Packet Scheduling in Optical Switches and Interconnects

A Dissertation Presented

by

Lin Liu

to

The Graduate School

in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

in

Electrical Engineering

Stony Brook University

May 2011

Stony Brook University

The Graduate School

Lin Liu

We, the dissertation committee for the above candidate for the Doctor of Philosophy degree, hereby recommend acceptance of this dissertation.

Yuanyuan Yang – Dissertation Advisor
Professor, Department of Electrical and Computer Engineering

Sangjin Hong – Chairperson of Defense
Associate Professor, Department of Electrical and Computer Engineering

Alex Doboli
Associate Professor, Department of Electrical and Computer Engineering

Esther M. Arkin
Professor, Department of Applied Mathematics & Statistics

This dissertation is accepted by the Graduate School.

Lawrence Martin
Dean of the Graduate School

Abstract of the Dissertation

Packet Scheduling in Optical Switches and Interconnects

by

Lin Liu

Doctor of Philosophy

in

Electrical Engineering

Stony Brook University

2011

Optical interconnects and switches are widely considered as a promising candidate to provide high and ultra-high speed interconnection. This thesis addresses several important issues and proposes solutions in packet scheduling and performance evaluation for various optical switching architectures, including: (1) optimal packet scheduling for output-buffered optical switches with limited-range wavelength conversion capability; (2) admissible traffic and maximum throughput for input-buffered optical switches; (3) packet scheduling in single-wavelength and wavelength-division-multiplexed (WDM) OpCut switches, a low-latency optical-electronic hybrid switch architecture; (4) energy-aware routing in hybrid optical networks-on-chip (NoC).

In recent years, switches and interconnects draw increasingly more attention due to the fact that they tend to become a bottleneck at all levels: intra-chip, chip-to-chip, board level, and computer networks. There are many requirements posed on an interconnect network, such as low latency, high throughput, low error rate, low power consumption, as well as scalability. Finding a solution that can satisfy all these needs is a non-trivial task.

Due to the huge bandwidth and low error rate, optical interconnects and switches are widely considered as a promising candidate for future high and ultra-high speed interconnect networks. As key topics in the development of optical switching networks, scheduling and performance evaluation have been attracting considerable research interest. While there has been extensive research in these fields for electronic networks, most of them cannot be directly leveraged for optical networks - on one hand, some components are still missing in optical domain, for example optical random access memory (RAM); on the other hand, the solutions for electronic networks do not take into consideration unique characteristics of optics, such as the capability of wavelength conversion.

This dissertation addresses several important issues and proposes solutions in packet scheduling and performance evaluation for various optical switching architectures, including (1) the Augment to Full packet scheduling algorithm that maximizes throughput and minimizes average queuing delay simultaneously for output-buffered optical switches, making use of limited-range wavelength conversion; (2) a new fiber-delay-line (FDL) based input buffering fabric that is able to provide flexible buffering delay, and a weight-based scheduling algorithm, named Most Packet Wavelength-Fiber Pair First (MPWFPF), that delivers 100% throughput for input-buffered optical switches at speedup 1; (3) a basic three-stage scheduling procedure for the OpCut switch, furthermore, a pipeline mechanism for single-wavelength OpCut switches to relax time constraint and improve system throughput, and NP-hardness and inapproximability proof for the optimal scheduling problem in WDM OpCut switches, as well as bounded approximation algorithms; (4) an energy-aware routing mechanism for hybrid optical NoCs.

Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank my Ph.D. advisor, Dr. Yuanyuan Yang, for opening up the world of interconnection networks for me with patience, guidance, and foresight. Without her advice and support through the years of my Ph.D. study, I would not have been able to carry out this work.

Many thanks to the past and present members of the High Performance Computing and Networking Research Lab: Zhenghao Zhang, Ming Ma, Chi Ma, Deng Pan, Min Yang, Miao Zhao, Xi Deng, Ji Li, Dawei Gong, Zhiyang Guo, and Zhemin Zhang, for their generous help over the years. It was a privilege to work with them. Special thanks to Dr. Zhenghao Zhang, who gave me tremendous help in my research.

My sincere gratitude goes to my wife Shuting Peng. Her constant support, encouragement and patience throughout the duration of my graduate study have been an inspiration to keep me moving forward. Through life's ups and downs, she is my strongest supporter, closest companion, and greatest motivation. I am forever grateful to my parents Xingqiu Liu and Tilin Wang, and parents-in-law, Shijun Peng and Min Zhang, for their unconditional love. Also I would like to thank my grandparents, uncles, aunts, and cousins, for their unfailing support at every step of the way.

Sincere thanks to Dr. Wei Zhu, Dr. Yeming Ma, Dr. Sean Li, and Grace Tan, who are great mentors and helpful friends of my wife and I, for all the favors they have bestowed on me and my family.

Finally, I would like to thank to my committee members, Dr. Sangjin Hong, Dr. Alex Doboli, and Dr. Esther M. Arkin, who have also served on the committee of my qualification exam, for their guidance and support.

Chapter 1

Introduction

This chapter starts with a brief overview of basic switch architectures and packet scheduling algorithms, followed by an explanation of the motivation and the contributions of this dissertation, as well as the dissertation outline.

1.1 Basic Switch Architectures and Packet Scheduling Algorithms

In this thesis, a switch, or an interconnect, refers to a switching network that interconnects multiple components at any level - intra-chip, chip-to-chip, board level, and computer networks, and transmits data from its input ports to its output ports. Based on where the packet buffering takes place, switches can be classified into four basic architectures, input-queued (IQ), output-queued (OQ), combined-input-and-output-queued (CIOQ), and combined input-crosspoint queueing (CICQ, also known as buffered crossbar), each of which has its own advantages and disadvantages. OQ switches always guarantee maximum throughput. However, it is achieved at the expense of an up-to-N speedup, i.e. the memory has to run N times faster than the line card rate, where N

is the size of the interconnect. Switches with input buffer do not require such a high speedup, but they used to suffer from the Head-of-Line (HOL) blocking problem. The HOL blocking occurs when there is an output channel available but packets destined to it are blocked by other packets ahead of them in the same input queue, which are waiting to be routed to a different output channel. It was shown in [1] that, if HOL exists, input-queued interconnects can achieve at most 58.6% throughput under uniform i.i.d Bernoulli traffic. That is, only 58.6% of all the packets are successfully delivered to the switch output. HOL blocking can be removed by the virtual output queueing (VOQ) technique, under which multiple queues are maintained at each input and packets destined to different outputs are stored in different queues. IQ and CIOQ switches have better scalability than OQ switches and are widely used in today's networks.

Many scheduling algorithms have been proposed for electronic switches by formalizing the scheduling problem into a matching problem. Existing scheduling algorithms can be roughly divided into two categories: *maximum weighted matching* based optimal algorithms and *maximal sized matching* based fast algorithms. The first category includes algorithms such as Longest Queue First (LQF) and Oldest Cell First (OCF) [2]. These algorithms have impractically high computing complexity, but are of theoretical importance as they deliver 100% throughput under virtually any admissible traffic. The second category includes, for example, Round Robin Greedy Scheduling (RRGS) [3], Parallel Iterative Matching (PIM) [4] and iSLIP [5]. These algorithms only look for a maximal matching in each time slot hence have practical time complexity. They are therefore preferred in real systems, although they can only give sub-optimal schedules.

1.2 Motivation for This Dissertation

In recent years, switches and interconnects draw increasingly more attention due to the fact that they tend to become a bottleneck at all levels. There are many requirements posed on an interconnect network, such as low latency, high throughput, low error rate, low power consumption, as well as scalability. Finding a solution that can satisfy all these needs is a non-trivial task.

Due to the huge bandwidth and low error rate, optics is playing an increasingly important role in switching networks. Many optical switch and interconnect architectures, have been proposed in recent years, see, for example, [6, 7] [8] [13] [14] [15]. On one hand, the rapid growth of the optical technologies, such as wavelength division multiplexing (WDM) and wavelength conversion, provides a platform to exploit the huge capacity of optical fibers for switching networks. On the other hand, there remain challenges to the deployment of optical switches, of which the lack of optical random access memory (RAM) is a major one.

In a WDM switch, the multiplexing of multiple optical signals on a single fiber is achieved by carrying each signal on a separate wavelength. Contention arises in a switch when more than one packets request the same output port. Contention arises in a switch when more than one packets request the same output port. While buffering is a primary method to resolve contention in electronic networks, currently optical buffers are much less powerful and commonly implemented by fiber delay lines (FDL), which are essentially segments of optical fibers. The buffering effect is achieved by sending the optical data to be buffered to the FDL. Assume the length of the FDL is L and the optical signal transmits in the FDL at a speed v, then the FDL can provide a fixed delay of L/v to any incoming signal. The advantage of FDL is that it requires nothing but a regular fiber, while the drawback is that it can only provide a fixed buffering time. To provide flexible delays, FDL has to be combined with switches. Alternatively, there are emerging techniques to provide optical buffering by slowing down the light [16, 17, 17, 19, 20]. While these works present interesting results towards implementing all-optical buffers, it is still unclear whether this method can provide sufficiently large bandwidth and buffering capacity for practical systems. As pointed out in [21], slow-light optical buffers are constrained by some fundamental physical limitations. Therefore, at least for the near future, efficient implementation of optical buffers will remain a challenging issue.

Despite the lack of optical RAM, there is yet a unique approach to resolving contention in WDM optical networks, which is the conversion in the wavelength domain. If wavelength conversion is available, when two packets compete for one output wavelength channel, one of them can be converted to another wavelength on which the output channel is idle. Buffering is not necessary as long as such an alternative wavelength can be found. To maximize throughput while keeping queuing delay under control, a well-designed optical switch needs to function in both the time domain and the wavelength domain. Given the fact that optical buffering is limited at current stage, wavelength conversion is of particular importance to contention resolution in optical switches.

As key topics in the development of optical switching networks, scheduling and performance evaluation have been attracting considerable research interest. While there has been extensive research in these fields for electronic networks, most of them cannot be directly leveraged for optical networks - on one hand, some components, such as RAM, are still missing in optical domain; on the other hand, the solutions for electronic networks do not take into consideration unique characteristics of optics. This dissertation tries to address these issues and tackles several important questions in packet scheduling and performance evaluation for optical switches.

1.3 Contributions

The main contributions of this dissertation include:

- **A optimal packet scheduling algorithm for output-buffered optical switches [22].** The proposed Augment to Full algorithm, along with the schedule construction algorithm, simultaneously maximizes system throughput and minimizes average queuing delay, making use of limited-range wavelength conversions. It is shown that the complexity of the proposed scheme asymptotically matches the lower bound of the scheduling problem.

- **A mechanism to achieving 100% throughput for input-buffered optical switches [23].** To show that an input-bufferd WDM switch is capable of delivering at its full capacity, we first propose a novel fiber-delay-line based input buffering fabric with multiple "exits" that is able to provide flexible buffering delay. Then we design a weight-based packet scheduling algorithm, named Most Packet Wavelength-Fiber Pair First (MPWFPF), and present a theoretical proof that the combination of the new buffer and scheduling algorithm deliver 100% throughput for WDM switches with input buffer under virtually any practical traffic, without any requirement on speedup. To the best of our knowledge, this is the first work that theoretically proves WDM packet switches are able to deliver 100% throughput with no assumption on traffic or wavelength conversion patterns. Furthermore, we propose a more practical, parallel iterative matching based scheduling algorithm, WDM-iSLIP, that can efficiently determine an approximate optimal scheduling with much lower time complexity.

- **A basic scheduler and a pipeline algorithm for packet scheduling in single-wavelength OpCut switch [24] [25].** OpCut switch is a recently proposed, optical-electronic hybrid switching architecture that features low latency and minimum optical/electronic/optical (O/E/O) conversions. We decompose the packet scheduling procedure in an Opcut switch into three stages and propose possible implementation for each stage. To further relax time constraint and improve throughput, we design a novel mechanism to pipeline the scheduling process, which is further made adaptive to reduce pipeline overhead.

- **NP-hardness, inapproximability proof of the optimal scheduling problem in a WDM OpCut switch and a series of bounded approximation algorithms [26].** We theoretically prove that an optimal scheduling problem in a WDM OpCut switch is not only NP-hard but also inapproximable in polynomial time within any constant factor, by reducing it to the set packing problem. We then propose an approximation algorithm, called "Longest-or-Heads", that approximates the original scheduling problem by a factor in line with the best known approximation algorithm for set packing. The implementation of Longest-or-Head and its variations is also discussed in details.

- **An energy-aware packet routing mechanism for hybrid optical networks-on-chip (NoCs) [27].** Using a detailed model of optical routers we reduce the energy-aware routing problem into a shortest-path problem, which can then be solved using one of the many well known techniques. By applying our approach to different popular topologies, we show that the energy consumed in data communication in an optical NoC can be significantly reduced.

The research combines algorithm design, hardware design, analytical, and simulation techniques to conduct comprehensive studies on above issues. We expect the research result to have a significant impact on switch architecture design, scheduler implementation and performance evaluation for the development of future high and ultra high speed optical switching networks. The outcome of this project is generally applicable to a wide range of interconnect networks, including computer networks, inter-chip switching, intra-chip switching, and networks-on-chip.

1.4 Dissertation Outline

The rest of the dissertation is organized as follows. Chapter 2 proposes Augment to Full, an optimal packet scheduling algorithm in optical switches with output buffer and limited-range wavelength conversion capability. Chapter 3 studies the problem of achieving maximum throughput for input-buffered optical switches. Chapter 4 and 5 study packet scheduling problems in OpCut switches and focus on the single-wavelength and WDM scenario, respectively. Chapter 6 presents an energy-aware routing mechanism for hybrid optical NoCs. Finally Chapter 7 concludes the dissertation.

Chapter 2

Optimal Packet Scheduling in Output-Buffered WDM Optical Switches

All-optical packet switching is a promising candidate for future high-speed switching. However, due to the absence of optical random access memory, the traditional Virtual Output Queue (VOQ) based input-queued switches are difficult to implement in the optical domain. In this chapter we consider output-buffered optical packet switches. We focus on packet scheduling in an output-buffered optical packet switch with limited-range wavelength conversion, aiming at maximizing throughput and minimizing average queuing delay simultaneously. We show that this problem can be converted to a minimum cost maximum network flow problem. To cope with the high complexity of general network flow algorithms, we present an algorithm that can efficiently find an optimal schedule in $O(\min\{NW, BW\})$ time, where N is the switch size, W is the number of wavelength channels per fiber and B is the length of the longest FDL at the output of the switch. The complexity of the new algorithm asymptotically matches the lower bound of the scheduling problem. We also conduct extensive simulations to test the performance of the proposed scheduling algorithm under different traffic models.

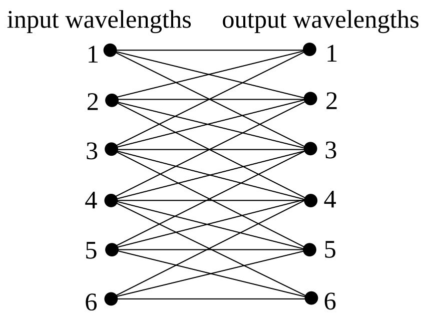


Figure 2.1: The wavelength conversion model for a switch with $W = 6$ and $d = 2$.

The rest of the chapter is organized as follows. Section 2.1 introduces the switch model we consider. Section 2.2 formalizes optimal packet scheduling in the switch into a network flow problem. Section 2.3 presents the new algorithm for finding an optimal schedule. Section 2.4 gives the correctness proof of the new algorithm. Section 2.5 discusses the implementation details and the complexity of the new algorithm. Section 2.6 gives the simulation results. Finally, Section 2.7 concludes the chapter.

2.1 WDM Optical Switch Model

In this section we introduce the WDM optical packet switch model considered in this chapter, including the wavelength conversion model and the switch architecture.

2.1.1 Wavelength Conversion Model

As mentioned earlier, wavelength conversion is a unique approach to resolving contention in WDM switches. Packet scheduling in WDM switches needs to take into consideration the wavelength conversion capability of the switch, since each packet has more than one possible output wavelength channel to choose from. In electronic networks, little scheduling needs to be done in an output-queued switch unless extra requirements such as Quality of Service (QoS) must be met, which is also the case in a WDM switch with no wavelength conversion. If full-range wavelength conversion is available in a WDM switch, in which any wavelength can be converted to any other wavelength in the optical system, the scheduling is also trivial since no matter which wavelength a packet comes from, all the output wavelength channels are available for it. Thus, packets coming from different wavelengths are identical in terms of scheduling, unless some extra requirements such as QoS are imposed.

However, WDM switches without wavelength conversion ability cannot exploit the wavelength domain when there is contention for channels, while it is expensive to implement full-range wavelength conversion under current technology [28]. Therefore, we adopt limited-range wavelength conversion in the switch we consider. With limited-range wavelength conversion, a wavelength can only be converted to a fixed set of its adjacent wavelengths. Previous studies have shown that with careful designs, limited-range wavelength conversion can achieve performance comparable to that with full-range conversion [29] [30]. In this chapter, we consider a system with a total of W different wavelengths, indexed from 1 to W. It is assumed that the convertibility between two wavelengths is symmetric. That is, if wavelength i can be converted to wavelength j, then wavelength j can also be converted to wavelength i. The conversion range of the i_{th} wavelength is given by $[\max\{1, i-d\}, \min\{W, i+d\}]$, where d is a small integral constant and is called the conversion degree. An example of this wavelength conversion model for $W = 6$ and $d = 2$ is shown in Fig. 2.1. Note that some wavelengths at the top and the bottom of the figure have a conversion range less than $2d + 1$.

2.1.2 WDM Switch Architecture

The WDM optical packet switch architecture considered in this chapter is shown in Fig. 2.2. In this chapter, we follow the same assumptions as many other optical switch designs that the switch

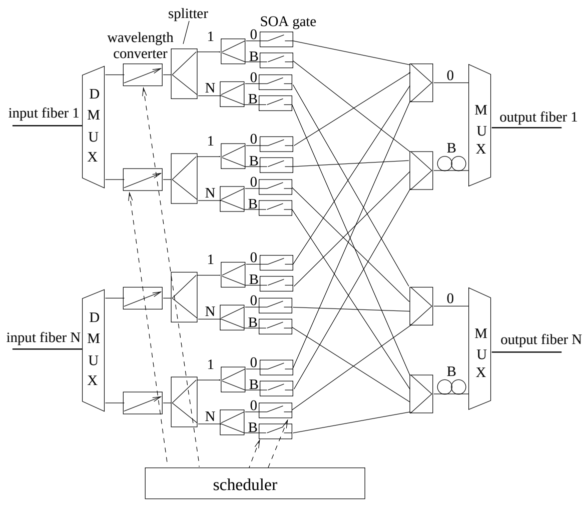


Figure 2.2: The output-buffered WDM optical switch. The output buffer at each output port is implemented by $B + 1$ fiber delay lines.

works in a time-slotted manner, and packets of a fixed length arrive at the switch at the beginning of time slots. The length of a time slot is equal to the duration of a packet. In the rest of this chapter, we use N to denote the number of fibers at the input/output of the switch, W to denote the number of wavelength channels that each fiber contains, and d to denote the conversion degree. Since the scheduling of packets destined to different output fibers is mutually independent, we only consider the scheduling of packets destined to a particular output fiber. As we deal with a single output fiber, “output wavelength channel” and “output wavelength” have the same meaning in the rest of this chapter.

