited Range Wavelength Conversion

Zhenghao Zhang and Yuanyuan Yang

Dept. of Electrical & Computer Engineering, State University of New York, Stony Brook, NY 11794, USA

ABSTRACT

In this paper we study scheduling problems in unbuffered WDM packet switching networks when the packets have different priorities. The WDM packet switching networks we consider have wavelength conversion ability. We focus on limited range wavelength conversion since it is easier to implement and more cost effective than full range wavelength conversion and also contains the latter as a special case. We formalize the problem of optimal scheduling as a problem of finding an optimal matching in a weighted bipartite graph. The optimal matching is capable of both maximizing network throughput and providing good service differentiation. We give a simple algorithm called Zig-Zag Path Insertion Algorithm which solves the problem in $O(NK\log(Nk)+NkD)$ time, where N is the number of input/output fibers of the switching network, k is the number of wavelengths per fiber and D is the conversion degree, as compared to $O(N^2k^2)$ time if directly adopting other existing algorithms.

Index Terms: Wavelength-division-multiplexing (WDM), optical switching network, scheduling, wavelength conversion, limited range wavelength conversion, bipartite graphs, matching, matroid.

I. INTRODUCTION

In this paper, we study Optical Packet Switching Networks with Wavelength Division Multiplexing (WDM) as it has better flexibility and better exploitations of the huge bandwidth of optical systems [7]. Since optical buffers are made of fiber delay lines and are costly and bulky, we will focus on unbuffered switching networks.

In a WDM optical packet switched network, output contention arises when some packets on the same wavelength are destined for the same output fiber. The contention can be solved by converting the wavelength of a packet to some idle wavelength (if there are any) on the output fiber. The translation of wavelengths is achieved by using a wavelength converter which converts a signal on one wavelength to another. The converter can be *full range*, which is capable of converting a wavelength to any other wavelengths, or *limited range*, which only converts a wavelength to several adjacent wavelengths. Full range wavelength converters are quite difficult and expensive to implement due to technological limitations [4], [2], therefore, a more cost-effective and realistic choice is to use limited range wavelength converters.

In many networking applications, packets arrived at a switch may have different priorities. To meet the QoS requirement for different priorities, service differentiation should be given to packets such that the loss probability is higher for lower priority packets than for higher priority packets. Therefore, a good scheduling algorithm is needed for a WDM switch to decide which packet to accept and which packet to reject. Current approaches to meeting the QoS requirement in WDM networks are mainly based on reserving resources for higher priority connection requests, [8]. This approach is simple and easy to implement but may result in low resource utilization. In this paper we will give an optimal scheduling algorithm based on optimal matchings in bipartite graphs. Our proposed method is more cost-effective in the sense that all resources (wavelength channels) can be assigned to all packets whenever possible.

Extensive research has been conducted on scheduling algorithms for various electronic switches (which can be considered as a single wavelength switch). For example, [1] considered parallel scheduling algorithms in input-buffered electronic switches under unicast traffic. Scheduling algorithms for WDM broadcast and select networks were also well studied in recent years. Time slotted WDM switches with limited range wavelength conversion was considered in [3], [5]. [3] only provided some rather preliminary simulation results. [5] gave an optimal algorithm for nonprioritized scheduling in such WDM switches. In this paper we will consider the prioritized scheduling.

II. PRELIMINARIES

A. The Switch Model

The WDM switching network is shown in Figure 1. It has input N fibers and N output fibers, on each fiber there are k wavelengths that carry independent data. An input fiber is first fed into a demultiplexer, where different wavelength channels are separated from one another. An input wavelength is then fed into a wavelength converter to be converted to a proper wavelength. The output of a wavelength converter is then split into N copies of itself, which are connected to each of the output fibers under the control of N SOA gates. The signal can reach the output fiber if the SOA gate is on, otherwise it is blocked. In the front of each output fiber there is an optical combiner which multiplexes the signals on different wavelengths into one composite signal and send to the output fiber. Apparently, it is required that all signals fed to the optical combiner should be on different wavelengths.

