Packet Scheduling in a Low-Latency Optical Switch with Wavelength Division Multiplexing and Electronic Buffer

Lin Liu — Dept. Electrical & Computer Engineering, Stony Brook University, Stony Brook, NY 11794, USA

Zhenghao Zhang — Computer Science Department, Florida State University, Tallahassee, FL 30306, USA

Yuanyuan Yang — Dept. Electrical & Computer Engineering, Stony Brook University, Stony Brook, NY 11794, USA

***Abstract*—Optical switches are widely considered as the most promising candidate to provide ultra-high speed interconnections. Due to the difficulty in implementing all-optical buffer, optical switches with electronic buffers have been proposed recently [1] [4] [5]. Among these switches, the Optical Cut-Through (OpCut) switch has the capability to achieve low latency and minimize optical-electronic-optical (O/E/O) conversions. In this paper, we consider packet scheduling in this switch with wavelength division multiplexing (WDM). Our goal is to maximize throughput and maintain packet order at the same time. While we prove that such an optimal scheduling problem is NP-hard and inapproximable in polynomial time within any constant factor by reducing it to the set packing problem, we present an approximation algorithm that maintains packet order and approximates the optimal scheduling within a factor of $\sqrt{2Nk}$ with regard to the number of packets transmitted, where N is the switch size and k is the number of wavelengths multiplexed on each fiber. This result is in line with the best known approximation algorithm for set packing. Based on the approximation algorithm, we also give practical schedulers that can be implemented in fast optical switches. Simulation results show that the schedulers achieve close performance to the ideal WDM output-queued switch in terms of packet delay under various traffic models.**

Index Terms: Optical switch, wavelength division multiplexing (WDM), electronic buffer, packet scheduling, approximate algorithm.

I. Introduction

Optical switches are widely considered as the most promising candidate to provide ultra-high speed interconnections for future applications. While optics serve as the ideal media for interconnection, the development of optical switches currently faces several challenges, one of which is the lack of optical Random Access Memory (RAM). To address these challenges, hybrid optical/electronic switch architectures have been proposed, for example, the PERCS project [1] and the OSMOSIS project [2], [3], [4] at IBM, and the low-latency optical switch proposed in [5]. Among these architectures, the switch in [5], which we refer to as the *OpCut* (Optical Cut-through) switch, has the capability to achieve low latency and minimize optical-electronic-optical (O/E/O) conversions, and will be the focus in this paper. The OpCut switch is equipped with recirculating electronic buffers. An optical packet that arrives at the switch input is sent to the output directly, i.e., “cut-through” the switch, whenever possible. Only the packets that cannot cut-through are converted into electronic format and sent to the electronic buffers. Such packets can be converted back to optics and sent to the output later through the optical transmitters. By allowing packets to cut-through, the OpCut switch is able to achieve low latency as well as minimizing the number of O/E/O conversions.

The packet scheduling problem in a single-wavelength OpCut switch has been studied in [6]. In this paper, we study the OpCut switch under *Wavelength Division Multiplexing (WDM)*. With WDM, the bandwidth of an optical fiber consists of multiple wavelengths, each of which may carry independent data [7]. In the OpCut switch with WDM, both the input and the output fibers carry multiple wavelengths, such that the system capacity is greatly increased comparing to the single wavelength switches. On the other hand, the switch needs a more advanced scheduler to schedule packet transmissions, because multiple packets may arrive at an input port and multiple packets may leave an output port in a time slot. A simple random packet scheduling algorithm was assumed in [5] for the convenience of analytical study. However, a potential problem of the random scheduling algorithm is that random functions are difficult to implement at high speed. Besides, it cannot maintain packet order, which is generally desirable for switches [12] [13]. Thus, our focus in this paper is the packet scheduling in the OpCut switch under WDM. The goal is to schedule as many as possible packets for transmission, while maintaining packet order.

There has been much work on packet scheduling in electronic switches. Packet scheduling is usually formalized as a bipartite matching problem between the input ports and the output ports of the switch. Many algorithms have been proposed, such as Parallel Iterative Matching (PIM) [10] and iSLIP [11]. However, as the OpCut switch has no buffer at the input, these algorithms cannot be applied directly. For example, in input-queued switches, maintaining packet order is trivial, as packets belonging to the same flow are stored in one Virtual Output Queue (VOQ) and can be sent in the order of their arrival time. In the OpCut switch, packets from the same flow may be picked up by different receivers, thus maintaining packet order is more challenging.

The rest of the paper is organized as follows. Section II introduces the WDM OpCut switch, as well as notations and queuing management in this switch. Section III describes the goal of an optimal schedule and the basic scheduling procedure. Section IV focuses on the scheduling problem, including the problem formalization, NP-hardness and inapproximability proof, an approximate algorithm with performance ratio, and practical schedulers based on the approximation algorithm. Section V presents simulation results. Section VI concludes the paper.

II. System Model

In this section we first describe the WDM OpCut switch architecture, then give the notations as well as discussions of the specific queuing management in the OpCut switch for maintaining packet order.

A. Switch Architecture

Like many other proposed switches in the literature [4], [2], the OpCut switch works in time slots. Packets are of fixed

Research supported by US National Science Foundation under grant numbers CCF-0915823 and CCF-0915495.

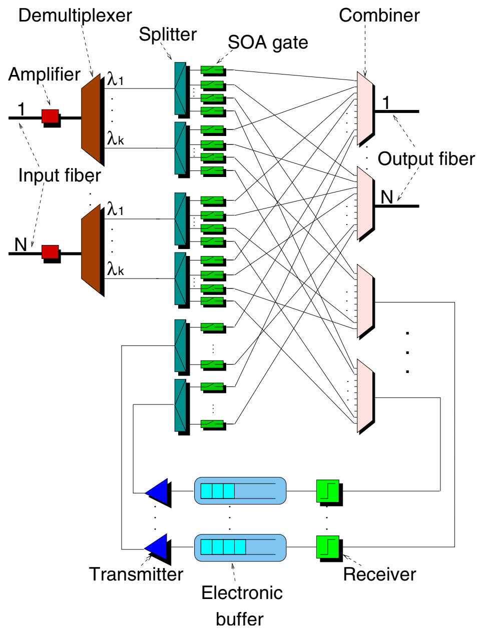

Fig. 1. A possible implementation of the WDM OpCut switch.

length and fit in exactly one time slot, which is approximately 50 ns and similar to that in the OSMOSIS switch [4], [2]. No speedup is assumed in the switch, i.e., the components of the switch run at the external line rate.