In this switch architecture, the demultiplexer at each input fiber demultiplexes the incoming optical signal to W signals, one on each wavelength. A limited-range wavelength converter is assigned to each incoming wavelength channel. After possible wavelength conversion, each signal is split into N copies and sent to the SOA gate array. An SOA gate can be turned on or off depending on which output and fiber delay line the packet needs to be scheduled to. For unicast traffic, in each time slot at most one SOA gate is turned on out of all the SOA gates on each input

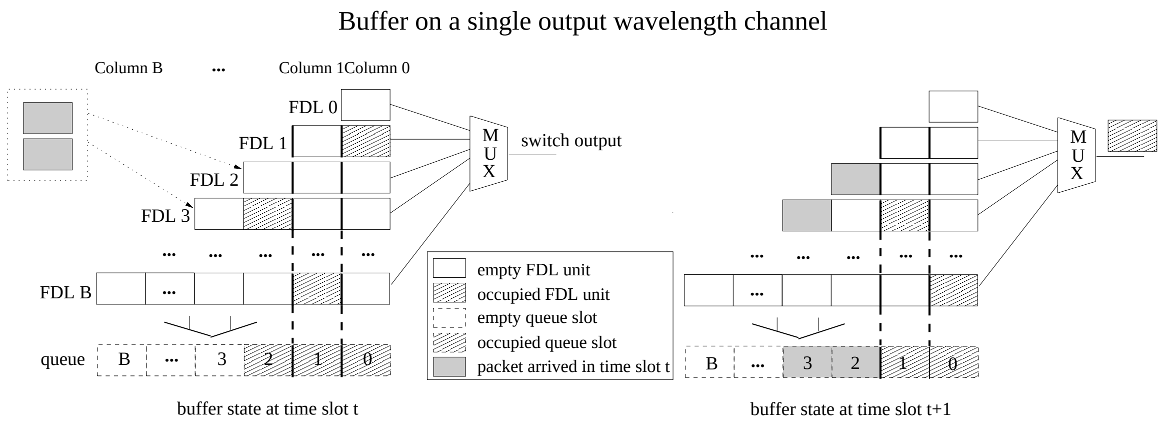


Figure 2.3: The physical and abstract buffer for an output wavelength channel. At any time all the physical buffer units in the same column can store no more than one packet in total, otherwise collision would occur at the switch output. Thus, all physical buffer units in the same column can be abstracted as a single logical buffer slot, and the buffer slots of all columns form a queue. In each time slot, the head-of-queue packet leaves the switch and more than one packet may join the queue.

wavelength channel. For multicast traffic, more gates may be turned on so that a packet can be sent to multiple outputs. All wavelength converters and SOA gates are managed by a central scheduler, thus packet routing is well coordinated.

It should be pointed out that although this switch has an OQ architecture, it does not require any speed-up. In other words, the output buffer and switching fabric do not have to run any faster than the input. This is achieved by implementing the output buffer at each output fiber with $B + 1$ fiber delay lines of lengths from 0 to B. Packets scheduled to the delay line of length i will leave the switch i time slots later. If there are multiple packets that need to be sent to the same wavelength channel of a particular output fiber, they can be transmitted to different delay lines of that fiber simultaneously, through the internal switching fabric that has B ports for each output wavelength channel. It can be considered as trading-off the space complexity to release the time constraint.

The delay lines are able to accept up to $B + 1$ packets per wavelength in each time slot. At first glance, it seems that up to $B(B+1)/2$ packets on each wavelength can be stored by the delay lines.

However, as pointed out in [31], this is not true, since on each wavelength there should be no more than one packet coming out of all the FDLs in each time slot. Otherwise collision would occur at the output. Consider the output buffer on a single wavelength channel as shown in Fig. 2.3. Each FDL can be divided into unit-length segments, each of which can delay one time slot and may hold one packet. Among all the FDLs, $B + 1 - i$ of them have a segment that is i time slots away from the switch output ($i = 0, 1, \dots, B$), which form column i in Fig. 2.3. To avoid collision, at most one segment in column i can be occupied by a packet at any instant. Thus for scheduling purpose, all segments in a column can be abstracted as a single *logical buffer slot*. Consequently, the entire FDLs on a single output wavelength channel can be abstracted as $B + 1$ logical buffer slots. As will be shown in Section 2.3.2.3.1, these $B + 1$ logical buffer slots for a single output wavelength can be further modeled as a first-in-first-out (FIFO) queue with $B + 1$ slots indexed from 0 to B. We refer to the queue on output wavelength channel i as Q^i, and the j_{th} slot (or slot j) in Q^i as $\langle i, j \rangle$. A packet at the switch input can be scheduled to Q^i if it is on wavelength i, or a wavelength convertible to wavelength i. Since the j_{th} slot in Q_i is j time slots away from the switch output, when we schedule a packet to $\langle i, j \rangle$, we actually put it into the j_{th} FDL of output wavelength channel i so that the packet will reach the switch output after j time slots.

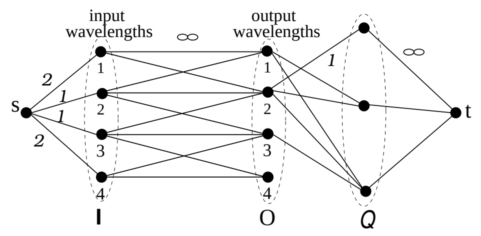
2.2 Network Flow Approach for Finding an Optimal Schedule

In the rest of the chapter, we will consider packet scheduling in the optical switch described in the previous section. In such an optical switch, in each time slot there are packets arriving on each input wavelength channel and each of them is destined to some output fiber. Packet scheduling is to decide how to put these packets in the output buffer of the switch. We are interested in finding an optimal schedule with the following properties: (1) It schedules the maximum number of packets, or equivalently, drops the minimum number of packets among all schedules; (2) It introduces the minimum total buffering delay among all schedules satisfying property (1). In other words, the highest priority during the scheduling is to minimize packet loss, then to shorten the delay as much as possible.

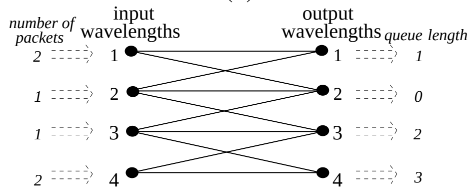
The first observation we can draw for packet scheduling in such a switch is that, there is no need to distinguish one packet from another if they are from the same input wavelength channel and destined to the same output. Therefore instead of scheduling those packets one by one, we can schedule as many of them as possible in a bulk at a time, which leads to our idea of adopting the network flow approach. Next we show how to formulate the optimal scheduling problem in the WDM switch into a minimum cost maximum flow problem.

We first introduce three sets of nodes, I, O and Q. $I = \{i_k \mid 1 \leq k \leq W\}$, with node i_k representing input wavelength k. $O = \{o_k \mid 1 \leq k \leq W\}$, with node o_k representing output wavelength k. $Q = \{q_j \mid 0 \leq j \leq B\}$, with node q_j representing the j_{th} slot of a queue. Node o_k and q_j are connected by an edge with unit capacity and cost j, if the j_{th} slot of the queue on output wavelength channel k, Q^k, is not occupied, i.e., if the length of Q^k is smaller than j. There are also two dummy nodes s and t. There is an edge between s and node i_k with capacity equal to the number of packets arriving at the input in the current time slot on wavelength k over all input fibers of the switch . Node i_u is connected to node o_v if wavelengths u and v are mutually convertible. Finally, all nodes in Q are directly connected to t. All other edges without specified capacity and cost are assumed to have an infinity capacity and zero cost. We refer to the subgraph of the flow graph containing nodes in sets I and O, and edges between the nodes in these two sets as a *request graph*. A request graph also includes information on the number of packets on each input wavelength channel and the queue length on each output wavelength channel. An example of the flow graph and its corresponding request graph is shown in Fig. 2.4.

With the flow graph, we can convert the optimal scheduling problem into a network flow problem. Finding the maximum number of packets that can be scheduled to the output in the current



(a)



(b)

Figure 2.4: (a) An example of the flow graph for $d = 1, W = 4$ and $B = 2$, where the number of packets arriving in the current time slot on each wavelength is $2, 1, 1$ and 2, respectively. The number above each edge between s and I represents the capacity of the edge. The capacity of edges between I and O and between Q and t is infinite, while the capacity of edges between O and Q is 1. In the graph, all queue slots are available on wavelength 2, while no queue slot is available on wavelength 4. (b) The corresponding request graph.

time slot is equivalent to finding a maximum flow from s to t. To see this, note s is only connected to the nodes in I, and the capacity of the edge between s and i_k equals the number of packets arrived on input wavelength k. Besides, t is only connected to the nodes in Q, and the maximum flow the network can sustain is limited by the total capacity of the edges between nodes in O and nodes in Q, which equals the total number of empty slots in all queues at this moment. In the meanwhile, we also need to minimize the total queuing delay, which implies that queue slots with a smaller index, or, edges between nodes in O and nodes in Q with a smaller cost, are preferred. Combining these two aspects, it is clear that an optimal schedule corresponds to a maximum flow with minimum cost in the flow graph. However, directly applying existing network flow algorithms may lead to the time complexity that may be too high for an optical switch that requires very fast switching. In fact, the best known algorithm for finding a minimum cost maximum flow in an arbitrary graph requires $O(nC(m + n\log n))$ time [32], where n is the total number of nodes in the flow graph, m is the number of edges, and C is the largest cost of all edges. In our case, $n = O(W+B)$, $m = O(WB)$ and $C = \max\{N, B\}$, thus the time complexity of the network flow algorithm is $O(WB(W + B) \cdot \max\{N, B\})$. Next, based on the request graph and the flow graph, we will design a new scheduling algorithm with lower time complexity by making use of some nice properties of the wavelength conversion model and the output buffer model of the switch.

2.3 The New Scheduling Scheme

In this section, we first give some preliminaries for the proposed new scheduling scheme and then present the new scheduling algorithm which includes the algorithm to construct a schedule and the algorithm to find an optimal schedule.

2.3.1 Preliminaries

As discussed in Section 2.1.2.1.2, the entire FDLs on a single output wavelength can be abstracted as $B + 1$ logic buffer slots. In the rest of the chapter, we will simply call them *buffer slots*. In the new scheduling algorithm, when we schedule a packet to an output wavelength channel, we always schedule the packet to the empty buffer slot closest to the switch output (i.e., with the shortest buffering delay) on that wavelength channel. It is a reasonable strategy since it leads to a shorter buffering delay at no extra cost, which is always desirable. Next we will show that by following this strategy, the buffer slots on an output wavelength form a FIFO queue.

Under this strategy, the output buffer has two useful properties. Consider any buffer slots i and j ($0 \leq i < j \leq B$) on an output wavelength. Recall that buffer slot i is i time slots away from the switch output. The first property comes directly from the strategy adopted: if buffer slot i is empty in a time slot, then buffer slot j cannot be used in this time slot. By "used" we mean a packet is sent to the buffer slot and occupies it according to the schedule. Conversely, if buffer slot j is used in a time slot, then buffer slot i cannot be empty at the end of this time slot. The second property, straightforwardly derivable from the first property, is that all empty buffer slots on an output wavelength channel are consecutive at the beginning of any time slot. In other words, when a time slot starts, if buffer slot j is occupied, then buffer slot i must be occupied. Conversely, if buffer slot i is empty, then buffer slot j must be empty.

Combining these two properties, we can draw the following observation: the FDL buffer on each output wavelength can be modeled as a queue. As a result, the buffering status on an output wavelength channel can be represented by a single variable: the length of the queue. In each time slot the head-of-queue packet leaves the switch. If there are packets assigned to this output wavelength channel by the scheduler, these packets join the queue. It will be seen later that making use of this fact greatly simplifies our optimal scheduling algorithm. Besides, it is also worth pointing out that our algorithm works not only with the switch architecture introduced in Section 2.1, but also with all output-buffered WDM optical switches whose buffer can be modeled as queues.

Before we go into the details of our algorithm, let us first look at a seemingly straightforward approach. Intuitively, one may wonder whether an optimal schedule can be achieved by scheduling packets one by one to the output wavelength channel with the shortest queue among all candidate output wavelength channels. Let us call this approach "Join Shortest Queue," or JSQ for short. Note that while JSQ's idea is simple, its complexity might not be as low as it appears - an accessible output wavelength channel with the shortest queue must be determined before each packet can be scheduled. Nevertheless, Fig. 2.5 shows a simple example in which JSQ leads to only a sub optimal schedule. Assume there are three wavelengths. The conversion model is shown in Fig. 2.5(a), along with the number of arrived packets on each input wavelength and the queue length on each output wavelength. Following JSQ, the packet on input wavelength 1 and 2 will be scheduled to output wavelength 1 and 2, respectively. Then two of the three packets on input wavelength 3 will

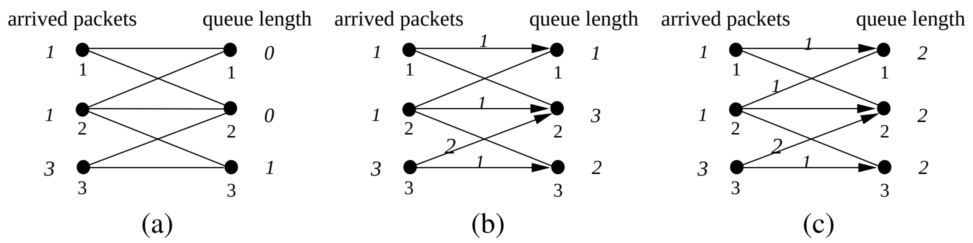


Figure 2.5: An example of Join Shortest Queue. The number on an edge indicates the number of assigned packets. (a) The request graph. (b) JSQ failing to find an optimal schedule. (c) An optimal schedule.

be sent to output wavelength 2, and the rest sent to output wavelength 3. This schedule is shown in Fig. 2.5(b) and results in a total delay of $(1 + 1 + 2 + 3 + 2) = 9$. However, as can be seen in Fig. 2.5(c), an optimal schedule introduces only a total delay of $(1 + 2 + 1 + 2 + 2) = 8$. The reason why JSQ may not be able to find an optimal schedule lies in the fact that given the conversion capability among wavelengths, the scheduling of packets on different wavelengths is dependent. Then applying JSQ to packets on one wavelength can lead to packet loss or long delay of packets on another wavelength, which may be avoided if the two wavelengths are coordinated better. What our algorithm does, in some sense, is to ensure the best coordination among all wavelengths, so that an overall optimal solution is achieved. Next we will describe our new algorithm.

2.3.2 Schedule Construction Algorithm

As mentioned earlier, packets at the switch input on the same wavelength and destined for the same output wavelength channel can be considered to be identical. Thus, instead of determining how each single packet should be scheduled, in a time slot the scheduler can simply calculate how many packets on each wavelength switch-wide should be transmitted from the input side of the switch, and how these packets should be distributed to the output buffers of the switch.

As shown in Fig. 2.6, let x_i denote the number of packets on input wavelength i (over all input fibers switch wide) to be transmitted to a particular output fiber in the current time slot according

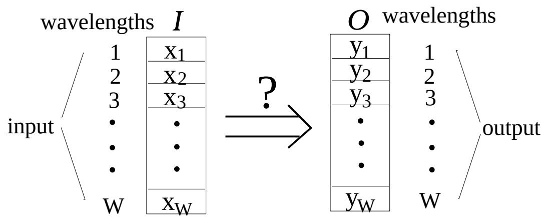


Figure 2.6: Vectors $\mathcal{I}$ and $\mathcal{O}$ of a schedule.

to some schedule, and y_i denote the number of buffer slots on wavelength channel i of that output fiber to be occupied by these packets, for $1 \leq i \leq W$. Clearly $\sum_{i=1}^{W} x_i = \sum_{j=1}^{W} y_j$ must hold. Let vector $\mathcal{I} = \langle x_0, x_1, \ldots, x_W \rangle$, and $\mathcal{O} = \langle y_0, y_1, \ldots, y_W \rangle$. Note that $\mathcal{I}$ and $\mathcal{O}$ differ from the node sets I and O in the flow graph and the request graph in Section 2.2. Generally speaking, knowing vectors $\mathcal{I}$ and $\mathcal{O}$ of a schedule is not sufficient to determine the corresponding schedule, unless the mapping from $\mathcal{I}$ to $\mathcal{O}$ is also known. Fortunately, in our case, once $\mathcal{I}$ and $\mathcal{O}$ are obtained, a schedule with the minimum queuing delay among all schedules that satisfy $\mathcal{I}$ and $\mathcal{O}$ can be constructed as follows.

Since our goals include minimizing the buffering delay, as previously discussed in Subsection 2.3.1, the output buffer works as queues. Then the task is to distribute the x_i packets on each input wavelength to the queues on the output wavelength channels, such that the queue on output wavelength channel i receives y_i packets, for $1 \leq i \leq W$. Starting from wavelength 1, we schedule as many as possible packets from input wavelength 1 to output wavelength channel 1. That is, we schedule $\min\{x_1, y_1\}$ packets from input wavelength 1 to output wavelength channel 1. This is equivalent to augmenting $\min\{x_1, y_1\}$ along edge (i_1, o_1) in the request graph. If $x_1 \geq y_1$, we schedule as many of the remaining $x_1 - y_1$ packets on input wavelength 1 as possible to output wavelength channel 2. On the other hand, if $x_1 < y_1$, then $y_1 - x_1$ queue slots on output wavelength channel 1 are still not used, and yet to be occupied by packets from other wavelengths. Thus we continue to schedule $\min\{x_2, y_1 - x_1\}$ packets from input wavelength 2 to the remaining queue slots on output wavelength channel 1. This process is carried on until all the $\sum_{i=1}^{W} y_i$ queue slots are occupied. After that, the number of packets that should be sent from each input wavelength to each output wavelength channel is known. Or, equivalently, a schedule is constructed from $\mathcal{I}$ and $\mathcal{O}$. The construction algorithm is formally described in Table 1.

The correctness proof of the schedule construction algorithm is postponed to Section 2.4. Note that this algorithm does not guarantee to reconstruct a specific schedule. Instead, it efficiently finds the schedule with the minimum total buffering delay among all schedules satisfying the given $\mathcal{I}$ and $\mathcal{O}$. In particular, such a schedule satisfying the $\mathcal{I}$ and $\mathcal{O}$ of an optimal schedule must be an optimal schedule as well. Based on the schedule construction algorithm, next we give an algorithm, called *Augment to Full Algorithm*, to efficiently determine an optimal schedule.

2.3.3 Augment to Full Algorithm

The Augment to Full Algorithm works on the request graph. It decomposes the task of finding an optimal schedule into multiple iterations. At the end of each iteration, the algorithm obtains an intermediate schedule that evolves toward an optimal schedule. During each iteration, the intermediate schedule is determined in two steps: first finding a valid combination of $\mathcal{I}$ and $\mathcal{O}$, then applying the schedule construction algorithm to the $\mathcal{I}$ and $\mathcal{O}$ found.