As in [3], [7], we assume that the switch operates in a synchronous mode and optical packets or cells arrive at the network at the beginning of time slots. We consider the case where all packets are of the same priority and of same length. 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

The research work was supported in part by the U.S. National Science Foundation under grant numbers CCR-0073085 and CCR-0207999.

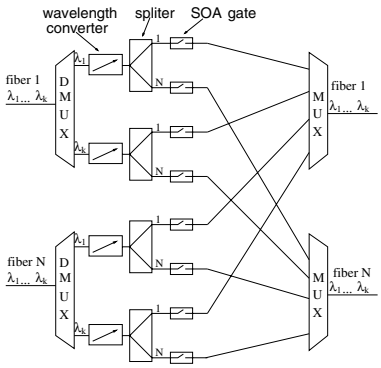

Fig. 1. A wavelength convertible WDM switch.

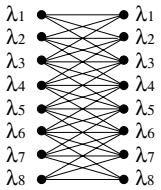

Fig. 2. Conversion on an optical fiber with 8 wavelengths.

directed to, and can be assigned to any free wavelength channel accessible to it.

B. 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. We also define the *conversion distance* as the largest difference between the index of a wavelength and a wavelength that it can be converted to. The conversion distance is denoted by d. The largest conversion degree of all wavelengths is denoted by D.

We assume that

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

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

Wavelength conversion can be illustrated by a bipartite graph, in which left side vertices represent input wavelengths and right side vertices represent output wavelengths. If input wavelength λ_i can be converted to output wavelength λ_j, there is an edge connecting them. Figure 2 is an example when $k = 8$ and $d = 2$.

III. Problem Formalization

In this section we show how optimal scheduling in the WDM switch can be formalized into a bipartite graph matching problem. First consider one output fiber and the packets arrived for this output fiber. The relationship between the packets and the wavelength channels on that output fiber can be described by a bipartite graph, called *request graph*. The left side vertices of the request graph represent the packets and the right side vertices represent output wavelengths. The vertices on each side of the graph are depicted according to their wavelength indexes, λ_1 first, then λ_2, then λ_3 and so on. For the right side vertices, there is exactly one vertex on each wavelength. There could be multiple left side vertices on the same wavelength since there may be more than one packets on the same wavelength going to the same output fiber. These vertices can be in an arbitrary order in this case. There is an edge connecting a left side vertex a and a right side vertex b if the wavelength of packet a can be converted to output wavelength b.

We can draw a request graph for each output fiber. Since we will never assign a wavelength channel on output fiber j to a packet destined to output fiber i if $i \neq j$, there will be no connections between vertices belonging to different request graphs. Therefore the decisions in one request graph do not affect the decisions in other request graphs, and in the following we will explain our algorithm for one request graph. We can run the same algorithm for each of the N request graphs, either one by one or in parallel, to obtain the optimal scheduling.

For convenience, we also define the *request vector*. A request vector is a $1 \times k$ row vector, with the i_{th} element representing the number of packets arrived on wavelength λ_i. Figure 3(a) shows the request graph when the request vector is $[1,2,2,2,0,0,0,1]$. The number in the parenthesis on the right of each left side vertex is the weight of this vertex which will be soon discussed.

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 b is assigned to packet a. Under unicast traffic, any connection request needs only one output channel and an output channel can be assigned to only one connection request. It follows that E' is a *matching* in G, since if two edges share a vertex, either one packet is assigned two wavelength channels or one wavelength channel is assigned to two packets.

For a given set of packets, to maximize network throughput, we should find a maximum matching in the request graph. This problem was studied and solved in [5]. If the connection requests have different priorities, we can assign weights to packets according on their priorities and find a group of contention-free connection requests with maximum total weight. The problem can be formalized as: Given a request graph with weighted left side vertices, find a maximum matching with maximum total weight of the covered left side vertices. Such a matching is called an *optimal matching*. As an example, Figure 3(b) shows an optimal matching for the request graph in Figure 3(a).