One possible implementation of the WDM OpCut switch is shown in Fig. 1. The switch has N input fibers and N output fibers. There are k wavelength channels multiplexed on each input or output fiber. The signal on each input fiber first goes through an amplifier, then is demultiplexed into k signals, one on each wavelength. Up to one packet may arrive in optical format on an input wavelength channel in each time slot. A packet can be routed to an output fiber or a receiver if the corresponding semiconductor optical amplifier (SOA) gate is closed. A newly arrived packet may be directly sent to its desired output fiber, or cut-through the switch. A packet that does not cut-through is picked up by one of the receivers, converted to electronic form and sent to the buffer connected to the receiver. The switch should have a sufficient number of receivers to avoid packet loss. Since up to Nk packets may arrive in a single time slot, Nk is the upper bound of the number of receivers. Each receiver buffer is connected to a transmitter. In each time slot, a transmitter can fetch up to one packet from the buffer it is connected to. The transmitter is a fast tunable laser, and may convert the packet back to optic forms on *any* of the k wavelengths. Under the control of the SOA gates, the packet can be sent to the corresponding wavelength channel of its destined output fiber. Unlike in other optical switches with electronic buffers where every packet goes through the O/E/O conversions, in the OpCut switch, a large percentage of the packets cut-through the switch directly and do not experience the O/E/O delay, as will be seen in the simulation results.

At the switch output, a combiner multiplexes multiple signals into a composite signal. There is no buffer or packet queuing at the switch output. A packet arriving at an output fiber will be picked up by its destined device connected to the output fiber, or be routed to the next stage switch.

B. Notations

In a WDM OpCut switch, input fiber i is denoted as I_i, and output fiber j is denoted as O_j. Wavelength channel λ of input fiber i and output fiber j are further denoted as I_i^λ and O_j^λ, respectively. A *flow* is defined as the sequence of packets from the same input fiber to the same output fiber. The flow from I_i to O_j is denoted as f_{ij}. The flow is defined between an input fiber and an output fiber instead of between an input channel and an output channel, because a packet arriving on one wavelength may appear at the output fiber on another wavelength. There are N^2 flows in total. The time slot in which a packet arrives at the switch input is referred to as the *timestamp* of the packet. Since multiple packets of the same flow can arrive at a switch in the same time slot on different wavelengths, the timestamp alone does not completely define the order of packets. The wavelength on which a packet arrives is used to solve the ambiguity. Namely, between two packets of the same flow that arrive at the switch in the same time slot, the one on a smaller wavelength is considered ahead of the other on a larger wavelength. Correspondingly, as will be seen later, our algorithm maintains packet order by ensuring the following property: assume p_1 and p_2 are two packets in the same flow, and p_1 is ahead of p_2 as defined above, then p_1 leaves the switch either earlier than p_2, or in the same time slot as p_2 but on a smaller wavelength channel. As a result, the order of packets in a flow is preserved no matter how many intermediate switches the packets have to travel through.

Among all packets of a flow that have arrived at the switch but have not been scheduled for transmission to the switch output, the one with the oldest timestamp and arrived on the smallest wavelength is referred to as the *head-of-flow* packet. Note that the head-of-flow packet is not necessarily the oldest packet of the flow currently in the switch. A packet becomes the head-of-flow packet once all packets in front of it in the same flow have been scheduled for transmission (although they may not be physically transmitted yet).

C. Packet Queue Management

In an OpCut switch, an electronic buffer may contain packets from different flows. To manage the packets, one possible way is to use a 3-Dimensional Queue [13], under which a dedicated queue is maintained for each flow in each buffer. However, this approach requires N^2 queues for each buffer, and N^3k queues for the switch, which is unlikely to be scalable. Instead, no queue is maintained in any receiver buffer of the OpCut switch, and any specific packet is located by its timestamp as discussed below.

Note that at any time slot, a receiver can pick up at most one packet. The receiver maintains an array of length 2^b to store the packets, where b is an integer. A packet arriving at the buffer at time slot t is stored in the $(t \mod 2^b)_{th}$ element of the array. Consequently, a packet in the buffer can be located in constant time given the lower b bits of its timestamp and its arrival wavelength. Under this approach, the maximum size of the queue is bounded by 2^b. A collision may occur only if a packet is routed to a receiver, yet another packet picked up by the same receiver $w2^b$ time slots earlier is still in the buffer, where w is a positive integer. When b is reasonably large, such a collision actually indicates heavy congestion since the "older" packet has been buffered for $w2^b$ time slots already. Hence when this occurs, it is fair to discard one of the packets.

For each output fiber, the scheduler of the OpCut switch keeps the information of the packets that are destined for that output fiber and are being buffered in a "virtual input queue" (VIQ) style. Basically, for output fiber O_j, the scheduler maintains N virtual queues denoted as F_{ij} for $1 \leq i \leq N$. For each packet belonging to flow f_{ij} and currently being buffered, F_{ij} maintains its timestamp as well as the index of the buffer the packet is in. These queues are referred to as *index queues* in the following. Note that an index queue does not hold the actual packets. Also, the wavelength on which a packet arrives is not included in the index queue, as it is only useful for determining the order of packets, which is reflected by the order the packets appear in the queue.

Fig. 2 provides an example to illustrate the relationship between the actual buffering status and the index queues. In this example, we assume that there are two buffers and two flows, and packet p_i^λ arrived at the switch on wavelength λ in time slot i. In addition, flow 1 and flow 2 are destined for the same output fiber, while flow 3 is destined for a different output. In the example, the index queue for flow 1 is $[1,1] \rightarrow [3,1] \rightarrow [3,2]$, indicating that the head-of-flow packet of flow 1 has timestamp 1 and is stored in buffer 1, the next packet of flow 1 has timestamp 3 and is in buffer 1, and the third has timestamp 3 and is in buffer 2. Note that in the figure there are two p_1^1. They both arrived in time slot 1 on wavelength channel 1, but of different input fibers (hence belong to different flows).

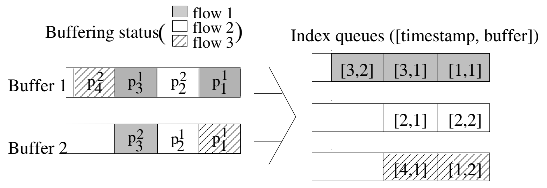

Fig. 2. The relationship between the actual buffer status and the index queues, assuming packet p_i^λ arrived at the switch on wavelength λ in time slot i. The index queues keep the timestamps and buffer indices of the packets being buffered.

In the following sections we will see how the scheduler of the OpCut switch makes the scheduling decision based on the information kept in the queues.

III. Basics of the Scheduler

In this section we introduce the basic packet scheduling process in the WDM OpCut switch. The basic scheduling procedure for a WDM OpCut switch consists of three stages, namely, newly arrived packets cutting-through, receivers picking up packets, and transmitters sending buffered packets to the switch output. Ideally, we would like to send the maximum number of packets to the switch output in each time slot while maintaining packet order. In this paper, we define an optimal schedule to be a schedule that satisfies these two conditions simultaneously.