The basic idea for the Augment to Full Algorithm to find a valid combination of $\mathcal{I}$ and $\mathcal{O}$ is similar to that of the schedule construction algorithm. Starting from output wavelength 1, we schedule as many as possible packets from input wavelength 1 to the empty queue slots on output wavelength 1. By "empty" we mean the queue slot is not occupied by packets arriving in earlier time slots. If all packets from input wavelength 1 can been scheduled, we say input wavelength 1 is full, or "filled" by output wavelength 1, and then we continue to schedule as many packets as possible from input wavelength 2 to output wavelength 1. If the queue on wavelength 1 has no more empty slots, we will turn to output wavelengths 2, 3, ..., until input wavelength 2 is finally "filled" by some output wavelength, or the largest wavelength that wavelength 2 can be converted to is reached. Then input wavelength 3 is to be filled. The process continues until there are no more available packets or queue slots. During this process we fill the input wavelengths one by one using the output wavelengths, thus the process is called "*filling process*." We also record the number of packets scheduled on input wavelength i, x_i, and the number of queue slots on output wavelength j to be occupied by these packets, y_j.

To ensure the final schedule is optimal, queue slots that introduce shorter delay should have a higher priority to be used. It is exactly why the Augment to Full Algorithm is split into $B+1$ iterations. In iteration i, only queue slots that are empty and with an index smaller or equal to i are eligible for filling the input wavelengths. That is, we assign packets to the queues under the constraint that the length of any queue can be augmented to at most i. If not all packets can be assigned in iteration i, we continue to iteration $i+1$, taking into account slot $i+1$ of the queues, if there are any empty slots, and so on. If a packet is assigned to a queue slot according to the result at the end of an iteration, we say the slot is selected in this iteration. Note that a queue slot being selected in an iteration does not mean that a packet has been *physically* scheduled to the slot and occupied it. In fact, no packet is transmitted until the final optimal schedule is obtained. The real implication that a queue slot is selected in iteration i is that, when packets are only allowed to be sent to the first i slots of each queue, a packet should be sent to this particular slot, if the minimum total delay is desired.

In fact, a critical idea of the Augment to Full Algorithm can be summarized into the following two rules: (1) for slot i of any queue, if it is empty and selected in iteration i of the algorithm, then it should be selected in all of the following iterations, and will be used in the final schedule. Otherwise, (2) if it is not selected in iteration i, then it should not be selected in any of the following iterations. Furthermore, since this queue slot will not be used in the final schedule, none of the queue slots behind it on the same wavelength should be used in the final schedule, according to the first property of the output buffer as stated in Subsection 2.3.1. We will soon show that by following these rules, the Augment to Full Algorithm indeed finds an optimal schedule.

Note that the filling process itself incorporates neither rule. In particular, it does not guarantee any priority for the queue slots selected in previous iterations to conform with the first rule. Instead, starting from wavelength 1, the filling process merely assigns packets to queue slots in an ascending order of wavelengths. Thus, it is possible that in iteration i, during the filling process, the place of a queue slot that was selected in iteration $i-1$, say, q, is taken by another queue slot q' that was not among the selected queue slots of iteration $i-1$. That is, during the filling process of iteration i, by assigning a packet to q', we end up with no packet to be assigned to q. We refer to this situation as slot q being *blocked* by slot q'. Fig. 2.7 illustrates a simple blocking scenario. According to the first rule, whenever such a blocking occurs, we should find the corresponding q' and remove it from the current result. Clearly, q' cannot be a queue slot with an index smaller than or equal to $i-1$, since any slot not selected in iteration $i-1$ is not even taken into consideration in iteration i according to the second rule. Hence q' must be the i_{th} slot of some queue. Since the filling process always accesses the wavelengths in an ascending order, q' is on a smaller wavelength than q. To be more specific, we can always choose q' to be the queue slot closest to q and with index i to which a packet is assigned during the filling process. To see this, assume that another packet is assigned to the i_{th} slot of the queue on a wavelength smaller than q'. Denote this queue slot as q''. If by removing q' and restarting the filling process, we still cannot assign a packet to q, then we cannot assign a packet to q even if we remove both q' and q''. The reason is that the packet assigned to q'', denoted as p'', must be on a wavelength smaller than or equal to that of the packet (denoted as p') assigned to q'. This is because that the filling process always starts from wavelength 1 and works towards wavelength W. In other words, the filling process works in a top-down manner. For the same reason, the packet assigned to any of the queue slots between q' and q is on a wavelength at least as large as that of p'. If after the removal of q' and the matching between p' and q', there is no

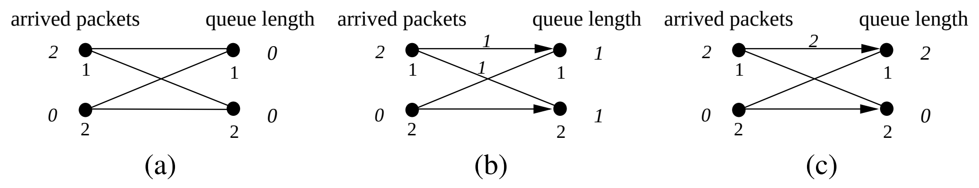


Figure 2.7: A simple blocking scenario (a) The request graph of wavelength 1 and 2 (the rest wavelengths are assumed of no interest hence not shown). (b) Iteration 0. Packets are scheduled to queue slots with index 0 on both wavelengths. (c) Iteration 1. Scheduling packets to queue slots with index 0 or 1. Queue slot with index 0 on wavelength 2 is blocked by queue slot with index 1 on wavelength 1.

augmenting path from p' to q following which q can be added to the current result, then there is no augmenting path from p'' to q after the removal of q''. Similarly, it can be shown that if q cannot be selected after q' is removed, then q cannot be selected even if all the queue slots with index i are removed. This contradicts the fact that q was selected in previous iterations. Thus it must be true that after the removal of q', q can be selected.

The remaining issue is how to locate q', which blocks the previously selected queue slot. This can be done by maintaining an auxiliary stack that is reset at the beginning of each iteration. During a filling process, whenever a packet is scheduled to the i_{th} slot of a queue, the corresponding output wavelength is pushed into the stack. Later, if a previously selected queue slot q is found blocked, the top element of the stack is popped, on which the i_{th} queue slot is the corresponding q'. We then replace q' with q in the current result. Since we are calculating $\mathcal{I}$ and $\mathcal{O}$, if q and q' are on wavelengths w_q and $w_{q'}$, respectively, we add 1 to y_{w_q} and deduct 1 from $y_{w'_q}$ to reflect the change.

Note that although $\mathcal{I}$ and $\mathcal{O}$ can be forced to conform with the first rule when such a blocking occurs as discussed above, how $\mathcal{I}$ is mapped to $\mathcal{O}$ is lost during the process. This is why we need the second step in each iteration to recover the mapping with the schedule construction algorithm. When the schedule construction is done, to enforce the second rule, we scan the output wavelengths one more time. If slot i of the queue on an output wavelength is empty but not selected in this iteration, the output wavelength is *locked.* Locking an output wavelength in iteration i means that queue slot i and behind on that wavelength are permanently removed from the pool of queue slots that are eligible to be selected. These slots will not be considered in the remaining iterations, even if they are empty. That is, the "augmenting" of the queue on this output wavelength channel stops.

At the end of an iteration of the Augment to Full Algorithm, we can draw several interesting observations. First, no *crossing edges* are generated in the schedule obtained. By crossing edges we mean there are two edges (i_1, o_1) and (i_2, o_2) in the request graph, such that $i_1 < i_2$ and $o_1 > o_2$, and there are packets scheduled both from i_1 to o_1 and from i_2 to o_2. This property is straightforward since the Augment to Full Algorithm works in a top-down fashion. Second, at the end of an iteration, if an output wavelength is locked, it implies that all input wavelengths within the conversion range of that output wavelength have been filled. Conversely, if at the end of an iteration there exists an eligible but unselected queue slot on an output wavelength, and an unassigned packet on an input wavelength, then the two wavelengths are not convertible. Third, if an output wavelength is locked in an iteration, then the distribution of its queue slots to the input wavelengths is fixed hereafter. To see this, assume an output wavelength w is locked in iteration i and the distribution changes in iteration $i + 1$. It can only occur after a packet is assigned to slot $i + 1$ of some queue. Denote the packet as p and the queue slot as q. Then, if q is removed from consideration, an augmenting path can be found from p to queue slot $\langle w, i\rangle$ (otherwise assigning p to q should not have affected w). However, that means in iteration i, $\langle w, i\rangle$ should have been selected, and w should have not been locked at all!

The above third observation implies that, after output wavelength w is locked, both w and the packets assigned to w do not need to be considered in the remaining iterations, since these packets are guaranteed to be assigned to w, and w cannot take any more packets. Therefore, we define the following sets for each iteration of the Augment to Full Algorithm:

- N_i: the set of wavelengths whose buffer length was i at the beginning of the current time slot. The Augment to Full Algorithm starts to take these wavelengths into consideration from iteration i.

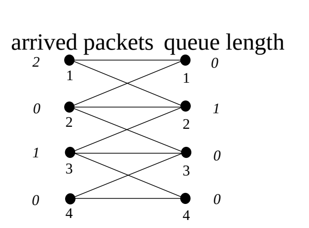
- L_i: the set of wavelengths locked during iteration i.

- A_i: the set of "active" output wavelengths in iteration i, i.e., the output wavelengths that have been taken into consideration by the Augment to Full Algorithm and are not locked yet. Clearly, $A_i = \bigcup_0^i N_i - \bigcup_0^{i-1} L_i$ for $1 \leq i \leq B$, and $A_0 = N_0$.

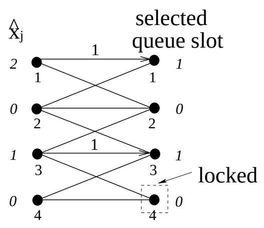
Before starting iteration i, we calculate $A_i = A_{i-1} \cup N_i - L_{i-1}$. Similarly, before ending iteration i, we check how many packets from each input wavelength are assigned to a newly locked wavelength, and subtract them from the number of packets to be scheduled. In each iteration, we only deal with output wavelengths that are in the set of active output wavelengths, and input wavelengths that are not filled by locked output wavelengths yet.

The detailed Augment To Full Algorithm is given in Table 2. Note that the algorithm may terminate early if after an iteration all output wavelengths are locked, or all input wavelengths are filled by locked output wavelengths.

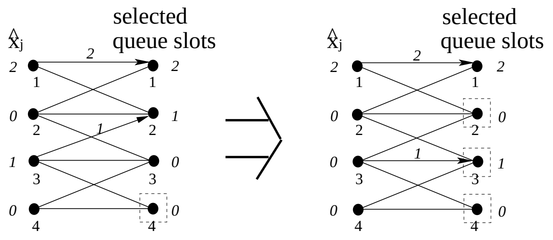
Fig. 2.8 gives an example to show how the algorithm works. The request graph of this example is given in Fig. 2.8(a). The numbers of packets to be scheduled are 2, 0, 1 and 0, respectively. The queues on output wavelengths 1, 3 and 4 are empty, while there is one packet being buffered on wavelength 2. In iteration 0, we schedule packets to queue slots with index 0. As shown in Fig. 2.8(b), the only packet on wavelength 3 is assigned to $\langle 3, 0 \rangle$ (recall that $\langle i, j \rangle$ refers to the j_{th} slot of the queue on wavelength i), and there is no packet to be assigned to wavelength 4. Thus, $\langle 4, 0 \rangle$ is not selected in iteration 0. Consequently, $\langle 4, 0 \rangle$ and any other queue slot behind it on wavelength 4 will not be used in the final schedule. Therefore, output wavelength 4 is locked. Since no packet is assigned to output wavelength 4, no update on $\{\hat{x}_i \mid 0 \leq i \leq W\}$ is needed, where $\hat{x}_i$ is defined as in Table 2. Besides, since in this iteration there exists an unscheduled packet on input wavelength



(a) The request graph with the number of packet arrivals in current time slot and the queue length on each wavelength. The number under each node indicates the wavelength it represents.



(b) Iteration 0: Scheduling packets to queue slot with index 0. $\langle 4, 0 \rangle$ is not selected, thus wavelength 4 is locked.



(c) Iteration 1: Scheduling packets to queue slots with index 1, or with index 0 and not locked. Although $\langle 3, 0 \rangle$ was selected in iteration 0, during the filling process of this iteration no packet is scheduled to it, as its place is taken by $\langle 2, 1 \rangle$. Thus $\langle 2, 1 \rangle$ is removed and $\langle 3, 0 \rangle$ re-selected. Since neither of $\langle 2, 1 \rangle$ and $\langle 3, 1 \rangle$ is selected in iteration 1, the two wavelengths are locked. Output wavelength 1 will also be locked in iteration 2. Then the algorithm terminates.

Figure 2.8: Applying the Augment to Full Algorithm to an example request graph. The number on an edge indicates the number of packets assigned.

1, the algorithm moves on to the next iteration.

In iteration 1, we reschedule all the packets to queue slots with index 0 and index 1 on output wavelengths that are no locked. During the filling process, both packets on input wavelength 1 are assigned to output wavelength 1, whose queue has two available slots $\langle 1, 0\rangle$ and $\langle 1, 1\rangle$ in this iteration. Then the packet on input wavelength 3 is assigned to $\langle 2, 1\rangle$. Output wavelength 1 and wavelength 2 are pushed into the stack due to the temporary selection of their queue slot 1. Now a blocking occurs: there is no packet to be assigned to $\langle 3, 0\rangle$, which was selected in iteration 0. By popping one element from the stack, we find that it is $\langle 2, 1\rangle$ that blocks $\langle 3, 0\rangle$. As a result, $\langle 2, 1\rangle$ is removed and $\langle 3, 0\rangle$ on wavelength 3 is selected again. As step 2 of iteration 1, the schedule construction algorithm calculates the intermediate schedule. Since neither of $\langle 2, 1\rangle$ and $\langle 3, 1\rangle$ is selected in iteration 1, the two wavelengths are locked after this iteration. As a result of output wavelength 3 being locked, $\hat{x}_j$ needs to be updated for all wavelength j convertible to wavelength 3. Therefore $\hat{x}_3$ is reduced to 0. Similarly, in iteration 2 (not shown in Fig. 2.8) output wavelength 1 is locked, and $\hat{x}_1$ is set to 0. Since now $\hat{x}_j = 0$ for all j, in other words, each packet at the input are now assigned to a queue slot, the algorithm terminates. From the buffer length of an output wavelength at the beginning of the current time slot and the iteration at which this output wavelength is locked, the number of queue slots on each wavelength to be used by the optimal schedule in this time slot can be easily calculated. For instance, the number of queue slots to be used is $1 - 0 = 1$ for wavelength 3, that is, the index of iteration at which it is locked, minus that at which the algorithm started to take this output wavelength into consideration.

2.4 Correctness Proof of the New Scheduling Scheme

In this section, we first show that the schedule construction algorithm works as expected. Then based on it we prove the correctness of the Augment to Full Algorithm.

2.4.1 Correctness of the Schedule Construction Algorithm

The correctness of the schedule construction algorithm can be stated as follows: given $\mathcal{I}$ and $\mathcal{O}$ of some schedule S, the schedule construction algorithm finds a schedule, such that (1) the $\mathcal{I}$ and $\mathcal{O}$ of the constructed schedule are exactly the same as the given ones; (2) the constructed schedule has the minimum total queuing delay among all schedules that satisfy condition (1).

Since the total number of packets assigned to each output wavelength channel is determined, condition (2) is satisfied trivially as long as the output buffer on each output wavelength works as a queue. Condition (1) can be proved by contradiction. Suppose the constructed schedule does not satisfy condition (1), which means that we end up with some unscheduled packets and queue slots that should have been scheduled, i.e., there exist some wavelengths i and j such that $x'_i < x_i$ and $y'_j < y_j$, where x'_i (y'_j) is the number of packets to be transmitted (queue slots to be used) on wavelength i (j) in the constructed schedule. We choose the first such unscheduled packet or queue slot, whichever occurs first, i.e., on a smaller wavelength. If it is an unscheduled queue slot on wavelength i, since the construction algorithm works in a top-down fashion as the filling process, any packet on a wavelength between wavelengths 1 and $\min\{i + d, W\}$ (the largest wavelength that wavelength i can be converted to), is assigned to the queue on output wavelength i or smaller in the constructed schedule. Even though, there is still a queue slot on wavelength i that cannot be used in the constructed schedule. That is to say, among all the packets scheduled by S, the maximum number of packets that can be assigned to output wavelengths $[1, i]$ is smaller than the number of queue slots used by S on these wavelengths, which is impossible. Similarly, if an unscheduled packet occurs first and it is a packet on wavelength i, then in the constructed schedule all the queue slots on wavelengths in the range of $[1, \min\{i + d, W\}]$ are occupied by packets on wavelengths smaller or equal to i. Yet there is still a packet on wavelength i that cannot be assigned. Consequently, the maximum possible number of queue slots that S uses for packets on wavelengths $[1, i]$ is smaller than the number of packets that are assigned to these wavelengths by S, which is also impossible. Thus the correctness of the schedule construction algorithm is proved.

2.4.2 Correctness of the Augment to Full Algorithm

Next we prove the correctness of the Augment to Full Algorithm. Let S_{af} denote the schedule found by the Augment to Full Algorithm. Let c_i^S denote the number of queue slots with index at most i that are used in schedule S, $0 \leq i \leq B$. Then we have the following lemma.

Lemma 1. *For any schedule S, $c_i^{S_{af}} \geq c_i^S$ for $0 \leq i \leq B$.*

Proof. It can be proved by induction. First, $c_0^{S_{af}} \geq c_0^{S_r}$ is clearly true, since in this case only slots with index 0 are considered in the Augment to Full Algorithm, and as many of them as possible will be used to fill the input wavelengths.

Now suppose it is true for $0 \leq i \leq k-1$. We prove by contradiction that it must be true for $i = k$ as follows. Assume there exists a schedule S_r such that $c_k^{S_{af}} < c_k^{S_r}$. This implies 1) S_{af} uses less queue slots with index k than S_r; 2) S_{af} schedules less packets to queue slots with index smaller than or equal to k than S_r. Therefore, after iteration k of the Augment to Full Algorithm, we must be able to find a list of packets $(p_0, p_1, \ldots, p_n)$ and queue slots $(q_0, q_1, \ldots, q_n)$

$$q_0 \longrightarrow p_0 \longrightarrow q_1 \longrightarrow p_1 \longrightarrow q_2 \longrightarrow p_2 \longrightarrow \cdots \longrightarrow q_n \longrightarrow p_n$$

such that q_0 is of index k, and p_i is assigned to q_i in S_r, but is assigned to q_{i+1} at the k_{th} iteration of the Augment to Full Algorithm. In other words, at the end of the k_{th} iteration of the Augment to Full Algorithm, q_0 is not selected, and p_n is not assigned. Assume p_i is on wavelength w_{p_i} and q_i on wavelength w_{q_i}. We claim that $w_{p_n} \geq w_{p_i}$ and $w_{q_0} \geq w_{q_i}$ for $0 \leq i \leq n$.