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. As in [5], 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. In the description of the algorithm n is the number of left side vertices. By this algorithm, left side vertex a_i is matched to the first available right side vertex adjacent to it. We can image this as picking the "top" edge in the request graph and adding it to

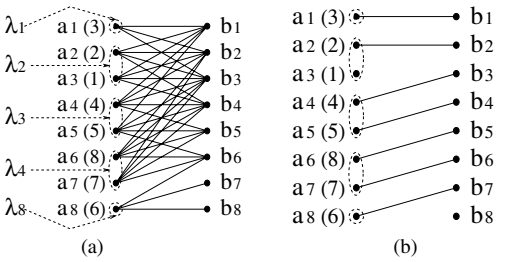

Fig. 3. Request graph and an optimal matching when the request vector is $[1, 2, 2, 2, 0, 0, 0, 1]$ in an 8-wavelength switch with conversion distance 2. The number in the parenthesis on the right of each left side vertex is the weight of this vertex. (a) Request Graph. (b) Optimal matching.

TABLE 1
FIRST AVAILABLE ALGORITHM

First Available Algorithm
for $i := 1$ **to** n **do**
Find b_j which is the right side vertex adjacent to a_i with the smallest index and is not matched to any other vertex yet.
if such b_j exists
match a_i to b_j
else
a_i is not matched
end if
end for

the matching in each iteration. [5] gave a proof for the following theorem.

Theorem 1: First Available Algorithm finds a maximum matching with no crossing edges in a request graph.

Finding the first available vertex is simple 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. OPTIMAL MATCHING IN A VERTEX-WEIGHTED REQUEST GRAPH

A. The Matroid Greedy Algorithm

In this section we show how to find optimal matchings in a vertex-weighted request graph. Optimal matching in an arbitrary bipartite graph can be found by the following simple greedy algorithm which can be called the Matroid Greedy Algorithm [6]. The algorithm starts with an empty set Π. In step t, let a be the left side vertex with the t_{th} largest weight. The algorithm checks whether there is a matching covering a and all the previously selected vertices in Π. If yes, add a to Π, otherwise leave a uncovered. Update $t \leftarrow t + 1$ and repeat until all vertices have been checked. When finished, Π stores left side vertices that can be covered by an optimal matching.

However, the time complexity of the matroid greedy algorithm is $O(n^3)$ for an n-vertex bipartite graph, since to check whether a vertex can be covered along with all the previously covered vertices one needs $O(n^2)$ time. The algorithms in [6] can be used to find an optimal matching in the request graph in $O(N^2k^2)$ time. Next we will give a much simpler and faster algorithm that runs in $O(NK\log(Nk) + NkD)$ time where D is the conversion degree.

B. The Zig-Zag Path Insertion Algorithm

Optimal matching in request graphs can be found more quickly, since we can find easier ways to determine whether a vertex can be covered along with all the previously selected vertices or not. Suppose we are at step t and are checking vertex a_i. We define G_t as the subgraph of G with vertex set of the previously selected left side vertices and all the right side vertices. Let M_t be the matching that is found by running the First Available Algorithm on G_t. We now add a_i to G_t based on the wavelength it is on. Call this graph G'_t.

We first check if there are some unsaturated right side vertices adjacent to a_i in G'_t. If yes, a_i can be covered. Otherwise, we check whether there is a left side vertex on a larger wavelength than that of a_i's. Any such left side vertex will be called a_i's lower neighbor. Among all a_i's lower neighbors, let a_l be the one with the smallest index. a_l is called a_i's *closest lower neighbor* and suppose a_l is matched to b_v. We have that

Lemma 1: If a_i is not adjacent to any unsaturated vertex, a_i can be covered only if a_i has lower neighbors and a_i is adjacent to b_v, the vertex matched to its closest lower neighbor.