Maintaining packet order is non-trivial in the OpCut switch, since packets from the same flow may be picked up by different receivers. To deal with this problem, the scheduler adopts a simple strategy: *allow a packet to be scheduled for transmission to the switch output only if it is a head-of-flow packet.* That is, when all packets ahead of it in the same flow have been transmitted, or scheduled for transmission, to the switch output.

The first stage of the scheduling process is to find a matching between the input wavelength channels and the output wavelength channels for newly-arrived packet cut-through. As discussed above, a newly arrived packet of flow f_{ij} is eligible to cut-through the switch only if it is the head-of-flow packet of f_{ij}. In this case, queue F_{ij} must be empty, and there should be no packet of the same flow that arrives on a smaller wavelength channel in the current time slot and has not been scheduled for cut-through. The cut-through process is essentially a matching process between such newly arrived packets (or equivalently, the input wavelength channels these packets are on) and the output wavelength channels. To ensure the cost-effectiveness of the WDM OpCut switch, we assume no wavelength converters at the switch input. As a result, a newly arrived packet on an input wavelength channel can only be sent to the same wavelength channel of its destined output port.

After the cut-through process, packets that cannot cut through need to be picked up by the receivers. This is done by connecting the receivers to the input wavelength channels in a round-robin fashion. At time slot t, the packet from wavelength λ of input fiber i will be sent to receiver r which is given by

$$r = [(i \cdot k + \lambda + t) \mod Nk] + 1$$

Note that instead of a fixed one-to-one connection, the inputs are connected to the receivers in a round-robin fashion such that better load balancing among the receivers can be achieved. As an example, according to our simulation, with non-uniform incoming traffic, if the connection between the inputs and the receivers is fixed, the average packet delay is typically more than 50% longer than that with round-robin connection.

The third stage of the scheduling process is to send packets from the electronic buffers to the output wavelength channels that do not receive a cut-through packet. Again, to maintain packet order, packets in a flow must be transmitted to the switch output in the same order as they arrived. Note that while in each time slot a transmitter can send out up to one packet, an output fiber can take up to k packets, as it has k wavelength channels. If multiple packets from the same flow are scheduled from different buffers to their destined output fiber in the same time slot, they are sent one by one, in the same order as they arrived, to the smallest wavelength that is still available. Finding good matchings while maintaining packet order is the main challenge of the scheduling problem, which will be discussed in detail next.

IV. Packet Scheduling Algorithms

As discussed in the previous section, the optimal schedule consists of two parts, namely, the matching between the input wavelength channels and the output wavelength channels for newly-arrived packet cut-through, and the matching between the transmitters and the output wavelength channels for the transmission of buffered packets. We denote the first matching as M_c and the second matching as M_b.

In the following, we first show that finding M_c and finding M_b can be converted into the same problem, which we refer to as the Maximum-Coverage Prefixes problem. We then prove that this problem is NP-hard, and give an approximation algorithm with performance ratio. We also discuss the implementations of the approximation algorithm in the high-speed switch,

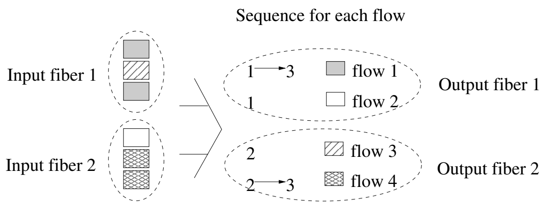

Fig. 3. An example of converting packet arrivals to sequences. Flow 1 contains packets from input fiber 1 to output fiber 1. There are two packet arrivals for flow 1 on wavelengths 1 and 3, respectively. Therefore the sequence for flow 1 is $\langle 1 \rightarrow 3\rangle$. Similarly a sequence can be determined for each of other three flows.

and other possible variations of the algorithm to reduce the complexity.

A. Problem Formalization

In this section we present the problem formalization of finding M_c and M_b. We start with a discussion on finding M_c.

A.1 Finding M_c

To compute M_c, what we have are the packet arrivals on each input wavelength channel in the current time slot. Recall that there is no wavelength conversion available at the input of the OpCut switch, thus during the cut-through process a packet that arrives on wavelength channel λ can only be sent to wavelength channel λ of its destined output fiber.

For each flow of packets from an input fiber to an output fiber, we can define a sequence of wavelengths. The sequence contains in ascending order all the wavelengths on which there is a packet arrival that belongs to the corresponding flow in the current time slot. If there is no packet arrival for a flow, the corresponding sequence is empty. Moreover, each wavelength can appear no more than once in a sequence, since in each time slot there is at most one packet arrival on each wavelength channel.

In the simple example shown in Fig. 3, there are two input fibers and two output fibers, each containing three wavelength channels. We assume flow 1 contains packets from input fiber 1 to output fiber 1. There are two packet arrivals for flow 1 on wavelengths 1 and 3, respectively. Therefore the sequence for flow 1 is $\langle 1 \rightarrow 3\rangle$. Similarly a sequence can be determined for each of other three flows.

It is not difficult to see that the packet cut-through for different output fibers is independent. Therefore in the following we focus on one output fiber and only consider packets destined for this specific output fiber.

To find M_c, we need to let the maximum number of packets that arrive in the current time slot cut through without violating the packet order. Recall that if two packets belong to the same flow and arrive in the same time slot, then the one arrives on the smaller wavelength channel is defined in front of the other in the flow. In other words, packets of the same flow must cut through in the same order as their arrival wavelengths appear in the sequence. If a packet cannot cut through, then all packets behind it should never cut through in the current time slot. For example, for flow 1 in Fig. 3, if the packet on wavelength 1 cannot cut through, then the packet on wavelength 3 should not cut through, even if wavelength 3 of output fiber 1 is available.

We define a *prefix* of a sequence as a (possibly null) segment of the sequence that starts from the head of the sequence. Consequently, letting the maximum number of packets cut through without violating the packet order is equivalent to selecting a prefix from each sequence, such that these prefixes contain the maximum number of elements (wavelengths). The restriction that each wavelength channel of an output fiber can take at most one packet in a time slot means that a wavelength can appear at most once in all these prefixes. To sum up, finding M_c for a specific output fiber can be formalized into the following problem:

Input: A collection of sequences of elements. No element is repeated in a single sequence.

Output: A prefix from each sequence, such that the maximum number of elements is covered by all the prefixes, and no element appears in more than one prefix.

We name it the Maximum-Coverage Prefixes (MCP) problem. As will be shown by the NP-hard proof later, the fact that the elements in each sequence are sorted does not reduce the complexity of the problem, hence is omitted.

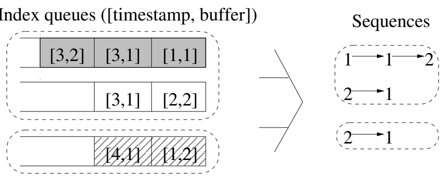

Fig. 4. The relationship between index queues and sequences. An index queue keeps the timestamps and buffer indices of the packets being buffered. A sequence keeps the buffer indices only. Both the index queues and sequences are grouped according to the destined output fiber of the packets.