The claim can be justified as follows. First $w_{p_n} \geq w_{p_{n-1}}$ must be true. Otherwise, p_n, instead of p_{n-1}, should have been assigned to q_n since the filling process works in a top-down manner. Assume that w_{p_n} is not the maximum among all w_{p_i}, and w_{p_m} is the one such that $m = \max\{i \mid w_{p_i} > w_{p_n}, 0 \leq i \leq n\}$. That is, $w_{p_{m+1}} \leq w_{p_n} < w_{p_m}$ holds. Clearly, $m + 1 \leq n - 1$ since $w_{p_n} \geq w_{p_{n-1}}$. In addition, p_m and p_{m+1} are assigned to q_{m+1} in S_{af} and S_r, respectively, which indicates that $w_{q_{m+1}}$ can be converted to both w_{p_m} and $w_{p_{m+1}}$. Thus $w_{q_{m+1}}$ can be converted to w_{p_n}. However, S_{af} should have assigned p_n, instead of p_m, to q_{m+1}, since $w_{p_n} < w_{p_m}$ and the algorithm accesses the input wavelengths in an ascending order in each iteration. This is a contradiction. Therefore, $w_{p_n} \geq w_{p_i}$ for $0 \leq i \leq n$. Similarly, it can be shown that $w_{q_0} \geq w_{q_i}$ for $0 \leq i \leq n$. The claim is proved.

According to the above claim, $w_{p_0} \leq w_{p_n}$ and $w_{q_0} \geq w_{q_n}$. Meanwhile, since p_0 and p_n are respectively scheduled to q_0 and q_n in S_r, w_{p_0} and w_{q_0} are convertible, and w_{p_n} and w_{q_n} are convertible. Therefore $w_{p_0} \leq w_{p_n} \leq w_{q_n} + d \leq w_{q_0} + d$, where d is the conversion degree. It implies that w_{p_n} is within the conversion range of w_{q_0}. Recall that as discussed in Section 2.3.2.3.3, the Augment to Full Algorithm has a property that if at the end of an iteration there exists an eligible but unselected queue slot on an output wavelength and an unassigned packet on an input wavelength, then the two wavelengths are not convertible. According to this property, either p_n is assigned, or q_0 must be selected. This contradicts with the previous assumption that both q_0 and p_n are not included after the k_{th} iteration of the Augment to Full Algorithm. Thus, such a list of packets and queue slots cannot be found, and $c_k^{S_{af}} \geq c_k^S$ must hold for any schedule S. Therefore, $c_i^{S_{af}} \geq c_i^S$ holds for $0 \leq i \leq B$ and any S. □

Now by Lemma 1, we can obtain the following corollary.

Corollary 1. *S_{af} is an optimal schedule.*

Proof. Since $c_B^{S_{af}} \geq c_B^S$ for any schedule S, S_{af} schedules the most number of packets out of all possible schedules. And the total delay D_{total} incurred by those packets can be expressed as

$$\begin{aligned} D_{total} &= \sum_{i=1}^{B} i \cdot (c_i^{S_{af}} - c_{i-1}^{S_{af}}) \\ &= B \cdot c_B^{S_{af}} - \sum_{i=0}^{B} c_i^{S_{af}} \end{aligned}$$

As $c_i^{S_{af}} \geq c_i^S$ for $0 \leq i \leq B$ and any S, S_{af} achieves the smallest average delay among all schedules that schedule $c_B^{S_{af}}$ packets. Therefore S_{af} is an optimal schedule. $\square$

2.5 Implementation and Complexity Analysis

In this section we discuss the implementation of the new scheduling scheme, and analyze the complexity of the proposed algorithm.

At the beginning of iteration i of the Augment to Full Algorithm, the set of active output wavelengths, A_i, needs to be computed according to $A_i = A_{i-1} \cup N_i - L_{i-1}$. As an implementation detail, elements in N_i, L_i and A_i are kept in an ascending order. At the beginning of each time slot, we scan the output wavelengths in an ascending order, and add each wavelength to the corresponding set in $\{N_1, N_2, \ldots, N_B\}$ according to the queue length on that wavelength. Thus $\{N_1, N_2, \ldots, N_B\}$ can be obtained in $O(W)$ time. Keeping elements in order for N_i is trivial since its elements, if any, are added to the set in an ascending order. L_i is obtained by scanning A_i at the end of the i_{th} iteration. If the i_{th} queue slot on a wavelength $w \in A_i$ is not selected in this iteration, w is added to L_i. Clearly, elements of L_i can be kept in order without extra cost if elements of A_i are in order. In particular, we implement A_i as a linked list. As a result, $A_{i-1} \cup N_i$ can be done in time linear to $|A_{i-1} \cup N_i| = O(|A_i|)$. Similarly, all the deletions of elements of L_{i-1} take $O(|A_i|)$.

During the filling process, we traverse the linked list representing A_i, and use them one by one to fill the input wavelengths. Recall that the conversion range of a wavelength is bounded by $2d+1$, which is also the maximum number of input wavelengths an output wavelength can fill in each filling process. Thus A_i tries to fill at most $(2d+1)|A_i| = O(|A_i|)$ input wavelengths, since d is a small constant. Besides, with the help of the auxiliary stack for blocking resolving, it takes only constant time to resolve a blocking. However, we do need to push as many as $|A_i|$ items into the stack during the whole filling process. After the filling process along with blocking resolution, the schedule construct algorithm is called and the remaining packets on an input wavelength may need update. This also takes $O(|A_i|)$ time. Suppose that the algorithm terminates at iteration s. Then by adding up all the above, we have

$$T_{af} = O(W) + \sum_{i=0}^{s} O(|A_i|)$$

where T_{af} is the time complexity of the Augment to Full Algorithm.

In iteration i, $|A_i|$ queue slots with index i are taken into consideration by the Augment to Full Algorithm. $|L_i|$ of them will not be selected, and lead to the lock of the corresponding wavelength; the remaining $|A_i| - |L_i|$ are selected in iteration i and will be used in the final schedule. Note that the total number of queue slots is BW. On the other hand, in a single time slot there are up to NW packet arrivals, hence at most NW queue slots are needed to schedule all the packets. In other words, when the algorithm terminates at iteration s, it must be true that

$$\sum_{i=0}^{s} |A_i| - |L_i| \leq \min\{NW, BW\}$$

It is also clear that

$$\sum_{i=0}^{s} |L_i| \leq \sum_{i=0}^{s} |N_i| \leq W$$

Thus

$$\begin{aligned}\sum_{i=0}^{s}|A_i| &\leq \min\{NW, BW\} + \sum_{i=0}^{s}|L_i| \\ &= O(\min\{NW, BW\})\end{aligned}$$

Therefore,

$$\begin{aligned}T_{af} &= O(W) + \sum_{i=0}^{s} O(|A_i|) \\ &= O(\min\{NW, BW\})\end{aligned}$$

On the other hand, since for each time slot, any algorithm has to determine whether each of the packets should be transmitted, or whether each of the queue slots should be occupied, the lower bound of the scheduling problem is $O(\min\{NW, BW\})$. Thus the complexity of the Augment to Full Algorithm asymptotically matches the lower bound of the problem. The low time complexity of our algorithm is mainly due to the properties of the switch discussed earlier: the output buffer can be treated as a FIFO queue on each output wavelength channel. As a result, available buffer slots on an output wavelength channel are consecutive.

2.6 Performance Evaluations

We have conducted extensive simulations to evaluate the performance of the proposed scheduling algorithm. The main performance criteria considered are packet loss probability and average queuing delay. Two switches are simulated: one has 16 input fibers and 16 output fibers, with 16 wavelengths on each fiber, and the other has 8 input fibers and 8 output fibers, with 4 wavelengths on each. By doing so we expect to rule out the possibility that certain performance measures are

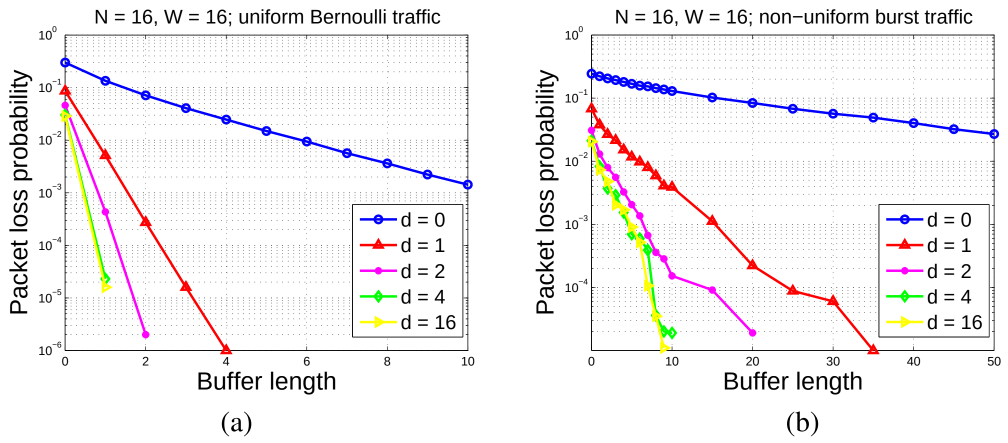


Figure 2.9: Packet loss probability versus buffer length with $\rho = 0.8, N = 16, W = 16$. (a) Uniform Bernoulli traffic. (b) Non-uniform burst traffic with geometric distribution.

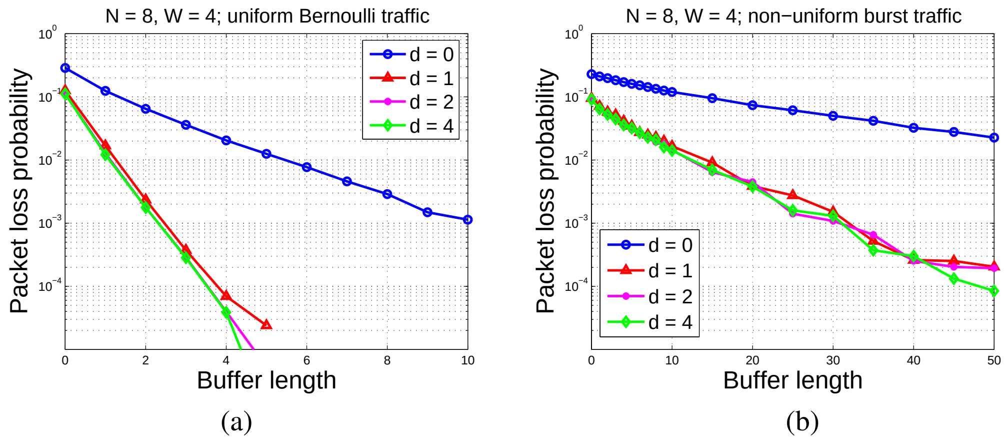
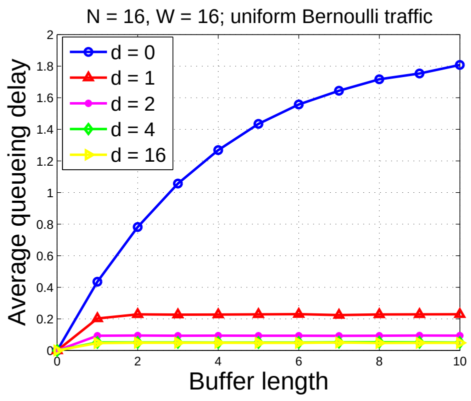


Figure 2.10: Packet loss probability versus buffer length with $\rho = 0.8, N = 8, W = 4$. (a) Uniform Bernoulli traffic. (b) Non-uniform burst traffic with geometric distribution.

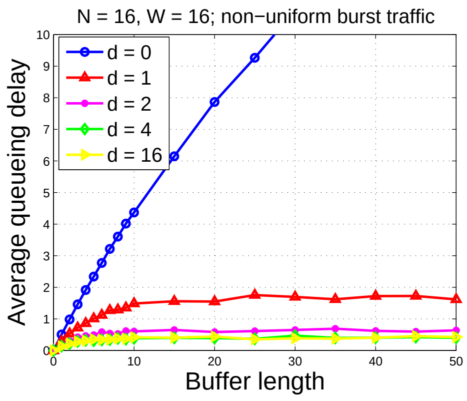
dependent on any particular value of the switch size, the number of wavelength channels, or the ratio of the two.

We conducted simulations under both Bernoulli uniform traffic and burst non-uniform traffic. Bernoulli arrival assumes that at the beginning of each time slot the probability that there is a packet arriving on any input wavelength is solely determined by the traffic intensity ρ. Under the burst arrival model, the status of an input wavelength channel alternates between "on" and "off." At the beginning of each time slot there is always a packet arriving on input channels which are "on," and no packet will arrive during an "off" period. The length of a state may follow different distributions. When it follows geometric distribution, the traffic is the ideal on-off traffic model [33]. The average burst length is 10 in the simulations. Under the non-uniform traffic model, there is a hotspot output fiber for each input wavelength channel. The possibility that a packet arriving at an input wavelength channel is destined for the corresponding hotspot output fiber and any other output fiber is p and $(1-p)/N$, respectively. In the simulations p is set to 50%. The simulation data were obtained by running each simulation for 10^6 time slots, and the results are plotted in Fig. 2.9 to Fig. 2.12.

It can be seen from Fig. 2.9 and Fig. 2.10 that when neither buffer nor wavelength conversion are available (i.e., $d = 0$ and $B = 0$), in both simulated switches the packet loss probability is almost the same regardless of the traffic models. Nevertheless, as shown in Fig. 2.9(a) and Fig. 2.10(a), under uniform Bernoulli traffic, FDLs that can store a few packets will suffice to bring the loss probability to an acceptable level, even when only very limited wavelength conversion is available. For example, $d = 1$ and $B = 4$ produces a loss probability smaller than 10^{-4} in both Fig. 2.9(a) and Fig. 2.10(a). While under non-uniform bursty traffic, the packet loss probability drops rather slowly with the increase of the buffer length. However, under both traffic models, the improvement in packet loss probability is significant when wavelength conversion becomes available. Especially under burst traffic where a small buffer can hardly help reduce the packet loss



(a)



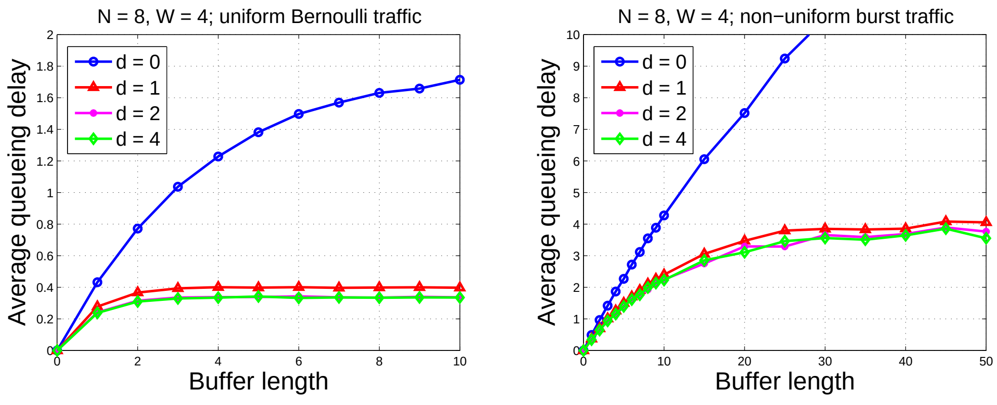
(b)

Figure 2.11: Average packet queuing delay versus buffer length with $\rho = 0.8, N = 16, W = 16$. (a) Uniform Bernoulli traffic. (b) Non-uniform burst traffic.

probability, the increase in wavelength conversion degree from 0 to 1 is crucial as can be seen in Fig. 2.9(b) and Fig. 2.10(b). Meanwhile, if we compare Fig. 2.10 with Fig. 2.9, it is not difficult to see that, when $W = 4$, i.e., each fiber contains only 4 wavelength channels, the effect of increasing the wavelength conversion capability is not as significant as that when $W = 16$.

Fig. 2.11 and Fig. 2.12 illustrate the average packet queuing delays under each traffic model. Under uniform Bernoulli traffic, the average queuing delay converges very fast with the increase of the buffer length as long as $d > 0$. In Fig. 2.11(a) and Fig. 2.12(a), when $d > 0$, the average queuing delay becomes rather stable after B increases to 2. However, under burst traffic (Fig. 2.11(b) and Fig. 2.12(b)), it requires a larger buffer for such converge to occur. On the other hand, the converged average queuing delay with full-range wavelength conversion is close to that with a much smaller conversion degree, which implies that for an optical packet switch the ability of wavelength conversion is critical, while it is not necessarily to be full-range conversion. A similar observation can be drawn for packet loss probability.

In summary, wavelength conversion is a unique method for optical networks to resolve channel contention. A scheduler can efficiently regulate the traffic at the switch input by making use of



(a) (b)

Figure 2.12: Average packet queuing delay versus buffer length with $\rho = 0.8, N = 8, W = 4$. (a) Uniform Bernoulli traffic. (b) Non-uniform burst traffic.

wavelength conversion. As a result, system performance can greatly benefit from the reduction of traffic burstness.

2.7 Conclusions

In this chapter we have studied packet scheduling in WDM optical packet switches with output buffer and limited-range wavelength conversion. We have shown that the output buffer can be viewed as a separate FIFO on each output wavelength channel. We have formalized the problem of finding an optimal schedule in such a switch into a minimum cost maximum flow problem, and presented a new scheduling scheme, called the Augment to Full Algorithm to find an optimal schedule. The new scheduling algorithm is able to find an optimal schedule in $O(\min\{NW, BW\})$ time, where N is the switch size, W is the number of wavelength channels per fiber and B is the length of the longest FDL at the output of the switch. Compared to directly applying a generic network flow algorithm, the Augment to Full Algorithm is much more efficient and asymptotically matches the lower bound of the scheduling problem. This optimal scheduling algorithm can be applied to any output-buffered WDM optical switches whose output buffer can be modeled as a queue on each wavelength channel. Simulations were conducted to test the performance of the proposed scheduling algorithm under different traffic models.