Proof. We first show that a_i must have lower neighbors by contradiction. If a_i does not have any lower neighbor, under M_t, all the right side vertices adjacent to a_i must be matched to vertices with smaller indices than a_i. Now suppose there is a matching covering all the left side vertices in G'_t. If there exists such a matching, it can be found by the First Available Algorithm and let it be M'_t. Now consider running the First Available Algorithm on G'_t. All the left side vertices other than a_i were checked before a_i, and their connections to the right side vertices are exactly the same in G_t as in G'_t. Therefore, the matching for these vertices are exactly the same in M_t as in M'_t, and all the vertices adjacent to a_i will again be matched to left side vertices with smaller indices than a_i under M'_t. As a result, a_i cannot be matched to any vertex. This is a contradiction.

In the second case, if a_i has a lower neighbor but a_i is not adjacent to b_v, a_i is not adjacent to any vertices matched to its lower neighbors, since M_t is non-crossing. It follows that all the vertices adjacent to a_i are also matched to vertices with smaller indices than a_i. With similar arguments to that in the first case, we can show that a_i cannot be covered. ■

If a_i is adjacent to b_v, a_i may or may not be covered. To see whether it can be covered or not we define the *zig-zag path* starting from a_i as follows. For simplicity, we use $mat[]$ to denote the vertex matched to a given vertex, for example, $mat[a_l]$ is b_v and $mat[b_v]$ is a_l. The zig-zag path first extends from a_i to b_v, and then from b_v to a_l. When the path extends back to a left side vertex, we call the vertex the current "checking point" and denote it as a_p. We use b_q to denote the right side vertex matched to a_p. At this time a_p is a_l and b_q is b_v. If a_p is not adjacent to b_{q+1}, the path terminates at a_p. Otherwise, if b_{q+1} is unsaturated, the path terminates at b_{q+1}. Else the path extends from a_p to b_{q+1} then to $mat[b_{q+1}]$, and the checking point is now $mat[b_{q+1}]$. This process continues until the path cannot be extended further.

The zig-zag path will eventually end since one of these two

cases will sooner or later become true: (1) The zig-zag path extends to a left side vertex, the current checking point, which is matched to its end vertex. (2) The zig-zag path extends to an unsaturated right side vertex. In this case, The current checking point a_p is not matched to its end vertex, i.e., a_p is adjacent to b_{q+1}, and b_{q+1} is now unsaturated.

We have that

Lemma 2: If a_i is not adjacent to any unsaturated vertex, a_i can be covered if and only if the zig-zag path ends in case (2).

Proof. It is easy to see that the condition is sufficient, since if the algorithm terminates in case (2), the zig-zag path is an M_t augmenting path starting from a_i. To see it is also sufficient we prove it by contradiction. Suppose the claim is not true, i.e., there is a case that a_i can be covered, but the zig-zag path terminates in case (1). If a_i can be covered along with all the previously selected vertices, there is a perfect matching from the left to the right in G'_t. Such a matching can be found by the First Available Algorithm. and let it be M'_t. As shown in the proof of Lemma 1, the matchings for the left side vertices with smaller indices than a_i must be the same in M'_t as in M_t.

Now, under M_t, suppose a_l is the closest lower neighbor of a_i and is matched to b_v. Let the last checking point of the zig-zag path be a_h and matched to b_w. If the zig-zag path terminates in case (1), b_w is the end vertex of a_h. Since the zig-zag path terminates at a_h, all the vertex between b_v and b_w must be saturated. Therefore there are exactly $w - v$ vertices covered by M_t from a_l to a_h.

Note that under M'_t, vertices from a_i to a_h can only be matched to vertices from b_v to b_w, since b_v is the vertex that would be matched to a_i in M'_t and b_w is the end vertex of a_h. Therefore there can be at most $w - v$ vertices covered by M'_t from a_i to a_h. This is a contradiction since M'_t must cover $w - v + 1$ vertices from a_i to a_h: those covered by M_t from a_l to a_h plus a_i. This completes our proof. ■

This lemma suggests that optimal matchings in request graphs can be found as follows. At step t, when checking vertex a_i, first check whether there are some unsaturated right side vertices adjacent to it. If yes, choose one to match to a_i. We call this "direct matching". Otherwise check if a_i has a closest lower neighbor a_l and if a_i is adjacent to b_v. If no, a_i cannot be covered. Otherwise start extending the zig-zag path from a_i. If the path terminates in case (1), a_i cannot be covered. Else a_i can be covered and we say a_i is "inserted".