A.2 Finding M_b

In this subsection we present the formalization of the problem of finding M_b. To compute M_b, the information we have is an index queue for each flow that contains the timestamp and buffer index of each packet being buffered. The timestamp of a packet is required to locate the packet in a buffer. However, for the purpose of computing M_b, all we need to know is the *order* of the packets being buffered, and in which buffer each of these packets can be located. Since the order of packets is naturally reflected by the order of their appearance in the index queues, the timestamps of packets are not needed to compute M_b. As a result, we can remove the timestamps from an index queue and simplify it into a sequence of buffer indices. Fig. 4 continues the example in Fig. 2 and shows how the index queues are converted to sequences. For instance, the index queue for flow 1 is $[1, 1] \rightarrow [3, 1] \rightarrow [3, 2]$, which stores both the timestamps and buffer indices of the packets of flow 1 that are currently being buffered. However, to compute M_b, it is sufficient to know that the head-of-flow packet of flow 1 is in buffer 1, the next packet of flow 1 is in buffer 1 as well, and the third is in buffer 2. That is exactly what the corresponding sequence $\langle 1 \rightarrow 1 \rightarrow 2\rangle$ tells. In the following, we interchangeably use index queue and sequence when there is no ambiguity.

Our goal is to send as many packets as possible from the buffers to the switch output. Mapped to sequences, it is equivalent to select a portion from each sequence , such that the maximum number of buffer indices is covered by the overall selection. For example, in Fig. 4, if buffer indices 1 and 2 are selected from the sequence for flow 1 ($\langle 1 \rightarrow 1 \rightarrow 2\rangle$) and flow

2 ($\langle 2 \rightarrow 1 \rangle$), respectively, it means that a packet of flow 1 currently in buffer 1, and a packet of flow 2 currently in buffer 2, can be transmitted to the respective output fibers. The requirement on packet order means that we can only select a prefix from each sequence. For instance, in the above example it is not legal to select buffer 1 from the sequence for flow 2 without selecting buffer 2 from the same sequence, since p_3, the packet of flow 2 in buffer 1, cannot be scheduled before p_2 in buffer 2, which also belongs to flow 2 and is older than p_3. Besides, the fact that at most one packet can be retrieved from a buffer and transmitted in a time slot implies that no buffer index can appear more than once in the prefixes selected. It also indicates that, if a buffer index appears multiple times in a single sequence, it is safe to consider only the segment of the sequence before the second appearance of that buffer index. In terms of the example above, the first sequence, $\langle 1 \rightarrow 1 \rightarrow 2 \rangle$ can be shortened to $\langle 1 \rangle$, because at most one of p_1 and p_5 can be retrieved from buffer 1 in a single time slot. Therefore the maximum length of a sequence is Nk, equal to the total number of buffers. Also, considering the number of available wavelength channels, if we group the sequences according to the destined output fiber of the corresponding index queues of flows, then it implies that the prefixes of all the sequences in group j should cover at most c_j buffer indices, where c_j is the number of available wavelength channels on output fiber j.

To sum up, the original problem of finding matching M_b can be formalized into the following problem:

Input: N groups of sequences. Each sequence contains at most Nk elements and no duplicated element. There are also N integers $c_1, c_2, \dots, c_N$, all in the range of $[0, k]$, where N and k are arbitrary positive integers.

Output: A prefix from each sequence, such that the maximum number of elements is covered by all the prefixes, and no element appears in more than one prefix. In addition, the prefixes of all the sequences from group j should cover no more than c_j elements.

We call this converted problem the "Constrained Prefix Coverage" (CPC) problem. This problem is more difficult than finding M_c, as the scheduling of packets from the buffers to different output fibers is dependent. In fact, it can be shown that the MCP problem is a simple special case of the CPC problem.

The special case of the CPC problem we consider is when $c_1 = k, c_2 = c_3 = \cdots = c_N = 0$. In terms of the original scheduling problem, this is the case when, for example, the cut-through packets occupy all of the output wavelength channels except those on output fiber O_1. In this case, only group 1 of the sequences, i.e., those for flows destined for output fiber 1, needs to be taken into consideration. As a result, the CPC problem is simplified into the MCP problem.

Next we will show that the MCP problem is not only NP-hard, but also inapproximable within any constant factor in polynomial time.

B. NP-Hardness and Inapproximability Proof

The following theorem states the NP-hardness of the MCP problem.

Theorem 1: The MCP problem is NP-hard.

Proof: We show that the MCP problem is NP-hard by reducing the set packing problem [8] to the MCP problem. An instance of the set packing problem can be expressed as follows. Given a finite set U and n subsets of U, $\mathcal{S} = \{S_i \mid S_i \subseteq U, i = 1, 2, \dots, n\}$, does there exist m pairwise disjoint (that is, containing no common elements) sets in $\mathcal{S}$? The set packing problem is NP-hard when each subset of U in $\mathcal{S}$ contains as few as 3 elements. In this instance, without affecting the correctness of the proof we assume the maximum cardinality of S_i for $i = 1, 2, \dots, n$, denoted as L, is no more than n.

We start the proof by converting each set in $\mathcal{S}$ into a sequence in the MCP problem. Initially all sequences are empty. They are constructed in three steps:

- *Step 1 - Set padding.* We pad each of the sets in $\mathcal{S}$ such that all of them have cardinality equal to L after padding. The newly added elements are all unique and are not in any of the original sets.
- *Step 2 - Adding intersection indicators to sequences.* For each pair of sets in $\mathcal{S}$, if their intersection is non-empty, we add a new element to the heads of the corresponding pair of sequences. The newly added element serves as an indicator of intersection. Similar to Step 1, the elements added to different pairs of sequences are unique, and are not in any of the padded sets. Let x denote the total number of unique elements added to the sequences in this step. Since there are $n(n-1)/2$ pairs of sets, $x \leq n(n-1)/2$ must hold.
- *Step 3 - Appending expanded sets to sequences.* Now for each padded set, we do an $(x+1)$-time-expansion. That is, we replace each element in the set with $x + 1$ new, unique elements. After that, all elements from each set are appended to the tail of the corresponding sequence.

The set-to-sequence conversion can be completed in polynomial time. Fig. 5 provides an example of this conversion process, where, for convenience, the numbers of elements in the sets are small, but the process can be applied to larger sets. Denote the sequence resulted from S_i as Q_i, $i = 1, 2, \dots, n$. It can be seen that Q_i consists of two parts. The first part contains the intersection indicators and the second part contains the elements from the set after padding and expanding. The minimum length of Q_i is $(x+1)L$, which occurs when S_i does not intersect with any other set in $\mathcal{S}$ and no intersection indicator has been added to Q_i. It is also clear that Q_i and Q_j contain no common element if and only if S_i and S_j do not intersect.