Table 1: Schedule construction from vectors $\mathcal{I}$ and $\mathcal{O}$

Input: $\mathcal{I} = \{x_i | i = 1, 2, \ldots, W\}$, $\mathcal{O} = \{y_i | i = 1, 2, \ldots, W\}$
Output: a schedule with the given $\mathcal{I}$, $\mathcal{O}$ and minimum delay
Notations:
in, out: pointers to input/output wavelengths currently being worked with, respectively
p_{in}: number of remaining packets on input wavelength in to be scheduled
q_{out}: number of remaining queue slots on output wavelength channel out to be used

Algorithm:

```
for each wavelength $i$
  $p_{in} \leftarrow x_{in}, q_{out} \leftarrow y_{out}$
end for
$in \leftarrow 1, out \leftarrow 1$
while $in \leq W$ and $out \leq W$
  if $q_{out} \leq p_{in}$
    /*schedule $q_{out}$ packets from $in$ to $out$*/
    $p_{in} \leftarrow p_{in} - c_{out}, out \leftarrow out + 1$
  else
    /*schedule $p_{in}$ packets from $in$ to $out$*/
    $q_{out} \leftarrow q_{out} - p_{in}, in \leftarrow in + 1$
  end if
  if $in > \min\{W, out + d\}$
    $out \leftarrow out + 1$
  end if
  if $out > \min\{W, in + d\}$
    $in \leftarrow in + 1$
  end if
end while
```

Table 2: The Augment to Full Algorithm

Input: request graph
Output: an optimal schedule
Notations:
x_j: number of assigned packets on input wavelength j in current iteration
y_j: number of selected queue slots on output wavelength j in current iteration
$\hat{x}_j$: number of packets on input wavelength j not assigned to locked output wavelengths
l_j: length of queue on output wavelength j at the beginning of current time slot
q_j: number of queue slots on wavelength j that can still be selected in current iteration.
p_j: number of packets to be assigned on input wavelength j
in, out: pointers to input/output wavelengths currently being filled or filling, respectively
s_j: iteration at which output wavelength channel j is locked
f: number of blocked queue slots

Algorithm:
```
for $i = 0$ to $B$
  exit if all output wavelengths locked, or all inputs filled by locked output wavelengths.
  $A_i \leftarrow A_{i-1} \cup N_i - L_{i-1}, L_i \leftarrow \emptyset$
  /* $i + 1 - l_j$ is the total number of eligible queue slots on wavelength $j$ in iteration $i$ */
  $q_j \leftarrow i + 1 - l_j, y_j \leftarrow 0$ for all $j \in A_i$
  $out \leftarrow$ the first active output wavelength, $in \leftarrow \max\{0, out - d\}, x_{in} \leftarrow 0, p_{in} \leftarrow \hat{x}_{in}$
  while $in \leq W$ and $out \leq W$
    if $q_{out} \leq p_{in}$
      $x_{in} \leftarrow x_{in} + q_{out}, y_{out} \leftarrow y_{out} + q_{out}, p_{in} \leftarrow p_{in} - q_{out}$
      push $out$ into the auxiliary stack
      $out \leftarrow$ next active output wavelength
    else
      $x_{in} \leftarrow x_{in} + p_{in}, y_{out} \leftarrow y_{out} + p_{in}, q_{out} \leftarrow q_{out} - p_{in}$
      $in \leftarrow in + 1, p_{in} \leftarrow \hat{x}_{in}$
    end if
    if $in > \min\{W, out + d\}$
      /*$f$ queue slots are blocked on $out$*/
      $f \leftarrow q_{out} - 1, y_{out} \leftarrow y_{out} + f$
      pop the top $f$ elements from the auxiliary stack, $y_w \leftarrow y_w - 1$ if $w$ popped
      $out \leftarrow$ next active output wavelength
    end if
    if $out > \min\{W, in + d\}$
      $in \leftarrow out - d$
    end if
  end while
  construct intermediate schedule based on $\{y_j \mid j \in A_i\}$ and
   $\{x_{j'} \mid$ wavelength $j'$ convertible to wavelength $j \in A_i\}$
  for each $j \in A_i$ s.t. $y_j < i + 1 - l_j$
      $L_i \leftarrow L_i \cup j, s_j \leftarrow i$
      update $\hat{x}_{j'}$ for all wavelength $j'$ convertible to wavelength $j$
  end for
end for
```

Chapter 3

Achieving 100% Throughput in Input-Buffered WDM Optical Packet Switches

In previous chapter, we have studied packet scheduling in output-buffered optical switches. This chapter focuses on input-buffered switches and is organized around the maximum throughput that can be achieved with such interconnect architecture. More specifically, packet scheduling algorithms that deliver 100% throughput under various types of traffic enable a switch to achieve its full capacity. While such algorithms have been proposed for electronic switches, they cannot be directly applied to WDM optical switches due to the following reasons. First, the optical counterpart of electronic random access memory (RAM) is absent; Second, the wavelength conversion capability of WDM switches changes the conditions for admissible traffic. To address these issues, in this chapter, we first introduce a new fiber-delay-line (FDL) based input buffering fabric that is able to provide flexible buffering delay, followed by a discussion on the conditions that admissible traffic must satisfy in a WDM switch. We then propose a weight-based scheduling algorithm, named Most-Packet Wavelength-Fiber Pair First (MPWFPF), and theoretically prove that given a buffering fabric with flexible delay, MPWFPF delivers 100% throughput for input-buffered WDM switches with no speedup required. Finally, we further propose the WDM-iSLIP algorithm, a generalized version of the iSLIP algorithm, for WDM switches, which efficiently finds an approximate optimal schedule with low time complexity. Extensive simulations have been conducted to verify the theoretical results, and test the performance of the proposed scheduling algorithms in input-buffered WDM switches.

The rest of the chapter is organized as follows. Section 3.1 presents some background and related work. Section 3.2 presents the new FDL-based buffering fabric and gives the conditions for admissible traffic in WDM switches. Section 3.3 presents the Most-Packet Wavelength-Fiber Pair First scheduling algorithm and proves that it achieves 100% throughput. Section 3.4 describes the WDM-iSLIP algorithm. Section 3.5 gives the simulation results. Section 3.6 concludes the chapter.

3.1 Background and Related Work

Similar to many other works in the literature [2] [34] [35], packets at the switch input are assumed to have a fixed size and arrive in a synchronized manner. Such packets are also referred to as *cells*. In a system with variable-length packets, it can be realized by segmenting packets into cells at the switch input and reassembling cells into packets at the output. Possible implementations of synchronous arrival of packets in optical domain are discussed in, for example, [36].

For electronic switches, many scheduling algorithms have been proposed by formalizing the scheduling problem into a matching problem. McKeown, etc. showed [2] that though maximum size matching algorithms schedule the maximum number of packets in each time slot, such algorithms may not guarantee optimum throughput if the traffic is admissible but non-uniform. Instead, maximum weighted matching based algorithms have been proved to be able to achieve 100% maximum throughput when the incoming traffic is i.i.d. [2] [37]. It was further shown in [38] that the Longest Queue First (LQF) algorithm proposed in [2] delivers 100% throughput as long as input traffic is admissible and satisfies the strong law of large numbers (SLLN). To reduce the computing complexity, parallel iterative matching based approximate algorithms, such as iSLIP [5], have been proposed and implemented in commercial products. iSLIP finds a maximal matching by breaking down the matching process into i iterations. During each iteration, each unmatched input sends a request to all outputs it has a packet destined to, then each unmatched output chooses one to grant among all requests it received in a round robin manner. Finally, each input selects one grant if there is any, and updates its state. Tradeoff between implementation complexity and performance can be achieved by adjusting the number of iterations, i.

However, WDM optical switches differ from electronic switches in at least two aspects. First, as mentioned in 1.2, due to the absence of optical RAM, currently the most common optical buffering method is to use fiber delay lines and switches. It was proposed in [39] to use FDLs and a large switching fabric to emulate a single input, single output optical priority queue. A switch with $\sqrt{K}$ input and output ports and FDLs of a total length $K + O(\sqrt{K})$ are required to emulate a priority queue with length K. In [40] the switch size was reduced to $\sqrt[3]{K}$. Recently, a construction of an optical FIFO was presented in [41], which requires $2n$ 2×2 switches and a total fiber length $3 \cdot 2^{n-1} - 2$ for an optical FIFO queue with buffer size $2^n - 1$. From the above discussion, it can be seen that while common in electronic switches, the implementation of a VOQ becomes rather complex in optical domain. Besides, to implement VOQ for a WDM optical switch with M input fibers and N output fibers where each fiber contains k wavelength channels, at least kMN queues are needed, which is k times of that in an $M \times N$ electronic switch. Hence the overall implementation complexity tends to be overwhelming. Thus a buffering fabric that is more practical than VOQ is desired.

The second fundamental difference lies in the conditions for admissible traffic. Traditionally, an arrival rate is defined between each pair of input and output ports. The traffic is considered as "admissible" if and only if the sum of the arrival rates from one input to all outputs, as well as that from all inputs to one output, is smaller than 1, in other words, no input or output port is oversubscribed. However, most of WDM packet switches are equipped with wavelength converters, which convert the wavelength of an optical signal to another wavelength to resolve output contention. As we will see later, due to wavelength conversion, arrival rates between each pair of input/output wavelength channels cannot be well defined in WDM switches. Because of these differences between electronic and WDM optical switches, most existing analyses and conclusions on the performance of electronic switches cannot be directly applied to WDM optical switches.

In this chapter, we propose practical solutions to solve the above problems. We will first introduce a new FDL-based buffering fabric which enables flexible buffering delay. Then we will discuss the conditions that admissible traffic must satisfy in a WDM optical switch. We will prove that input-buffered WDM switches using a buffering fabric with flexible delay are theoretically guaranteed to achieve 100% throughput under any admissible traffic with any wavelength conversion model, when working with a weight-based scheduling algorithm, in particular, the proposed Most-Packet Wavelength-Fiber Pair First scheduling algorithm. To the best of our knowledge, this is the first work that theoretically proves WDM packet switches are able to deliver 100% throughput with no assumption on traffic or wavelength conversion patterns. Furthermore, to reduce the scheduling complexity, we propose a more practical, parallel iterative matching based scheduling algorithm, WDM-iSLIP, that can efficiently determine an approximate optimal scheduling with much lower time complexity.

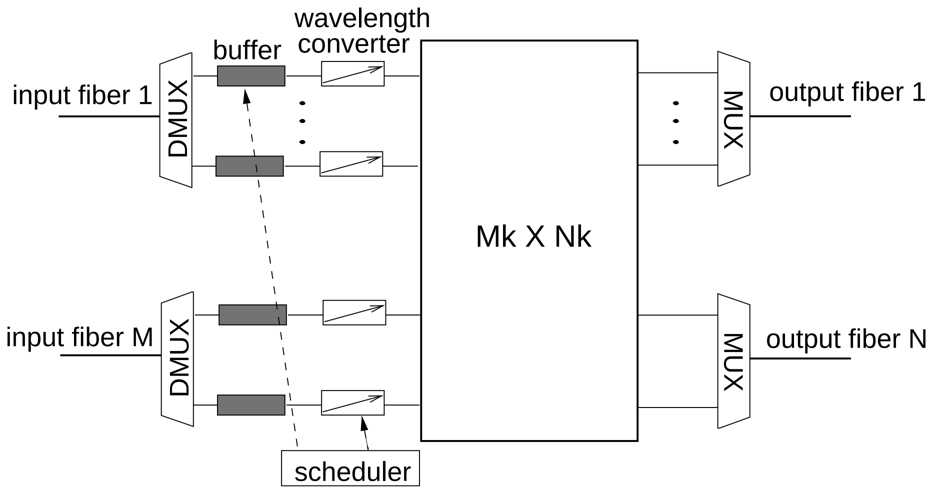


Figure 3.1: Architecture of the input-buffered WDM optical packet switch.

3.2 Input-Buffered WDM Optical Packet switches

The basic architecture of the input-buffered WDM optical packet switch considered in this chapter is shown in Fig. 3.1. In the switch, there is a demultiplexer at each input fiber that demultiplexes the incoming optical signal to k signals, one on each wavelength. There are also a buffer and a wavelength converter on each input wavelength channel that are controlled by the central scheduler. Finally, there is a multiplexer on each output fiber that multiplexes the signals on wavelength channels back into an output fiber at the output of the switch. In the next few subsections, we discuss several specific issues in such a WDM optical packet switch, including wavelength conversion, controllable FDL buffer and admissible traffic.

3.2.1 Wavelength Conversion

We define C_w as the set of accessible wavelengths of wavelength w, i.e. the wavelengths that w can be converted to, plus itself. C_S is similarly defined for a set of wavelengths, S. Taking the limited-range conversion model in previous chapter as an example, in Fig. 2.1, wavelength 1

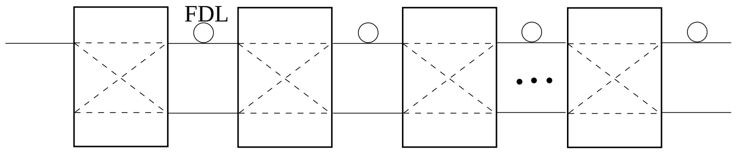


Figure 3.2: The basic architecture of conventional feed-forward FDL buffers.

can be converted to wavelengths 2 and 3, thus $C_1 = \{1, 2, 3\}$. Similarly, $C_{\{1,2\}} = \{1, 2, 3, 4\}$ and $C_{\{3,6\}} = C_{\{2,4\}} = \{1, 2, 3, 4, 5, 6\}$.

It is worth pointing out that we make *no* assumptions on wavelength conversion patterns in this chapter. In other words, our theoretical analyses to be presented in the following sections do not depend on any specific conversion patterns and are valid for the general case.

3.2.2 Controllable FDL Buffer

There has been some work considering adding FDLs to optical switches in the literature. In [34, 42], fixed FDLs were considered, where a light signal is buffered for a fixed amount of time by the FDLs. Controllable FDL buffers were considered by combining FDLs with switches in [43] [44] [45]. The buffers in these works share the same basic architecture as shown in Fig. 3.2. A packet may be delayed or not between each two consecutive switches depending on how the first switch is configured. A potential problem with this architecture is that it has at most two "exits," both located at the end of the buffer. As a result, a packet has to traverse all of the switches once entering the buffer, regardless how long it needs to be buffered. This may result in undesirable power loss and crosstalk.

In this chapter, we propose to use a controllable FDL buffer with multiple exits. Its diagram is shown in Fig. 3.3 and the functionalities can be explained as follows.

The controllable FDL buffer is composed of several cascaded FDLs connected by 1×2 all-optical switches, whose implementation can be found in, for example, [46]. When a packet reaches

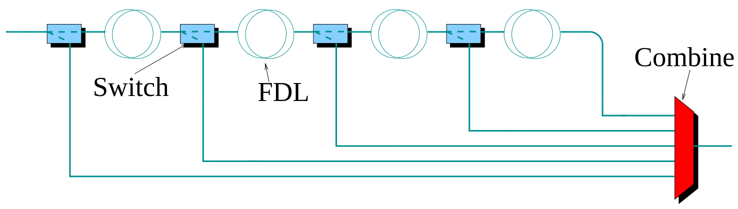


Figure 3.3: A controllable FDL with multiple "exits."

the end of an FDL segment, by default it is sent to the next FDL through one output port of the 1×2 switch. The scheduler configures the switch and let a packet leave the buffer through the other output of the switch only when the packet is ready to be transmitted to the output. To avoid conflicts, at any time only one packet can exit from each buffer. Under this structure, once entered the FDL buffer, a packet can leave the buffer through any of the exits, therefore the delay time is no longer fixed and can be controlled by the packet scheduler. The number of exits is limited by the size of the combiner, which can be implemented by a many-to-one optical coupler. With current technology, such a coupler can have 128 or more inputs [11]. The number of inputs can be further extended by coupler cascading if needed. Besides, having K exits does not mean that the FDL can hold at most K packets. The FDL between two exits can be longer than the unit length (the length needed to provide one time slot delay), hence can hold multiple packets.

It is interesting and important to see what kind of benefits can be achieved if controllable FDL buffers are used as the input buffer for a WDM packet switch. Note that there are many flexibilities in such a controllable FDL. For example, in each time slot we can let zero or one packet exit from the buffer on each input wavelength and be transmitted to the output of the switch. Also, we can select the wavelengths on which packets can be transmitted and choose which packets can leave the FDL buffer, depending on their destination fiber and the availabilities of wavelength channels on these fibers. Hence, an algorithm is required to schedule the packets to achieve goals such as maximum throughput.

3.2.3 Admissible Traffic for WDM Packet switches

The next issue we will address is the conditions for admissible traffic for WDM packet switches. In an $M \times N$ electronic switch, an arrival process at input i for output j at rate $\lambda_{i,j}$ (normalized to packet arrivals per time slot) can be denoted as $A_{i,j}(\cdot)$, where $A_{i,j}(n)$ stands for the cumulative number of packets destined for output j that have arrived at input i by the n_{th} time slot. The set of all arrival processes $A(\cdot)$, which is defined as $A(\cdot) = \{A_{i,j}(\cdot), i = 1, 2, \ldots, M, j = 1, 2, \ldots, N\}$, is considered admissible if

$$\sum_i \lambda_{i,j} < 1 \text{ and } \sum_j \lambda_{i,j} < 1 \tag{3.1}$$

It can be seen that traffic being admissible is equivalent to that no input or output is oversubscribed.

It is worth pointing out that $\{A_{i,j}(\cdot), i = 1, 2, \ldots, M, j = 1, 2, \ldots, N\}$ are usually assumed to be independent, or at least to be jointly stationary processes. A process is stationary if its statistical properties, such as the mean and variance, do not change over time. Two processes $X(t)$ and $Y(t)$ are jointly stationary if both of them are stationary, and their cross-correlation function $R_{XY}(t, t+\tau) = E[X(t)Y(t+\tau)]$ depends only on the time interval τ but not on the starting time t. This assumption holds in electronic switches. For WDM switches that do not have wavelength conversion capability, the assumption also holds. Consider a WDM packet switch with M input fibers and N output fibers. There are k wavelengths multiplexed on each fiber, each of which operates at rate r. Without wavelength conversion, such a system can be simply considered as k $M \times N$ traditional crossbar switches since packet scheduling on different wavelengths is independent, and the rate of each input/output is the rate of a wavelength, r. However, since wavelength conversion is an important method to resolve resource contention in WDM packet switches, in most of existing designs of WDM switches, wavelength conversion capability was considered [13, 35] [47] [34] [14] [48] [49]. Now the problem is that with wavelength conversion, packet scheduling on different wavelengths is dependent. In other words, if we define an arrival process for each pair of input/output wavelength channel, these processes are no longer independent. Such processes are not jointly stationary neither, since they depend on not only the arrival of packets but also the scheduler. As a result, the previous assumption may no longer be valid.

From the above discussion, we can see that in WDM switches it may not be appropriate to define an arrival process per input/output wavelength channel pair. Furthermore, it cannot work well either if an arrival process is defined for each pair of input fiber/output fiber. The reason is that even the sum of arrival rates is well bounded for each fiber, it never guarantees that no wavelength channel is oversubscribed. In other words, traffic that seems admissible at fiber level may be inadmissible at wavelength granularity. Therefore, for wavelength w on input fiber i, we define $A_{i,j}^{w}(n)$ as the cumulative number of packets destined for output fiber j that have arrived on wavelength w at input fiber i by the n_{th} time slot. It is easy to see that $\{A_{i,j}^{w}(\cdot), i = 1, 2, \ldots, M, j = 1, 2, \ldots, N, w = w_1, w_2, \ldots, w_k\}$ are independent processes. We can then define the rate of each arrival process $A_{i,j}^{w}(\cdot)$ as long as it satisfies the strong law of large numbers (SLLN), i.e., with probability one, $\lim_{n\to\infty} A_{i,j}^{w}(n)$/n = $\lambda_{i,j}^{w}$, where $\lambda_{i,j}^{w}$ is the arrival rate. As pointed out in [38], this is true for almost all real traffic processes.