If a_i can be covered, we need to update the matching. We have seen in the proof of the lemmas that it is very simple to determine whether a_i can be covered if M_t is the same as the matching found by the First Available Algorithm. Therefore we would like to keep this property in the new matching, M_{t+1}. Apparently it would make no sense if we run the First Available Algorithm every time we check a vertex. Instead, if M_t is the same as the matching found by the First Available Algorithm, M_{t+1} can be made to have the same property by only modifying M_t. We now explain how this can be done.

At step t, if a_i cannot be directly matched but can be covered by insertion, we can match a_i to b_v, match $mat[b_v]$ to b_{v+1}, match $mat[b_{v+1}]$ to b_{v+2}, ..., and match a_h which is the last checking point to the unsaturated right side vertex, as what is suggested by the zig-zag path. The matchings for all other vertices are kept unchanged. This new matching will be exactly the same as that would be found by the First Available Algorithm on G'_t, since the matchings for left side vertices with smaller indices than a_i and larger indices than a_h are not changed, and vertices from a_i to a_h are all matched to their first available vertices. Else, if a_i can be directly matched, we first find the unsaturated vertex adjacent to a_i with the smallest index and let it be b_u. Suppose a_l is matched to b_v in M_t. If $u < v$, a_i should be matched to b_u, and the matchings for all other vertices need not be changed. This new matching will also be the same as the matching found by the First Available Algorithm as b_u is the first available vertex for a_i. If $u > v$, b_v is the first available vertex for a_i and a_i should be matched to b_v. The matchings for the vertex with larger indices than a_i and matched to vertices with smaller indices than b_u should be shifted down by one vertex.

TABLE 2
ZIG-ZAG PATH INSERTION ALGORITHM

Zig-Zag Path Insertion Algorithm
for $t := 1$ **to** n **do**
  Let a_i be the vertex with the t_{th} largest weight.
  Find b_u which is the right side vertex adjacent to a_i with the smallest index and is not matched to any other vertex yet.
  Find a_l which is the closest lower neighbor of a_i and suppose it is matched to b_v.
  if b_u exists
    if $u < v$ or b_v dose not exist
      match a_i to b_u
    else
      for $s := u$ **downto** $v + 1$ **do**
        $mat[b_s] = mat[b_{s-1}]$
      end for
      match a_i to b_v.
    end if
  else
    if a_l does not exist or a_i is not adjacent to b_v
      a_i cannot be matched.
    else
      Start extending the zig-zag path from a_i.
      if the zig-zag path ends in case (1)
        a_i cannot be covered.
      else
        Let b_w be the unsaturated vertex at the end of the path.
        for $s := w$ **downto** $v + 1$ **do**
          $mat[b_s] = mat[b_{s-1}]$
        end for
        match a_i to b_v.
      end if
    end if
  end if
end for

The complete algorithm is described in Table 2. Based on the discussion above, we conclude

Theorem 2: The Zig-Zag Path Insertion Algorithm finds an

optimal matching in a request graph.

We now analyze the complexity of the newly proposed Zig-Zag Path Insertion Algorithm. The work done in each step of the algorithm consists of the following four parts: (1) Find the first available right side vertex b_u. (2) Find the closest lower neighbor a_l and b_v which is the vertex matched to a_l. (3) Update the matching. (4) Extend the zig-zag path.

Part 1 is a linear search through the adjacency set of a_i and takes $O(D)$ time where D is the conversion degree. Part 2 also takes $O(D)$ time, as we need only to scan at most D wavelengths starting from the wavelength of a_i. If no left side vertices can be found in this range, even if a_i has a closest lower neighbor a_l, a_l will be matched to a vertex not adjacent to a_i. Since there are up to Nk left side vertices, overall the time spent in parts 1 and 2 is $O(NkD)$.