Next we show that the original set packing problem has a solution if and only if the MCP problem, given the resulted sequences as input, has a solution that covers at least $(x+1)Lm$ elements.

First, assume that the set packing problem has a solution $\mathcal{S}^*$. That is, $\mathcal{S}^* \subseteq \mathcal{S}$ contains m pairwise disjoint sets. Denote the collection of the sequences corresponding to the subsets in $\mathcal{S}^*$ as $\mathcal{Q}^*$. It must be true that the sequences in $\mathcal{Q}^*$ are pairwise disjoint. As a result, the sequences in $\mathcal{Q}^*$ cover at least $(x+1)L|\mathcal{S}^*| \geq (x+1)Lm$ elements.

Next, denote a solution to the MCP problem as $\mathcal{P} = \{P_i^* \mid i = 1, 2, \dots, n\}$, where P_i^* is a prefix of Q_i. Note that P_i^* may be null for some i. We claim that for all i, either $P_i^* = Q_i$, or P_i^* ends before the second part of Q_i. To see this, suppose that there exists some j such that P_j^* ends in the middle of part 2

Sets	Sequences	Sets	Sequences	Sets	Sequences	Sets	Sequences
{a, b, c}	$\langle\rangle$	{a, b, c}	$\langle\rangle$	{a, b, c}	$\langle i_{12} \rightarrow i_{13}\rangle$	{a, b, c}	$\langle i_{12} \rightarrow i_{13} \rightarrow a_1 \rightarrow a_2 \rightarrow a_3 \rightarrow b_1 \rightarrow b_2 \rightarrow b_3 \rightarrow c_1 \rightarrow c_2 \rightarrow c_3\rangle$
{c, a}	$\langle\rangle$	{c, a, z}	$\langle\rangle$	{c, a, z}	$\langle i_{12}\rangle$	{c, a, z}	$\langle i_{12} \rightarrow c_1 \rightarrow c_2 \rightarrow c_3 \rightarrow a_1 \rightarrow a_2 \rightarrow a_3 \rightarrow z_1 \rightarrow z_2 \rightarrow z_3\rangle$
{b, d}	$\langle\rangle$	{b, d, y}	$\langle\rangle$	{b, d, y}	$\langle i_{13}\rangle$	{b, d, y}	$\langle i_{13} \rightarrow b_1 \rightarrow b_2 \rightarrow b_3 \rightarrow d_1 \rightarrow d_2 \rightarrow d_3 \rightarrow y_1 \rightarrow y_2 \rightarrow y_3\rangle$
{e}	$\langle\rangle$	{e, x, w}	$\langle\rangle$	{e, x, w}	$\langle\rangle$	{e, x, w}	$\langle e_1 \rightarrow e_2 \rightarrow e_3 \rightarrow x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow w_1 \rightarrow w_2 \rightarrow w_3\rangle$
(a)		(b)		(c)		(d)	

Fig. 5. An example of converting sets to sequences. (a) The original sets. The sequences are initially null. (b) Step 1: Set padding. w, x, y and z are the newly added elements. (c) Step 2: Adding intersection indicators to sequences. A new element i_{12} is added to both sequence 1 and sequence 2 as an indication of intersection of set 1 and set 2. Similarly, i_{13} is added to sequence 1 and sequence 3. (d) Step3: Appending expanded sets to sequences. Since two intersection indicators were used in Step 2, each set is 3-time expanded then appended to the corresponding sequence.

of Q_j, right before element b. Let $Q_{j'}$ denote another sequence that also contains b. The fact that b is in part 2 of both sequences implies that the original sets S_j and $S_{j'}$ intersect. Thus an intersection indicator, denoted as a', must have been added to both Q_j and $Q_{j'}$. Since P_j^* includes the entire part 1 of Q_j, it contains a'. As a result, $P_{j'}^*$ has to end somewhere in $Q_{j'}$ before b. Otherwise it must contain a' and violate the "no repeat" condition, since a' is ahead of b in $Q_{j'}$. It follows that b is not included in any prefix in $\mathcal{P}$. Hence, by extending P_i^* to include b, a better solution is obtained for the MCP problem. This contradicts with the assumption that $\mathcal{P}$ is optimal, thus the claim justified.

Now assume that $\mathcal{P}$ covers $(x+1)Lm$ or more elements. Then in $\mathcal{P}$ there exist at least m prefixes each of which spans the whole corresponding sequence. The reason is that, as proved above, if a prefix P_i^* in $\mathcal{P}$ is not equal to Q_i, it must end before the second part of Q_i. In other words, it covers at most all the intersection indicators in Q_i. Overall, all such prefixes can cover at most x elements, the total number of intersection indicators. Hence if the number of prefixes in $\mathcal{P}$ that cover the whole sequence is less than m, at most $x + (x+1)L(m-1) \leq (x+1)Lm - 1$ elements can be covered. Therefore, given that $\mathcal{P}$ covers $(x+1)Lm$ or more elements, it must contain at least m full sequences. Besides, these m sequences must not contain any common elements, which implies that the corresponding m original sets in $\mathcal{S}$ form a solution to the set packing problem.

In conclusion, since set packing is NP-hard and is reducible to the MCP problem, the MCP problem is also NP-hard. ■

It can be further proved that MCP cannot be approximated in polynomial time within any constant factor. We prove it by showing that MCP is as difficult to approximate as maximum set packing, the optimization version of the set packing problem.

Theorem 2: The MCP problem cannot be approximated within any constant factor in polynomial time unless P = NP.

Proof: Let $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ be an instance of the maximum set packing problem. That is, $\mathcal{S}$ is a collection of finite sets, and we wish to determine the maximum number of pairwise disjoint sets in $\mathcal{S}$. Assume that there is a polynomial time approximation algorithm with a constant approximation ratio ρ for MCP. We convert each set in C into a sequence of the MCP problem as we have done in the proof of Theorem 1. However, instead of expanding the padded sets $x+1$ times, they are expanded $t \geq n(n-1)/(\rho L)$ times. Then by applying this algorithm to the resulted sequences, we have

$$APX_{MCP} \geq \rho \cdot OPT_{MCP}$$

where APX_{MCP} is the number of elements covered by the approximation algorithm, and OPT_{MCP} is that of an optimal solution. Note that as previously discussed, a solution to the MCP problem, optimal or approximate, can be divided into two parts: the part that covers the intersection indicators, and the other part that covers the elements in the sets after padding and expanding. Suppose that the approximate solution covers N_1 intersection indicators and N_2 set elements. Immediately we have $0 \leq N_1 \leq x$. Besides, if an optimal solution of the maximum set packing problem contains OPT_{SP} sets, then the corresponding OPT_{SP} sequences in MCP are pairwise disjoint. Hence an optimal solution of MCP will contain OPT_{SP} full sequences, each of length at least Lt. That is,

$$OPT_{MCP} \geq L \cdot t \cdot OPT_{SP}$$

Therefore, from $APX_{MCP} = N_1 + N_2 \geq \rho OPT_{MCP}$ it can be derived that

$$\begin{aligned} N_2 &\geq \rho OPT_{MCP} - N_1 \\ &\geq \rho Lt \cdot OPT_{SP} - x \\ &\geq \rho Lt \cdot OPT_{SP} - \frac{n(n-1)}{2} \end{aligned}$$

L and n are known once the instance is given, and ρ is a constant. Then

$$\begin{aligned} \frac{N_2}{Lt} &\geq \rho \cdot OPT_{SP} - \frac{n(n-1)}{2Lt} \\ &\geq \rho \cdot OPT_{SP} - \frac{\rho}{2} \quad \text{(since we choose } t \geq n(n-1)/(\rho L)\text{)} \\ &\geq \frac{\rho}{2} \cdot OPT_{SP} \end{aligned}$$

The last step holds since $OPT_{SP} \geq 1$ is always true as long as $\mathcal{S} \neq \emptyset$.