With the arrival rates defined, we now return to the question that what conditions admissible traffic must satisfy in WDM packet switches. Apparently, we can no longer obtain equations as neat as (3.1), although the basic rule is the same, i.e. no channel should be oversubscribed. First, for each input fiber, the sum of arrival rates from one wavelength channel to all the output fibers should be smaller than 1. Thus we have the following equation:

$$\sum_{j} \lambda_{i,j}^{w} < 1 \tag{3.2}$$

for all i and w as defined above. At the same time, the sum of arrival rates on a particular wavelength over all input fibers that are destined to the same output fiber cannot exceed the cardinality of the set of accessible wavelengths of that wavelength, i.e.

$$\sum_i \lambda_{i,j}^w < |C_w| \tag{3.3}$$

for all j and w. Note that $|C_w|$ can be larger than 1, which makes the above condition looser than its counterpart for electronic switches. For example, in a switch with the wavelength conversion model given in Fig. 2.1, the sum of arrival rates on wavelength 1 over all input fibers and destined to a particular output fiber can be as large as 3, since incoming packets on wavelength 1 can be scheduled to wavelength channels 1, 2 or 3 of that output fiber. Furthermore, as explained earlier, packet scheduling on convertible wavelengths is dependent. Thus, for any set of wavelengths, S, the sum of arrival rates on these wavelengths that are destined to a particular output fiber over all input fibers cannot exceed the cardinality of the accessible wavelength set of S. In other words,

$$\sum_{w \in S} \sum_i \lambda_{i,j}^w < |C_S| \tag{3.4}$$

must hold for all j and S. In fact, (3.3) is a special case of (3.4) when the set contains only one wavelength. Thus the traffic is admissible if and only if the arrival rates satisfy (3.2) for every input wavelength channel and (3.4) for every output fiber and combination of input wavelengths.

3.3 Most-Packet Wavelength-Fiber Pair First Algorithm

In this section we give an optimal packet scheduling algorithm for the WDM packet switches described in the previous section, called the Most-Packet Wavelength-Fiber Pair First Algorithm (MPWFPF), and prove that it is theoretically guaranteed to deliver 100% throughput under any admissible traffic.

The basic idea of MPWFPF is to favor input wavelength channel/output fiber pairs that are more congested than others. Let $Z_{i,j}^{w}(n)$ denote the number of packets destined to output fiber j that are buffered on wavelength w of input fiber i at the beginning of time slot n, then we have

$$Z_{i,j}^{w}(n) = A_{i,j}^{w}(n) - D_{i,j}^{w}(n) \tag{3.5}$$

where $D_{i,j}^{w}(n)$ is the cumulative number of departed packets that came from wavelength w of input fiber i to output fiber j up to time slot n. Let $S^{w}(n)$ be an $M \times N$ matrix, such that each of its element $S_{i,j}^{w}(n)$ is the *service indicator* and

$$S_{i,j}^{w}(n) = \begin{cases} 1, & \text{if a packet on wavelength } w \text{ of input } i \text{ scheduled to output } j \text{ in time slot } n \\ 0, & \text{otherwise} \end{cases} \tag{3.6}$$

Note that there are a total of k such matrices $S^{w}(n)$, one for each wavelength. To simplify our proof later, we merge the k matrices into a single $kM \times N$ "super" matrix $\pi(n)$, such that $\pi(n) = [S^{1}(n) \;\; S^{2}(n) \;\; \dots \;\; S^{k}(n)]$, or equivalently, $\pi_{i',j}(n) = S_{i,j}^{w}(n)$, where $i' = i + M(w-1)$, for $1 \leq w \leq k$, $1 \leq i \leq M$ and $1 \leq j \leq N$. We refer to $\pi(n)$ as the $schedule$ matrix in the following. Similarly, we define $\mathcal{Z}(n), \mathcal{R}$ and $\mathcal{D}(n)$, which are the merged formats for $Z_{i,j}^{w}(n), \lambda_{i,j}^{w}$ and $D_{i,j}^{w}(n)$, respectively. $\mathcal{Z}(n), \mathcal{R}$ and $\mathcal{D}(n)$ are called the $weight$ matrix, $rate$ matrix, and $departure$ matrix, respectively.

Note that with buffer at the switch input, packet scheduling for different output fibers is dependent, since packets buffered on the same input wavelength channel destined to different output fibers will compete for transmission. In Table 1, we show the function that MPWFPF performs in a time slot, say, slot n, which is basically to find under certain constraints the optimal schedule matrix $\pi(n)$ (or equivalently the service indicator matrices $S^{w}(n)$ for all w). In other words, each pair of input wavelength/output fiber is assigned a weight, which equals the number of packets that belong to this pair and are currently being buffered. The more packets buffered, the larger the input wavelength channel/output fiber pair is weighted, and the higher priority the pair needs to be served. As a result, it is unlikely that the number of packets on an input wavelength destined to an output fiber goes to infinity. The constraints assure that $\pi(n)$ is feasible. For instance, the value of an element of $\pi(n)$ can only be 0 or 1, since any element of $\pi(n)$ is a service indicator. Also, the sum of each column of $\pi(n)$ must be smaller than or equal to 1 since there is no speedup and each input wavelength channel can transmit at most one packet during each time slot. However, this does not hold for each row of $\pi(n)$, as an output fiber can receive as many as k packets in each time slot. Nevertheless, in each time slot, the number of packets coming from a set of wavelengths, W, for example, that an output fiber can receive is at most C_W, because those packets can only be converted and sent to C_W out of all the wavelength channels on that output fiber. They cannot access any of the remaining wavelength channels on that output fiber even those channels are free, since they cannot be converted to those wavelengths.

An interesting observation is that the conditions $\pi(n)$ must satisfy are exactly the conditions admissible traffic must satisfy, given by the equations in Section 3.2.3, plus that $\pi_{i,j}(n)$ can only be 0 or 1 for all i and j. It should not be surprising though, since the constrains are essentially the same, i.e., no input or output wavelength channel should be oversubscribed.

Next we show that theoretically MPWFPF can deliver 100% throughput under any admissible traffic. Similar to other works in the literature on the proof of 100% throughput, we adopt the assumption that there is no packet loss due to buffer overflow at the input of the switch. It requires that the controllable FDL buffer has sufficiently large capacity. Practically, this can be ensured by adopting a large electronic or all-optical backup buffer for the controllable FDL, such as those proposed in [48] [50], to store packets if the controllable FDL buffer is full. When the two buffers are coordinated well, the minimum length of the controllable FDL on each input wavelength can

Table 1: Most-Packet Wavelength-Fiber Pair First Algorithm

```
Input – weight matrix $\mathcal{Z}(n)$
Output – schedule matrix $\pi(n)$

Find $\pi(n)$ that maximizes
        $\langle\pi(n), \mathcal{Z}(n)\rangle = \sum_{ij} \pi_{i,j}(n)\mathcal{Z}_{i,j}(n)$
such that
        $\pi_{i+M(w-1),j}(n) = S^w_{i,j}(n)$, and
        $S^w_{i,j}(n)$ = 0 or 1,
        $\sum_j S^w_{i,j}(n) < 1$,
        $\sum_{w\in W} \sum_i S^w_{i,j}(n) < |C_W|$
        for $1 \le w \le k, 1 \le i \le M, 1 \le j \le N$
        and all possible sets of wavelengths, $W$.
```

be as small as the number of output fibers, so that at least one packet destined to each output fiber can be buffered. As will be seen from the performance evaluations section, the above assumption is quite reasonable and mild. Our simulation results reveal that under MPWFPF, the average buffering time for a packet is within a practical range even when the switch is heavily loaded.

First of all, let $T_\pi(n)$ be the cumulative amount of time that schedule matrix π has been used by time slot n. Then we have

$$\begin{aligned} D^w_{i,j}(n) &= \sum_{l=1}^{n} S^w_{i,j}(l) \cdot 1_{\{Z^w_{i,j}(l)>0\}} \\ &= \sum_{\pi}\sum_{l=1}^{n} \pi_{i,j} 1_{\{Z^w_{i,j}(l)>0\}}(T_\pi(l) - T_\pi(l-1)) \end{aligned} \tag{3.7}$$

where $1_{\{Z^w_{i,j}(l)>0\}} = 1$ if $Z^w_{i,j}(l) > 0$ is true, and 0 otherwise. By applying the fluid method [38] and combining different wavelengths into the super matrix format, the continuous fluid version of equations (3.5) and (3.7) can be written respectively as follows:

for $t \geq 0, 1 \leq i \leq M, 1 \leq j \leq N$,

$$\mathcal{Z}_{i,j}(t) = \mathcal{R}_{i,j} \cdot t - \mathcal{D}_{i,j}(t) \geq 0 \tag{3.8}$$

$$\frac{d\mathcal{D}_{i,j}(t)}{dt} = \sum_{\pi} \pi_{i,j} \frac{dT_{\pi}(t)}{dt}, \text{if } \mathcal{Z}_{i,j}(t) > 0 \tag{3.9}$$

Next we prove an important lemma that will be used in the proof for 100% throughput:

Lemma 2. *For any admissible traffic, the corresponding rate matrix $\mathcal{R}$ must satisfy*

$$\langle \mathcal{R}, \mathcal{Z}(t) \rangle \leq \langle \pi_0, \mathcal{Z}(t) \rangle \tag{3.10}$$

where $\langle X, Y \rangle = \sum_{ij} X_{ij} Y_{ij}$ and π_0 is an optimal schedule matrix that maximizes $\langle \pi, \mathcal{Z}(t) \rangle$ under the conditions given in Table 1.

Proof. We formalize the scheduling problem into a network flow problem as illustrated in Fig. 3.4. Consider a WDM switch with M input fibers, N output fibers and k wavelength channels on each fiber. First, we introduce three sets of nodes I, O and W, which have M, N and kM nodes, respectively. I and O denote the input fibers and output fibers, respectively. Each node in I is connected to k different nodes in W, representing its k wavelength channels. There are N identical $k \times k$ switching blocks, each of which is responsible for the packet scheduling to a particular output fiber. The connection pattern from the inputs to the outputs in a switching block is solely determined by the wavelength conversion pattern in the original WDM switch. Each node in W denoting wavelength i is connected to the i_{th} input of each switching block. All outputs of a switching block are connected to one node in O. We add two more dummy nodes, s and t, such that s is connected to all nodes in I and t is connected to all nodes in O. Every edge between I and W, as well as between the outputs of the switching blocks and O, has unit capacity. The rest of the edges can be assumed to have a capacity of k, which is the upper bound of the edge capacity.

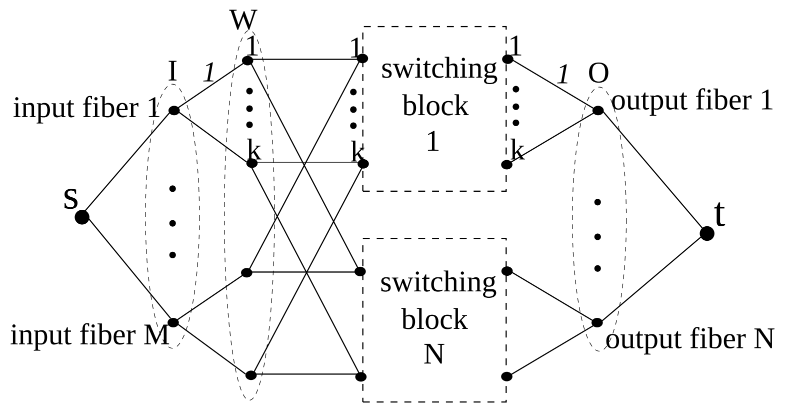


Figure 3.4: Formalizing the scheduling into a network flow problem. Node sets I, O and W represent input fibers, output fibers and input wavelength channels, respectively. For each edge between a node in W and an input of a switching block, a weight is assigned which equals to the number of buffered packets belonging to the corresponding input wavelength / output fiber pair. MPWFPF finds a maximum weighted flow in the flow graph in each time slot.

For each unit flow passing it, an edge between a node of W and one of the inputs of the switching blocks generates a fixed profit, which is its weight. Suppose $x \in W$ is the node representing wavelength i of input fiber j. Then the edge between x and the i_{th} output of switching block l, denoted as $e^i_{j,l}$, has a weight $Z^i_{j,l}(t)$, which is the number of packets currently being buffered on wavelength i at input fiber j that are destined to output fiber l. Such a weight indicates that at this moment if a packet is transmitted on wavelength i of input fiber j to output fiber l, a profit of $Z^i_{j,l}(t)$ is earned.

It can be seen that what MPWFPF performs is equivalent to finding a maximum weighted (profit) flow in the flow graph. Let $\mathcal{F}$ be a $kM \times N$ matrix such that $\mathcal{F}_{i+M(w-1),j}$ equals the amount of flow on edge $e^w_{i,j}$ of a feasible (not necessarily integral) flow in the graph. Then the total profit generated by the feasible flow is given by

$$P(\mathcal{F}) = \langle \mathcal{F}, \mathcal{Z}(t) \rangle \tag{3.11}$$

Since the maximum weighted flow problem is essentially a transshipment problem and the capacities and weights of all edges are integral, it has an integral optimal solution [32]. In other words, there is an integral flow that achieves the maximum profit among all feasible flows. As each integral flow in the flow graph corresponds to a scheduling in the original switch, let π_0 be the schedule matrix that corresponds to the integral optimal flow. Then for any $\mathcal{F}$, we have

$$\langle \mathcal{F}, \mathcal{Z}(t) \rangle \leq \langle \pi_0, \mathcal{Z}(t) \rangle$$

Since any admissible traffic corresponds to a feasible flow in this graph, its rate matrix must satisfy

$$\langle \mathcal{R}, \mathcal{Z}(t) \rangle \leq \langle \pi_0, \mathcal{Z}(t) \rangle$$

That proves the lemma. □

From the above lemma, it follows that

$$\begin{aligned} \langle \mathcal{Z}(t), \frac{d\mathcal{Z}(t)}{dt} \rangle &= \langle \mathcal{Z}(t), \mathcal{R} - \frac{d\mathcal{D}(t)}{dt} \rangle \\ &= \langle \mathcal{Z}(t), \mathcal{R} \rangle - \langle \mathcal{Z}(t), \frac{d\mathcal{D}(t)}{dt} \rangle \\ &= \langle \mathcal{Z}(t), \mathcal{R} \rangle - \langle \mathcal{Z}(t), \sum_{\pi} \frac{\pi dT_{\pi}(t)}{dt} \rangle \\ &= \langle \mathcal{Z}(t), \mathcal{R} \rangle - \langle \mathcal{Z}(t), \pi_0 \rangle \\ &\leq 0 \end{aligned}$$

The second last step holds because π_0 is the schedule matrix employed. In other words,

$$\frac{dT_{\pi_0}(t)}{dt} = 1, \text{ and } \sum_{\pi \neq \pi_0} \frac{dT_{\pi}(t)}{dt} = 0 \tag{3.12}$$

The above result can be interpreted as follows. Whenever $\mathcal{Z}(t)$ becomes larger than 0, $d\mathcal{Z}(t)/dt \leq 0$ holds, i.e., $\mathcal{Z}$ starts to decrease. According to Lemma 1 in [38], it follows that $\mathcal{Z}(t) = 0$ for almost every $t \geq 0$. Thus the fluid model is weakly stable [38], which means that by using MPWFPF the corresponding switch delivers 100% throughput. Note that in the above proof, no assumption was made on the traffic pattern or wavelength conversion model, nor was there any requirement on speedup. Therefore, by combining these results, we have the following theorem.

Theorem 1. *At speedup 1, the MPWFPF algorithm achieves 100% throughput for an input-buffered WDM packet switch which uses the controllable FDL as input buffer under any admissible traffic that satisfies SLLN and any wavelength conversion pattern.*

Note that although the above proof is based on the controllable FDL buffer, the conclusion does not limited to it. For example, if in the future all-optical VOQs become available, the proof is completely valid with VOQs implemented at the switch input.

Now let's analyze the complexity of this approach. As we know, under MPWFPF a maximum weighted flow needs to be computed in each time slot. In the flow graph in Fig. 3.4, finding a maximum weighted flow with the state-of-the-art algorithms needs $O((M+N)MNk^2 \log k \log C)$ time, where C is the upper bound of the edge weight, i.e., the maximum value of $Z_{i,j}^{w}(t)$ at a given time t for all i, j and w. Thus while MPWFPF is important as it is theoretically proved to deliver 100% throughput, its complexity is relatively high. In the next section we will propose a fast approximate scheduling algorithm that has much lower scheduling complexity.

3.4 WDM-iSLIP Algorithm

As discussed in the previous section, similar to the maximum weighted matching based scheduling algorithms in electronic switches, the MPWFPF algorithm, while of theoretical importance since it is guaranteed to delivery 100% throughput for WDM optical packet switches, requires a relatively complex scheduler. In this section we give a fast scheduling algorithm for the WDM packet switches, called WDM-iSLIP algorithm, which is able to find an approximate optimal schedule with lower complexity.

As the name implies, WDM-iSLIP is essentially a generalized version of iSLIP algorithm for WDM switches. The iSLIP algorithm is a variation of the basic round-robin matching algorithm (RRM). Traditionally, RRM maintains a round-robin schedule and a pointer for each input port and output port and performs matching in iterations. Each iteration consists of the following three steps:

Request Step: Each unmatched input sends a request to every unmatched output that it has at least one queued packet.

Grant Step: Each output, if receiving any requests, chooses the one that appears first in its round-robin schedule from the current pointer, and sends a grant message to the corresponding input. The pointer is updated to the next location of the granted input.

Accept Step: Each input, if receiving any grants, accepts the one that appears first in its round-robin schedule from the current pointer, and updates the pointer to the next location of the accepted output.

The iSLIP differs from RRM in the grant step. In iSLIP, the pointer of an output will not be updated unless its grant message is accepted in the next step. By this modification, iSLIP largely eliminates the pointer synchronization problem in RRM, i.e. many outputs have their round robin pointers synchronized, thus all send the grant message to the same input in an iteration.

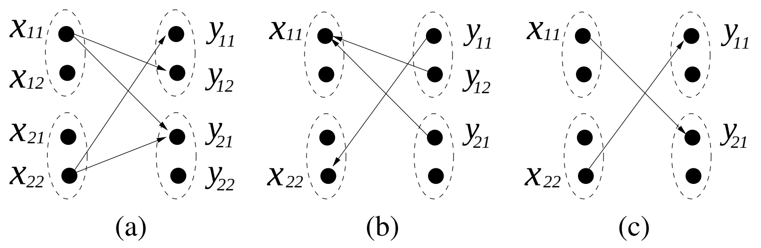
To extend the success of iSLIP to WDM packet switches, we propose the WDM-iSLIP algorithm. However, WDM-iSLIP is not a simple transplant of iSLIP to the WDM case. In WDM-iSLIP, each input wavelength channel maintains two round robin schedules. One is the output fiber (OF) schedule, consisting of N output fibers, and the other is the accessible wavelength (AW) schedule, consisting of its accessible wavelengths. Correspondingly, there are two pointers for each input wavelength, pointing to the current highest priority element in the OF schedule and the AW schedule, respectively. Each output wavelength channel also maintains two round robin schedules, the input fiber (IF) schedule and the AW schedule. Similarly, two pointers are defined for each output wavelength channel. Like iSLIP, WDM-iSLIP runs in i iterations, each of which consists of three steps. As the iteration goes on, the output of the algorithm improves towards a maximal matching between the input and output wavelength channels. Nevertheless, each step in one iteration of WDM-iSLIP differs from its counterpart in iSLIP. The details of a single iteration of WDM-iSLIP are described below, assuming the switch has M input fibers and N output fibers, with k wavelengths multiplexed on each fiber.