There are two cases in part 3, update the matching. The first case is the simple one: match a_i to b_u without changing other matchings, which can be done in constant time. Overall the time spent in this simple update is bounded by $O(k)$, since there can be at most k vertices covered by the optimal matching. The second case of update requires shifting down the matchings for several left side vertices and the number of vertices involved in this shift could vary. However, note that when update a matching, the matching for a left side vertex will only move down and will never go up. Therefore, a left side vertex can be involved in this type of update for no more than D times, where D is the conversion degree. Overall the time spent on this type of update is bounded by $O(kD)$.

The time spent on part 4 is determined by how many vertices the zig-zag path visited before reaching the end point and can be done in $O(kD)$ time.

As a conclusion, the time complexity of the Zig-Zag Path Insertion Algorithm is $O(NkD)$. Note that in order to find the optimal scheduling, the left side vertices should first be sorted according to their weights since in the algorithm the vertices with larger weights are checked first. The sorting will need $O(Nk\log(Nk))$ time. Overall, including sorting, we need $O(Nk\log(Nk)+NkD)$ time to find the optimal scheduling.

VI. Simulation Results

Besides giving proofs and analyses for the proposed scheduling algorithm, we also implemented the algorithm in software and tested them by simulations. We tested the switching networks of various sizes, due to the limit of space we show the results for $N = 8$ and $k = 8$. In the simulations, we assume that the traffic is bursty and the length of the busy and idle periods follows geometric distribution. The network performance is measured by the *packet loss probability*.

In Figure 4 we plot the packet loss probabilities for various prioritized packets as a function of conversion distance. The tested traffic load is $\rho = 0.8$, and average busy period is 16 time slots and average idle period is 4 time slots. There are four priorities, each of 25% of the total traffic The highest priority is 1 and the lowest priority is 4. We can see that the packet loss probability for all priorities decreases as the conversion distance increases. But when the conversion distance is larger than a certain value, the decrease of packet loss probability is marginal. In this case there is little benefit for further increasing the conversion degree, which is exactly the reason for using limited range wavelength converters other than the full range wavelength converters. It can also be seen that the Zig-Zag Path Insertion Algorithm achieves good service differentiation.

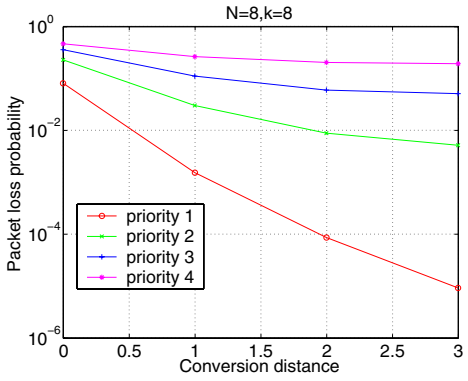

Fig. 4. Packet loss probability of WDM switching networks for packets with 4 different priorities in an 8×8 switch with 8 wavelengths per fiber.

VII. Conclusions

In this paper we have studied optimal scheduling in prioritized unbuffered WDM packet switching networks with limited wavelength conversion. We formalize the problem of optimal scheduling as a problem of finding an optimal matching in a weighted bipartite graph. We give a simple algorithm called Zig-Zag Path Insertion Algorithm which solves the problem in $O(NK\log(Nk)+NkD)$ time, where N is the number of input/output fibers of the switching network, k is the number of wavelengths per fiber and D is the conversion degree, as compared to $O(N^2k^2)$ time if directly adopting other existing algorithms. We also conducted simulations to study the network performance of the switching networks under this scheduling algorithm and the results show that the algorithm indeed gives good service differentiation.

References

[1] N. McKeown, "The iSLIP scheduling algorithm input-queued switch," *IEEE/ACM Trans. Networking*, vol. 7, pp. 188-201, Apr. 1999.

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

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

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

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

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

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

[8] A. Kaheel, et. al., "Quality-of-service mechanisms in IP-over-WDM networks," *IEEE Communications Magazine* , vol. 40, no. 12, pp. 38 -43, Dec. 2002.