The approximate solution contains a prefix of each sequence, some of which may be null. Let $\tilde{n}$ denote the number of sequences whose prefix in the approximate solution contains at least one element of the second part of the sequence. Then $\tilde{n} \geq N_2/(Lt)$ holds since each sequence has Lt part-2 elements, and N_2 is the total number of part-2 elements covered by the approximation algorithm. In addition, any two among the $\tilde{n}$ sequences do not contain any common element, since the whole part 1 of all these sequences is included in the solution. Therefore, the corresponding $\tilde{n}$ sets in $\mathcal{S}$ are pairwise disjoint. If we select these $\tilde{n}$ sets, we have found an approximate solution to the original maximum set packing problem such that

$$APX_{SP} = \tilde{n} \geq \frac{N_2}{Lt} \geq \frac{\rho}{2} \cdot OPT_{SP}$$

This contradicts the fact that maximum set packing has no constant factor polynomial time approximation algorithm [8]. Putting all of these together, MCP cannot be approximated within any constant factor in polynomial time. ■

That completes our proof of the NP-hardness and inapproximability of the MCP problem, which leads to the NP-hardness and inapproximability of the CPC problem and the original scheduling problem. In the following we will propose an approximation algorithm with approximation ratio $\sqrt{2Nk}$. We call this algorithm the Longest-Or-Heads (LOH) algorithm.

C. The Longest-or-Heads (LOH) Approximation Algorithm

We first introduce the concept of *effective length* of a sequence. Recall that the maximum number of packets that can be transmitted from the buffers to output fiber O_j, or in terms of the CPC problem, the maximum number of elements that can be covered by the sequences in group j, is limited by c_j, the number of available wavelength channels on O_j. Hence here we define the effective length of a sequence in group j to be the minimum of its actual length and c_j. For simplicity, in the following we abuse the phrase "longest sequence" a bit and use it to refer to the sequence with the maximum effective length. Also we say a packet is on a sequence if the information of the packet is being kept in the corresponding index queue.

The basic idea of the LOH Algorithm is simple and exactly as its name suggests: in each time slot, depending on which ends up with more packets scheduled, we either schedule all the packets on the longest sequence, or consider only packets at the head of the sequences and schedule as many of them as possible.

Next we derive the approximation ratio of the LOH algorithm. Denote the length of the longest sequence as n_l. Denote the size of a maximal matching between the transmitters and the output wavelengths as n_h, when only head-of-sequence packets are considered. Note that we consider a maximal matching, not a maximum matching here, because a maximal matching is much easier to find in practice than a maximum matching. Also, using a maximal matching changes the performance ratio by a constant factor comparing to using a maximum matching. Let C^* denote an optimal solution and let $|C^*|$ denote the number of packets scheduled according to C^*. As n_h is the size of a maximal matching, the size of a maximum matching with the same input can be at most $2n_h$ [15]. Since an arbitrary solution can contain packets from at most $2n_h$ sequences when only the heads of sequences are considered, it can contain packets from at most $2n_h$ sequences when the full sequences are considered. Moreover, the number of packets on a single sequence contained in any solution, including C^*, is no more than n_l, which is the maximum length of all sequences. Thus

$$|C^*| \leq n_l \cdot 2n_h \leq 2C_L^2$$

where $C_L = \max\{n_l, n_h\}$ is the number of packets scheduled by the LOH algorithm. Consequently

$$\sqrt{\frac{|C^*|}{2}} \leq C_L \leq |C^*|$$

Back to our original packet scheduling problem, $|C^*|$ can be at most Nk, the total number of output wavelength channels. Thus

$$\max\left\{\frac{C_L}{|C^*|}, \frac{|C^*|}{C_L}\right\} \leq \sqrt{2Nk}$$

which is the approximation ratio of the LOH algorithm. It may be of some interest to note that the result is consistent with the fact that the best known algorithm approximates the maximum set packing problem within a factor of $O(\sqrt{|U|})$, where U is the underlying base set [8].

Now consider the optimal scheduling problem as a whole. We will adopt a scheduler that first lets as many as possible newly arrived packets cut-through, i.e., finds M_c. Then the scheduler runs the LOH algorithm with the output wavelength channels that are still available to approximate M_b. The number of packets scheduled overall is at least

$$\frac{|M_c|}{\sqrt{2Nk}} + \frac{|M_b|}{\sqrt{2Nk}} \geq \frac{|M_c \cup M_b|}{\sqrt{2Nk}}$$

By definition $M_c \cup M_b$ is an optimal schedule. Thus we have

Theorem 3: The LOH algorithm approximates the optimal scheduling problem within a factor of $\sqrt{2Nk}$.

D. Implementation of the LOH Algorithm

The LOH algorithm is implementable in hardware. The overall longest sequence can be determined by a linear scan over all sequences. However, to make it more practical we will parallelize the process in a way similar to the Prioritized iSLIP [11]. An arbiter is assigned to each output fiber, as well as to each buffer. An output first determines the longest sequence within its group. Then it sends a request to a buffer if the buffer appears in the sequence. The priority level of a request is the length of the corresponding sequence. If a buffer receives any requests, it selects the one with the highest priority to grant. If there are multiple requests with the same priority level, an even break is needed. This is handled by maintaining a pointer to a round-robin schedule at each buffer. These pointers are synchronized initially and shift one position in each time slot, such that they always point to the same location of the round-robin schedule. By doing so, it is ensured that in case there are two sequences of the same length, all buffers that receive requests from both sequences will select the same sequence to grant. Therefore, at least for one of the longest sequences, all of its requests will be granted. Note that a sequence cannot be the (elected) global longest sequence if it misses a single grant. Thus if all requests from a sequence are granted, the sequence accepts all of the grants. Otherwise it accepts *no* grant. Strictly speaking, packets on a sequence can be sent as long as a prefix of a sequence is granted. However, to keep the scheduler simple we do not distinguish between whether it is a prefix of a sequence that has been granted or not. When the iteration ends, the number of accepted grants is recorded as n_l.