Request Step: Each unmatched input wavelength channel selects l output wavelength channels for which it has at least one queued packet, and sends a request to each of them. l is a parameter whose value can be adjusted to meet different requirements as will be discussed in more detail soon. The l wavelength channels are determined as follows. First, select the next l output fibers from the OF pointer in the OF schedule of the input wavelength. Then select the wavelength pointed by the AW pointer. If during the selection of the l fibers a new round begins (i.e. there is a jump from output fiber N to output fiber 1), then the wavelength next to the current AW pointer in the AW schedule should be selected for those output fibers belonging to the new round.

Grant Step: Each output wavelength channel, if receiving any requests, chooses the one from the input fiber that appears first in its IF schedule from its IF pointer. If there are more than one requests from that input fiber, choose the one from the wavelength that appears first in its AW schedule from its AW pointer. A grant message is sent to the corresponding input wavelength channel. If and only if the grant is accepted in the first iteration, the IF pointer and AW pointer will be updated. In that case, the IF pointer moves to one location beyond the accepted input fiber (modulo M) in the IF schedule, and the AW pointer of the output wavelength moves to the next accessible wavelength if and only if the IF pointer begins a new round in the IF schedule.

Accept Step: Each input wavelength channels, if receiving any grants, accepts the one from the output fiber that appears first in its OF schedule from the current OF pointer. If there are more than one grants from that output fiber, accept the one from the wavelength that appears first in its AW schedule from its AW pointer. The corresponding IF pointer and AW pointer are updated only in the first iteration. In that case, the OF pointer moves to one location beyond the accepted output fiber (modulo N). The AW pointer of the input wavelength moves to the next accessible wavelength if and only if the OF pointer begins a new round.

In Fig. 3.5, we give an example to explain how WDM-iSLIP works. In the figure, the switch has 2 input fibers and 2 output fibers, each containing 2 wavelengths that are mutually convertible. The value of l is 2. Let x_{ij} and y_{ij} denote wavelength channel j of fiber i at the input and the output, respectively. x_{11} and x_{22} have queued packets destined to both output fibers, and the rest of the input wavelength channels are currently idle. The round robin pointers for each input and output wavelength channel are shown in Fig 3.5(d) (those not shown have no impact on the result). In the request step, x_{11} first sends a request to y_{21}. The second request of x_{11} is sent to y_{12} instead of y_{11}, as there is a jump from output fiber 2 to output fiber 1. The requests of x_{22} are sent to y_{11} and y_{21} since there is no such jump. In the grant step, y_{11} and y_{12} both receive only one request hence they simply grant it. y_{21} receives two requests. Since its IF pointer equals 1, requests from input fiber 1 have higher priority, thus y_{21} grants the request from x_{11}. Finally, during the accept step, x_{22} accepts the only grant it receives, updates its OF pointer to 2, and keeps its AW pointer unchanged. x_{11}, who receives two grants, selects the one from y_{21} and updates its IF and AW



	x_{11}	x_{22}	y_{12}
OF	2	1	−
IF	−	−	1
AW	1	1	2

(d)

	x_{11}	x_{22}	y_{12}
OF	1	2	−
IF	−	−	2
AW	2	1	1

(e)

Figure 3.5: Example of one iteration of WDM-iSLIP. (a) Request step. Each input wavelength channel that is currently not idle sends 2 requests. (b) Grant step. Both y_{21} and y_{12} grant the request from x_{11} since it is of the highest priority among all received requests according to their IF and AW pointers. (c) Accept step. x_{11} accepts the grant from y_{21} because y_{21} has higher priority than y_{12} based on the OF pointer of x_{11}. Then the OF pointer of x_{11} is updated to point to output fiber 1, which triggers the update of the AW pointer of x_{11} to point to wavelength 2. (d) Round robin pointers before the current iteration. (e) Updated round robin pointers after the current iteration.

pointers to 1 and 2, respectively. The update of the AW pointer of x_{11} is triggered because its IF pointer jumps back to the beginning of the OF schedule.

WDM-iSLIP inherits some nice properties from iSLIP. For example, outputs are not likely to have their pointers synchronized and thus the grant messages can be distributed to different inputs. Also, because of the round robin fashion, it tends to have nice fairness property. Meanwhile, there are some significant differences between WDM-iSLIP and iSLIP, which can be summarized as "selective request" and "hierarchical round robin." These changes lead to the following advantages of WDM-iSLIP:

1. Reduced information exchange between the input and output in each iteration. In an $M \times N$ optical switch with k wavelength multiplexed on each fiber, if each input wavelength sends requests to every output wavelength it has buffered packets destined to, the total number

of requests generated in one step is $MN \sum_w |C_w|$, which can be as large as MNk^2 and is k^2 times of that in an electronic switch of the same size. Thus this may be too much for the information exchanged between the input and the output in one single step of an iteration. With WDM-iSLIP, the number of requests is reduced to Mkl, where the value of l can be adjusted to trade-off between the system performance and the amount of information exchanged between the switch input and output. For example, if the size of the switch is large, or the scheduler executes only a small number of iterations of WDM-iSLIP in each time slot, a larger l may be necessary to generate more requests per iteration, such that more packets can be scheduled. On the other hand, if the switch of interest is small, or many iterations of WDM-iSLIP are executed per time slot, then it is safe to keep l small.

2. Each input wavelength channel sends packets in a round robin manner to output fibers, instead of to the output wavelength channels. By separating the fiber pointer and the wavelength pointer, it becomes less likely that in successive time slots an input wavelength channel keeps transmitting packets to wavelengths on the same output fiber, or an output wavelength channel keeps being occupied by wavelengths on the same input fiber. As a result, the bandwidth (load) is distributed more fairly among different inputs (outputs).

The WDM-iSLIP algorithm can be implemented in a way similar to the iSLIP algorithm. For example, each output wavelength channel may receive a binary vector representing the requests sent by the input wavelength channels, and each input wavelength channel may receive a binary vector representing the grants sent by the output wavelength channels. There is an arbiter for each input/output wavelength channel which determines the highest priority element in the round-robin schedule based on the round-robin pointers. The only extra cost of WDM-iSLIP over classic iSLIP is that each arbiter needs to maintain a second pointer due to hierarchical round robin as explained above.

3.5 Performance Evaluations

We have evaluated the performance of the proposed buffering fabric and scheduling algorithms through simulations. To rule out the possibility that certain performance depends on any particular value of the switch size, the number of wavelength channels, or the ratio of the two, we have simulated two different switches. One has 8 input fibers and 8 output fibers, with 8 wavelengths multiplexed on each fiber, and the other has 32 input/output fibers, with 4 wavelengths on each fiber.

We have conducted simulations under three different traffic models: Bernoulli arrival, burst arrival with geometric distribution and burst arrival with Pareto distribution. The first two traffic patterns have been explained in details earlier in 2.6. When the length of states follows Pareto distribution, the traffic simulates self-similar traffic [51]. For each traffic model, we further consider two types of traffic patterns, uniform traffic and non-uniform traffic. Under uniform traffic, a packet arriving at an input has the same probability to be destined to any of the output fibers. Non-uniform traffic is implemented as hotspot traffic in our simulation, which means that packets on an input wavelength channel have a higher probability p_h to be destined to the corresponding "hotspot" output fiber than to the rest. Here we set the hotspot of wavelength channels of input fiber i to be output fiber i, and $p_h = 30\%$. The remaining 70% packets are uniformly distributed to other output fibers. The wavelength conversion patterns are randomly generated with parameter *conversion density*, denoted as d, in our simulation. d represents the probability of being convertible between an input/output wavelength pair. Apparently, the larger d is, on average the more output wavelengths an input wavelength can be converted to.

Next we present the simulation results. The main performance criteria considered here are average input buffering delay and packet loss probability (due to the limited length of FDL). The impact of some important parameters on system performance, including the FDL length at each input wavelength, L, the conversion density, d, the number of iterations in WDM-iSLIP, i, and the maximum number of requests an input wavelength channel can send in one iteration, l, will be examined.

3.5.1 Average Buffering Delay

The buffering delay of a packet is the interval from the arrival of the packet to the time it is being transmitted to the output of the switch. Since we are considering switches with input buffer only, the buffering delay of a packet is also its transmission delay, i.e. the interval from its arrival to its departure. To rule out the effect of packet loss caused by the limited FDL length on packet delay, we set $L = 10^5$ for simulations on buffering delay. In the next subsection we will examine packet loss by considering a much shorter FDL length.

Fig. 3.6 and Fig. 3.7 illustrate the average buffering delay of MPWFPF and WDM-iSLIP with different number of iterations under two traffic patterns in the two simulated switches, respectively. It can be seen that under all conditions, WDM-1SLIP has longer buffering delay, which is reasonable since it runs only one iteration towards a maximal matching and can hardly achieve very good performance. We also notice that in the first switch ($N = 8, k = 8$), WDM-2SLIP and WDM-4SLIP provide close performance. This indicates that for this switch the average number of convergence iterations, i.e., the number of iterations when the maximal matching is found, of WDM-iSLIP is around 2. On the other hand, in the second switch, WDM-2SLIP has much longer buffering delay than WDM-4SLIP when traffic load is over 0.8, which means that WDM-iSLIP needs more iterations to converge in this switch due to its larger size. At the same time, although MPWFPF performs close to WDM-2SLIP and WDM-4SLIP in terms of buffering delay when the offered load is light, as the load approaches 1, MPWFPF shows its advantage. This is consistent with the theoretical result that MPWFPF delivers 100% throughput while the maximal matching based WDM-iSLIP cannot.

We are also interested in how the wavelength conversion capability of the switch affects the buffering delay, which is given in Fig. 3.8. It can be seen that the system performance benefits from the increase in wavelength conversion capability. The larger the conversion density d is, the more wavelengths a wavelength can be converted to, hence the more flexibility of the scheduling. Meanwhile, as can be observed, for both MPWFPF and WDM-4SLIP, the decrease of average buffering delay becomes smaller and smaller with the increment of d. This conforms with the observations in the literature that, while wavelength conversion is important to resolve contention in WDM optical switches, it is not necessarily to be full-range conversion. Most of time limited-range or even simpler conversion pattern will suffice.

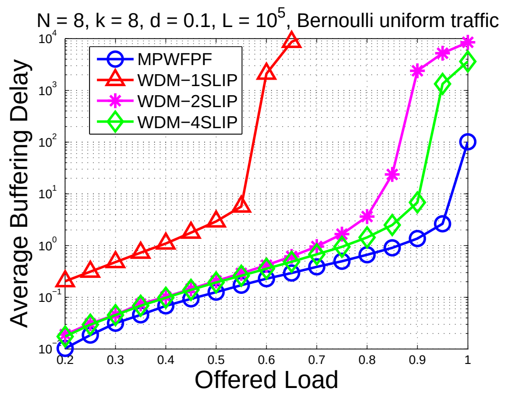
Under WDM-iSLIP, another parameter that may affect the average buffering delay is l, the maximum number of requests each input wavelength channel can send in an iteration. In our simulations, l is set to 4 by default. To see the effect of l, we now change its value to 2 and re-measure the average packet buffering delay under Bernoulli and burst traffic with geometrical distribution, keeping all other parameters unchanged as in Fig. 3.6. The results are plotted in Fig. 3.9. It can be seen that while under WDM-1SLIP, more traffic can be handled with $l = 4$ than that with $l = 2$, for WDM-2SLIP and WDM-4SLIP, the difference in average packet delay between the two cases is not obvious. This implies that when the iSLIP algorithm runs multiple iterations, it is possible to choose a small value of l to reduce information exchange between inputs and outputs of the switch, without introducing significant extra buffering delay.

3.5.2 Packet Loss Probability

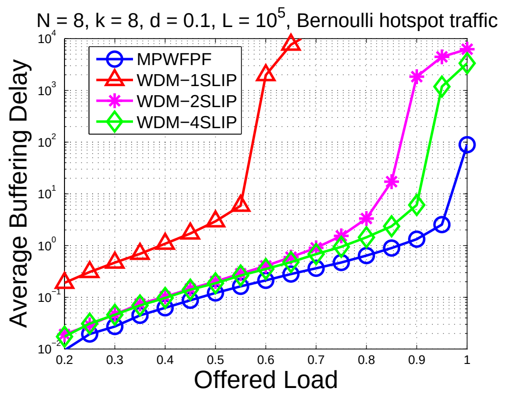
As mentioned earlier, the number of exits in the controllable FDL is limited in practice. Packet loss may occur when a packet arrives at the head of an FDL but still cannot be scheduled (note that there is no packet loss due to the arrival of new packets since the position at the tail of an FDL is always available at the beginning of a time slot). We have evaluated the packet loss probability for MPWFPF and WDM-4SLIP with FDL length L ranging from 0 to 100. The simulation results are shown in Fig. 3.10. It can be seen that for Bernoulli traffic, FDL with length less than 10 can effectively reduce the packet loss probability to below 10^{-3}. Burst traffic usually requires a larger buffer, but it is still within a practical range (around 10 for MPWFPF and around 80 for WDM-4SLIP).

3.6 Conclusions

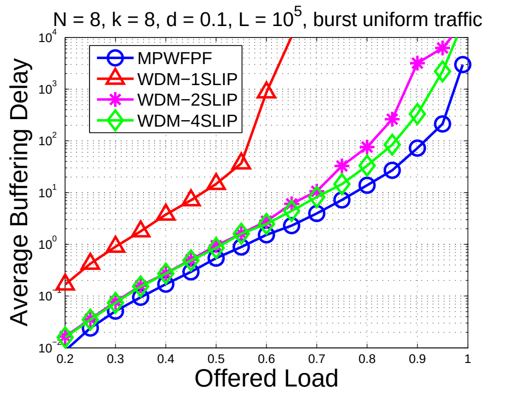
In this chapter we have studied optimal and fast packet scheduling in input-buffered WDM optical packet switches. We have proposed a controllable fiber delay line buffer that is able to provide flexible buffering time and employed it as the input buffer of the switch. We showed that the conditions for admissible traffic in WDM switches are different from that in electronic switches. By formalizing the scheduling problem into a network flow problem, we proved that a weight-based algorithm, in particular, the proposed Most-Packet Wavelength-Fiber Pair First algorithm, delivers 100% throughput under any admissible traffic with any wavelength conversion pattern for WDM packet switches. We have also presented a fast scheduling algorithm WDM-iSLIP which can efficiently find an approximate optimal schedule with lower time complexity. Simulations have been conducted to verify the theoretical analyses and test the performance of the proposed scheduling algorithms in input-buffered WDM packet switches.



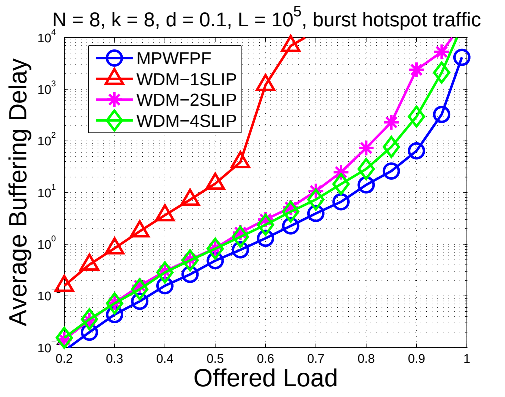
(a)



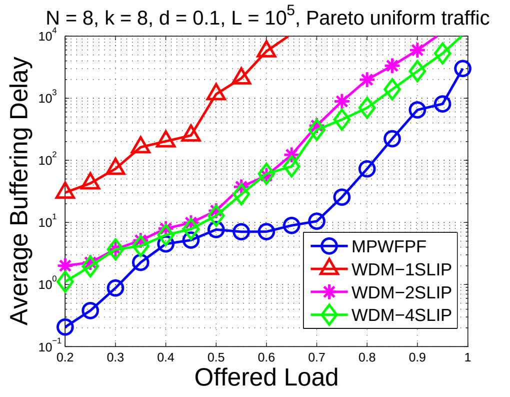
(b)



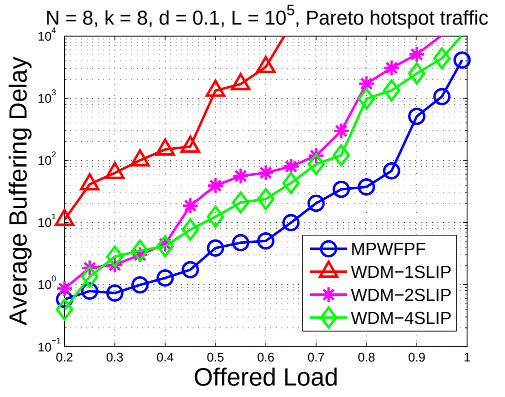
(c)



(d)

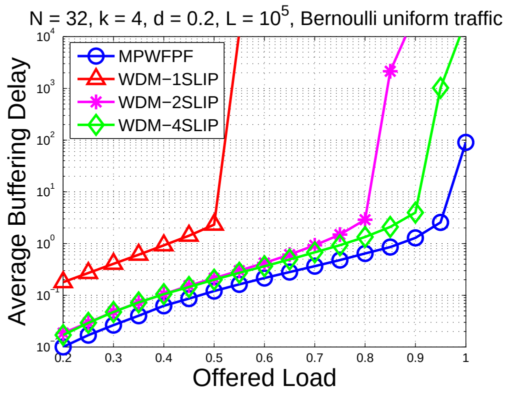


(e)

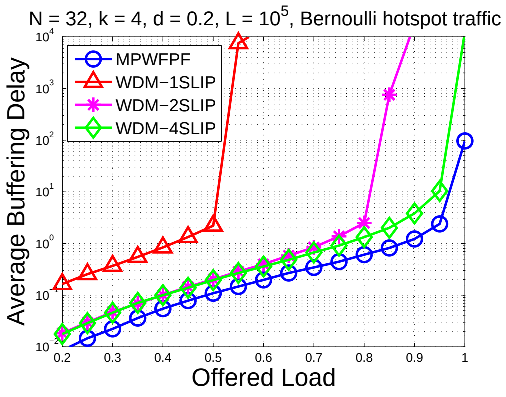


(f)

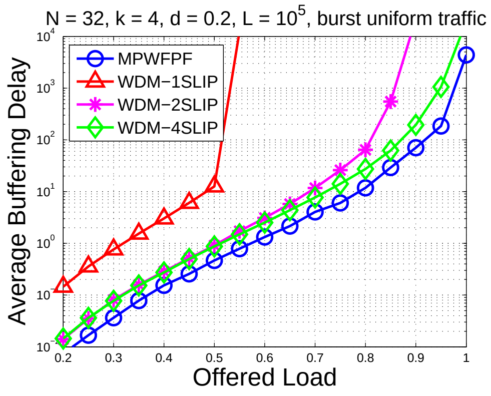
Figure 3.6: Buffering delay of MPWFPF and WDM-iSLIP with different number of iterations, i. The number of fibers $N = 8$, the number of wavelengths per fiber $k = 8$, conversion density $d = 0.1$ and FDL length $L = 10^5$. The delay (Y axis) is plotted in logarithmic scale.