Next, the scheduler tries the second option to schedule the packets at the head of sequences only by finding a maximal matching through the iSLIP algorithm.

- *Step 1 - Request.* Each output fiber, if still having available wavelength channels, sends a request to every buffer that appears at the head of some sequence in the group of sequences destined for this output.
- *Step 2 - Grant.* Besides the longest sequence pointer, buffer i maintains a second round robin pointer p_i. When it receives any requests, buffer i selects the one that appears next in the round robin schedule from the position pointed to by p_i. p_i is updated to one position beyond the granted output if and only if the grant is accepted in Step 3.
- *Step 3 - Accept.* If an output fiber with c available wavelength channels receives c' grants, it chooses $\min\{c, c'\}$ to

accept. That is, by maintaining a round-robin pointer, the output picks the first $\min\{c, c'\}$ elements that appear next in the round robin schedule from the position pointed by the pointer, and accepts the grants from the corresponding buffers. Then it increments its round-robin pointer by one beyond the first granted buffer.

Steps 1 and 3 differ from the original iSLIP algorithm in that each output fiber can take multiple grants in a single iteration. On average, within $\log(Nk)$ iterations, the result converges to a maximal matching, and n_h is the total number of accepted grants.

When both n_l and n_h are obtained, the scheduler simply chooses the better one as the final schedule to execute.

E. Variations of LOH

In this subsection we discuss some possible variations of the LOH algorithm. The first possible improvement is to take more packets into consideration, instead of only the packets that were at the head of the sequences when the scheduling process began. Note that it is safe to schedule a packet if all the packets ahead of it in the same sequence have been scheduled and it becomes the head-of-flow packet.

To incorporate this idea into the original LOH algorithm, the first and the third step of each iteration of the LOH algorithm are modified as follows.

- *Step 1 - Request.* Each output fiber, if still having available wavelength channels, sends a request to every buffer that has a head-of-flow packet destined for this output fiber.
- *Step 3 - Accept.* Whenever a grant is accepted, the head of the corresponding sequence is removed, and the next packet in the sequence becomes the new head-of-flow packet.

This modified version is referred to as Variation 1 of LOH later in the simulation section. Second, note that making the longest sequence as an option for scheduling has its importance, especially theoretically, since it is necessary in terms of achieving the approximate ratio. Nevertheless, it has potential drawbacks because it increases the complexity of the scheduler. Therefore, another possible variation is to eliminate the "longest sequence" option, which leads to a simpler scheduler. It is referred to as Variation 2 of LOH in the following. The performance of these variations is compared with that of the original LOH algorithm through simulations.

V. Performance Evaluation

In this section we present the simulation results. We conducted simulations under two traffic models that are widely used in the performance evaluation of high-speed switches, for example, in [2], [14]. The first model is Bernoulli uniform traffic, which assumes the packet arrival at each input wavelength channel is a Bernoulli process and the destinations of packets are uniformly distributed over all output fibers. The second model is burst non-uniform traffic. Under this model, the status of an input alternates between "on" and "off." The length of a state follows a geometric distribution. A packet arrives at an input wavelength channel at the beginning of a time slot if and only if the input wavelength channel is "on." The packets arrived during an on state have the same destination and form a burst. In addition, output fiber O_i is assumed to be the "hotspot" for input fiber I_i. A certain portion, denoted as p_h, out of the entire traffic from an input fiber is dedicated to the corresponding hotspot output fiber. The remaining traffic is uniformly distributed to all other output fibers. In our simulations, p_h is set to 50%, the value that results in the worst performance according to [14].

The WDM OpCut switch is simulated with $N = 64$ and $k = 8$. The LOH algorithm, as well as its variations as discussed in Section IV-E, is implemented with 8-iteration iSLIP. Each simulation was run for 10^6 time slots. The main performance criteria considered include the packet cut-through ratio and the average packet delay.

A. Cut-Through Ratio

The packet cut-through ratio is an important criterion of the WDM OpCut switch. The higher the ratio, the more packets are sent directly to the switch output and avoid the O/E/O conversion. Fig. 6 plots the cut-through ratio under different schedulers and traffic models. It can be seen that under Bernoulli uniform traffic, the three schedulers lead to almost identical cut-through ratio. The ratio is as high as 0.9 when the load is around 0.2, and remains above 0.6 when the load is increased to 1 for LOH and LOHV1 . Under burst non-uniform traffic, the cut-through ratio drops faster with the increase in load. Nevertheless, still more than 30% of the packets can cut-through even when the switch is fully loaded. Besides, LOHV1 results in higher cut-through ratio than other two schedulers under burst non-uniform traffic, which shows the effect of taking more packets into consideration besides the sequence heads when scheduling.

We notice that under Bernoulli uniform traffic, there is a sharp drop in the cut-through ratio for LOHV2, which occurs around 0.95 load. As will be confirmed shortly by the average packet delay, these are the points at which the OpCut switch is saturated when LOHV2 is used as the scheduler.

B. Average Packet Delay

The average packet delay in a WDM OpCut switch is simulated and compared with the single-wavelength case and the ideal WDM output-queued (OQ) switch. Single wavelength can be considered as a special case of WDM with $k = 1$. In a single-wavelength OpCut switch, at most one packet can be transmitted to an output port in each time slot, hence it is sufficient to consider only the sequence heads for scheduling. For the WDM OQ switch, full wavelength conversion capability and N-speedup memory are assumed, such that a newly arrived packet can be wavelength-converted and directly sent to an arbitrary wavelength channel of its destined output fiber, and the buffer on each output wavelength channel can receive as many as N packets in a single time slot. Note that the ideal WDM OQ switch architecture is impractical; we take it into consideration merely because its average packet delay serves as the lower bound for other switches and schedulers.

It can be seen from Fig. 7 that the packet delay in the WDM OpCut switch is significantly shorter than that of the single wavelength case. In other words, WDM not only multiplies the bandwidth but is also able to reduce packet delay. This is due to the fact that a packet sent to the electronic buffer in a WDM OpCut switch can choose from multiple candidate output wavelength channels. Similar to the cut-through ratio, under Bernoulli uniform traffic the three schedulers for the WDM

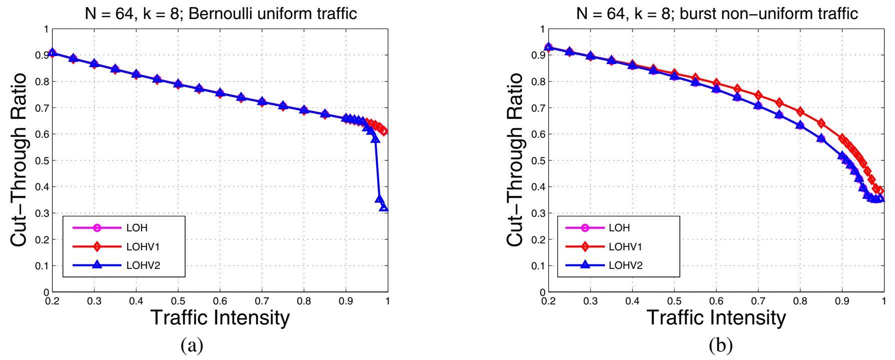

Fig. 6. Packet cut-through ratio under different schedulers and traffic models. LOH: scheduler that executes the LOH algorithm. LOHV1: Variation 1 of LOH. LOHV2: Variation 2 of LOH.

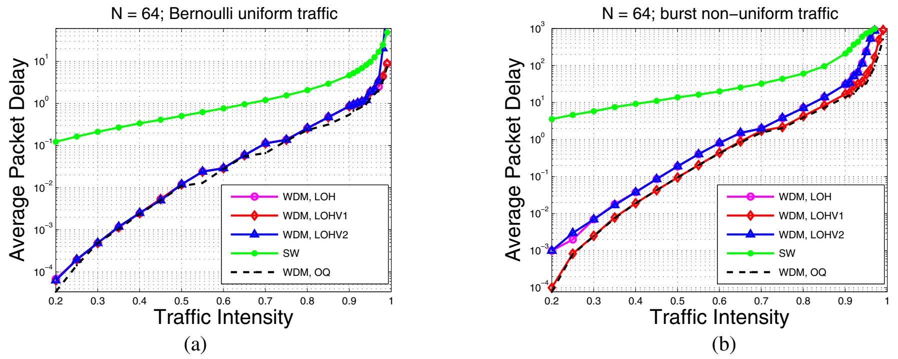

Fig. 7. Average packet delay under different schedulers and traffic models. The notations of schedulers are the same as in Fig. 6. WDM: the multi-wavelength scenario, as opposite to single wavelength (SW). OQ: ideal output-queued switch.

OpCut switch achieve almost the same average packet delay when the load is below 0.95 , which is also very close to that of the ideal WDM OQ switch. Under burst non-uniform traffic, LOHV1 outperforms the other two schedulers. This shows that considering only the sequence heads for scheduling is sufficient under Bernoulli uniform traffic, but leaves room for improvement under burst non-uniform traffic. There is no significant difference between the performance of LOH and LOHV2 when the load is below 0.95, On the other hand, the performance of LOH and LOHV2 is indistinguishable in all scenarios, which implies that in practice the scheduler can be effectively simplified without affecting the system performance by not looking for the longest sequence.

VI. Conclusions

In this paper, we have studied packet scheduling in the low-latency OpCut switch with Wavelength Division Multiplexing. Our goal of optimal scheduling is to maximize the throughput in each time slot while maintaining packet order. We formalized this problem as the Maximum-Coverage Prefixes (MCP) problem and proved its NP-hardness and inapproximability. We designed an approximation algorithm for the MCP problem which approximates the optimal scheduling algorithm with a factor of $\sqrt{2Nk}$ with regard to the number of packets transmitted. Based on the approximation algorithm, we proposed practical schedulers and discussed hardware implementations. Simulation results show that these schedulers achieve very good performance under various traffic models.

References

[1] K.J. Barker, A. Benner, R. Hoare, A. Hoisie, A.K. Jones, D.J. Kerbyson, D. Li, R. Melhem, R. Rajamony, E. Schenfeld, S. Shao, C. Stunkel and P. Walker, "On the feasibility of optical circuit switching for high performance computing systems," *ACM/IEEE Conference on Supercomputing (SC '05)*, Seattle, WA, Nov. 2005.

[2] I. Iliadis and C. Minkenberg, "Performance of a speculative transmission scheme for scheduling-latency reduction," *IEEE/ACM Trans. Networking*, vol. 16, no. 1, pp. 182-195, Feb. 2008.

[3] R.R. Grzybowski, B.R. Hemenway, M. Sauer, C. Minkenberg, F. Abel, P. Muller and R. Luijten "The OSMOSIS optical packet switch for supercomputers: Enabling technologies and measured performance," *Proc. Photonics in Switching 2007*, pp. 21-22, Aug. 2007.

[4] R. Hemenway, R.R. Grzybowski, C. Minkenberg and R. Luijten, "Optical-packet-switched interconnect for supercomputer applications," *Journal of Optical Networking*, vol. 3, no. 12, pp. 900-913, Dec. 2004.

[5] Z. Zhang and Y. Yang, "Performance analysis of optical packet switches enhanced with electronic buffering," *23th IEEE International Parallel and Distributed Processing Symposium (IPDPS '09),* May 2009.

[6] L. Liu, Z. Zhang and Y. Yang, "Packet scheduling in a low latency optical packet switch," *11th IEEE International Conference on High Performance Switching and Routing (HPSR 2010)*, Texas, June 2010.

[7] D. Staessens, D. Colle, U. Lievens, M. Pickavet, P. Demeester, W. Colitti, A. Nowe, K. Steenhaut, R. Romeral, " Enabling high availability over multiple optical networks," *IEEE Communications Magazine*, pp. 120-126, June 2008.

[8] Viggo Kann, "Maximum Set Packing,"*A compendium of NP optimization problems.* www.nada.kth.se/ viggo/wwwcompendium/node144.html.

[9] N. McKeown, A. Mekkittikul, V. Anantharam and J. Walrand, "Achieving 100% throughput in an input-queued switch," *IEEE Trans. Communications*, vol. 47, no. 8, pp. 1260-1267, Aug. 1999.

[10] T. Anderson, S. Owicki, J. Saxe and C. Thacker, "High speed switch scheduling for local area networks," *ACM Trans. Computer Systems*, pp. 319-352, Nov. 1993.

[11] N. McKeown, "The *i*SLIP scheduling algorithm for input-queued switches,"*IEEE/ACM Trans. Networking*, vol. 7, no. 2, Apr. 1999.

[12] C.-S. Chang, D.-S. Lee, Y.-J. Shih and C.-L Yu, "Mailbox switch: a scalable two-stage switch architecture for conflict resolution of ordered packets,"*IEEE Trans. Communications*, vol. 56, no. 1, pp. 136-149, 2008.

[13] I. Keslassy and N. McKeown, "Maintaining Packet Order in Two-Stage Switches," *IEEE INFOCOM 2002*, vol. 2, pp. 1032-1041, Jun. 2002.

[14] R. Rojas-Cessa, E. Oki, Z. Jing and H. Chao, "CIXB-1: Combined input-one-cell-crosspoint buffered switch," In Proc. *2001 IEEE Workshop on High-Performance Switching and Routing (HPSR 2001)*, pp. 324-329, Dallas, TX, May 2001.

[15] T.H. Cormen, C.E. Leiserson, R.L. Rivest and C. Stein, *Introduction to Algorithms,* 2nd edition, The MIT Press, Sep. 2001.