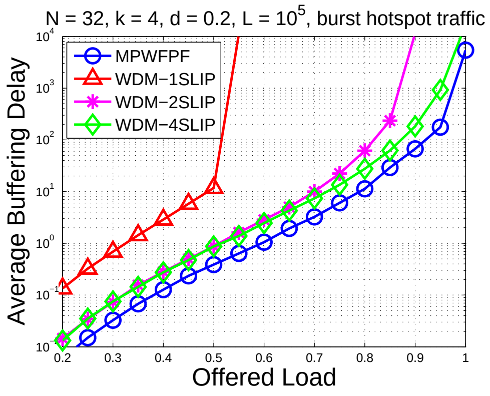
(a)



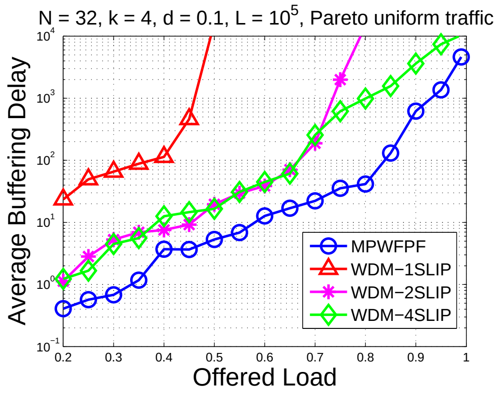
(b)



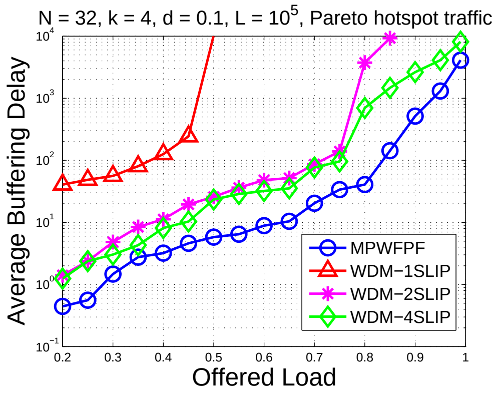
(c)



(d)

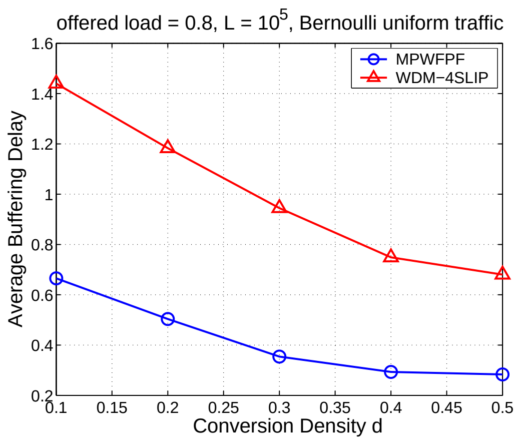


(e)

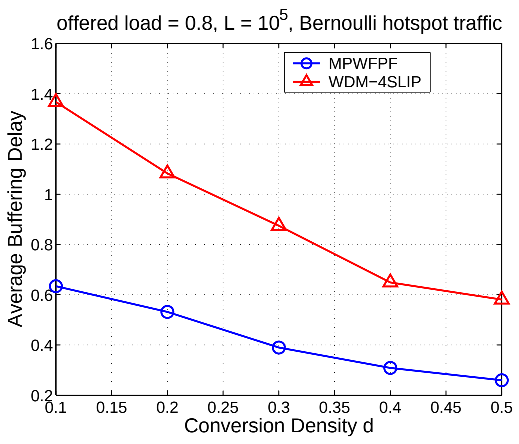


(f)

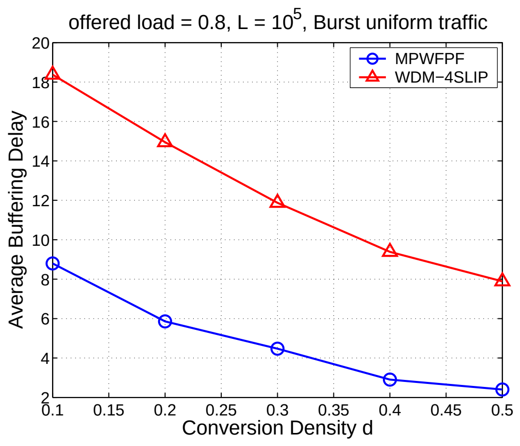
Figure 3.7: Buffering delay of MPWFPF and WDM-iSLIP with different number of iterations, i. The number of fibers $N = 32$, the number of wavelengths per fiber $k = 4$, conversion density $d = 0.2$ and FDL length $L = 10^5$.



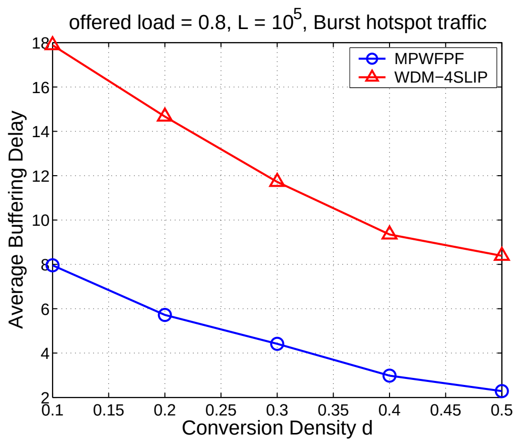
(a)



(b)



(c)



(d)

Figure 3.8: Buffering delay of MPWFPF and WDM-4SLIP under different wavelength conversion densities. $N = 8, k = 8$, offered load = 0.8, FDL length $L = 10^5$.

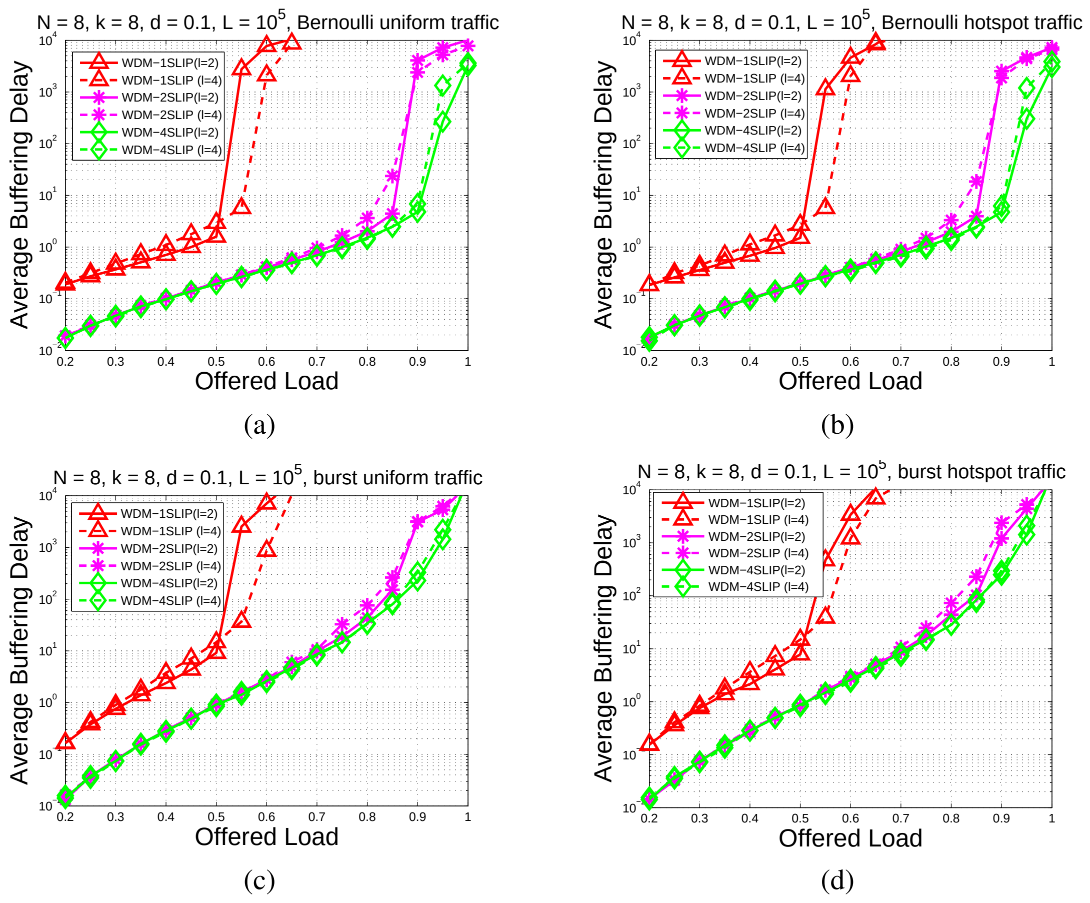
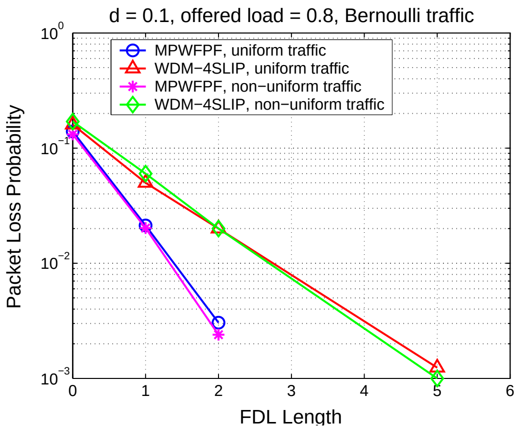
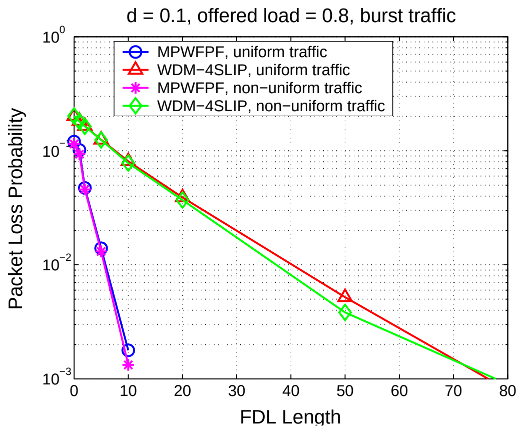


Figure 3.9: Buffering delay under WDM-iSLIP with different l, and $N = 8, k = 8, d = 0.1, L = 10^5$.



(a)



(b)

Figure 3.10: Packet loss probability of MPWFPF and WDM-4SLIP under different FDL lengths. $N = 8, k = 8$, offered load = 0.8 and wavelength conversion density $d = 0.1$.

Chapter 4

Packet Scheduling in the OpCut Switch - Single Wavelength Scenario

So far in this dissertation we have been focused on all-optical switches. While all-optical architectures are widely recognized as the ultimate solution to future ultra-high speed interconnect network due to electronic bottleneck and data transparency, currently they suffer from the lack of optical RAMs and are constrained by the physical limit of fiber delay lines. To overcome these challenges, optical interconnect architectures with electronic buffers have been proposed recently [6][9][10][12] . Out of these architectures, the *OpCut* switch [6], combining optical switching with recirculating electronic buffer, achieves low latency and minimizes optical-electronic-optical (O/E/O) conversions by allowing packets to cut-through the switch. Fig. 4.1(a) shows a high-level view of the switch. The key feature of the OpCut switch is that the arrived optical packets will be routed to the output directly, or "cut through" the switch, whenever possible. Only those that cannot cut through are sent to the receivers, converted to electronic signals and buffered, which can be sent to the output ports later by the optical transmitters. Also, the OpCut switch does not hold any packet at the input ports. It always sends packets to the switching fabric, because with

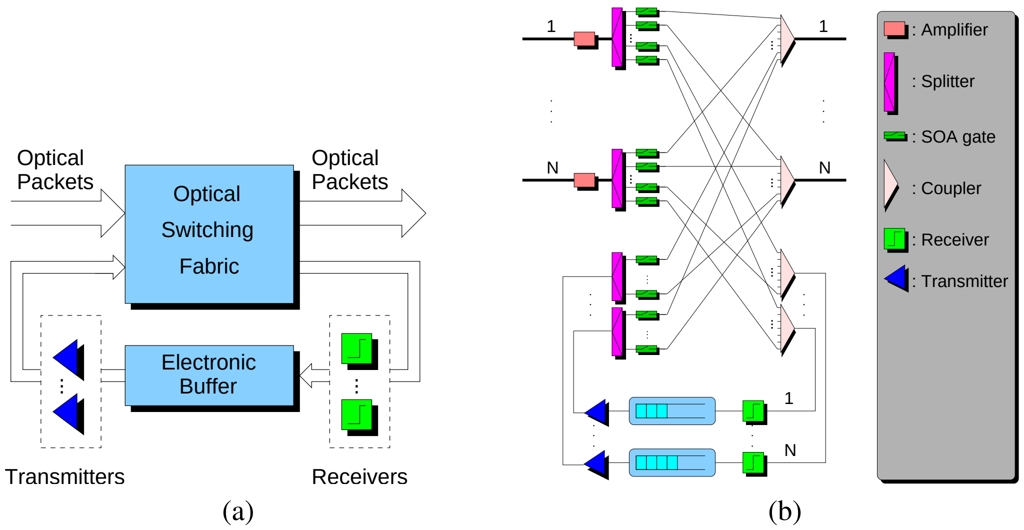


Figure 4.1: (a) A high level view of the OpCut switch. (b) A possible implementation of the OpCut switch.

the number of receivers equal to the number of input ports, there can always be found a receiver to "pick up" a packet that needs to be buffered. Hence, the OpCut architecture eliminates the round-trip delay between the input ports and the scheduler. Due to these reasons, the OpCut switch architecture holds the potential of achieving very low latency.

In this chapter, we study packet scheduling in the single-wavelength OpCut switch, aiming to achieve overall low packet latency while maintaining packet order. We first decompose the scheduling problem into three modules and present a basic scheduler with satisfactory performance. To relax the time constraint on computing a schedule and further improve system throughput, we propose a mechanism to pipeline packet scheduling in the OpCut switch by distributing the scheduling task to multiple "sub-schedulers." An adaptive pipelining scheme is also proposed to minimize the extra delay introduced by pipelining. Our simulation results show that the OpCut switch with the proposed scheduling algorithms achieve close performance to the ideal output-queued (OQ) switch in terms of packet latency, and that the pipelined mechanism is effective in reducing scheduler complexity and improving throughput.

The rest of this chapter is organized as follows. Section 4.1 contains a brief review of related work. Section 4.2 introduces the OpCut switch architecture. Section 4.3 presents the basic packet scheduler for the OpCut switch. Section 4.4 proposes a mechanism to pipeline the scheduling procedure, including an adaptive pipelining scheme. Section 4.5 presents the simulation results. Finally Section 4.6 concludes the chapter.

4.1 Related Work

Recently, IBM has built some prototypes of optical interconnects with tera bit switching capacity [8, 11]. The IBM PERCS project [12] adopts optical circuit switching on fixed routes to transfer long-lived bulk data packets. If data at a capacity of a fraction of a wavelength's granularity is to be carried, some fiber capacity will be wasted. To overcome this problem, the PERCS project uses electronic packet switching for short-lived data exchanges. Thus, in this sense, it may be considered as an intermediate feasible solution before more efficient optical packet switching becomes reality.

Similar to the OpCut switch, the prototype switch of the IBM OSMOSIS project [8, 11] also uses electronic buffer to overcome the buffering problem in optical packet switching. The OSMOSIS switch adopts an input-buffered architecture. At the input ports, all packets are converted from optical to electrical to be stored in electronic memory. The packets are then converted back to optical before being switched. As a result, all packets experience the Optical-Electronic-Optical (OEO) conversion delay. On the other hand, as the simulation results show, in the OpCut switch a large percentage of the packets cut-through the switch and do not experience such delay.

The idea of recirculating buffer can be traced back to the the Starlite switch [52] and the Sunshine switch [53]. Both of them are electronic switches based on banyan networks, and rely on the combination of a Batcher sorting network, a trap network and a concentrator to achieve internal nonblocking. Their drawback is that the size of the sorting network and the concentrator needs to be very large for the switch to achieve low packet loss when heavily loaded. In [54, 55], optical switches with "recirculating buffer" were proposed and analyzed. These switches also allow the packets that cannot be sent to the output port to be sent to a buffer inside the switch. However, the buffer is optical and its size is limited by the size of the switching fabric. More importantly, the problem of maintaining packet order was not addressed in [54, 55] and packets may be out of order after exiting the switch.

The two-stage switch in [56] bears some interesting similarities to the OpCut switch. In a two-stage switch with N inputs and N outputs, there are two $N \times N$ switches with buffers sandwiched in between. The two switches follow a fixed connection pattern to connect the inputs to the outputs: at time slot t, input i is connected to output $(i+t) \mod N$. Basically, the first stage switch spreads the input packets evenly on the buffers and the second stage switch sends the packets out in a round-robin manner. Like in the OpCut switch, maintaining packet order is challenging in the two-stage switch. To solve this problem, the two-stage switch needs additional queueing management such as the three-dimensional queue proposed in [57], or a sophisticated mechanism to feedback packet departure times to the inputs [58]. A potential problem with the two-stage switch is that it results in long packet delays because a packet may have to wait for N time slots before being sent out of the switch, even when the traffic load is very light. On the other hand, as will be discussed, the OpCut switch maintains packet order by simply scheduling head-of-flow packets only and achieves very low latency by allowing packets to cut-through the switch. Furthermore, the OpCut switch can adopt pipelined scheduling adaptively to achieve the best balance between performance and scheduler complexity.

4.2 The OpCut Switch

In this section we briefly describe the architecture of the OpCut switch. The OpCut architecture may adopt any switching fabric that provides non-blocking connections from the inputs to the outputs. One possible switching fabric is shown in Fig. 4.1(b). It has N input fibers and N output fibers, numbered from 1 to N, and there is one wavelength on each fiber. In the following, we interchangeably use input (output) fiber, input (output) port, and input (output). Each input fiber is sent to an amplifier. The amplified signal is broadcast to N output fibers under the control of SOA gates. In addition, each signal is also broadcast to N receivers. An output fiber receives the signal and routes it to the processor or the next stage switch. A receiver converts the optical packet into electronic forms and stores it in its buffer. There is one transmitter per receiver buffer. A transmitter can read the packets and broadcast the selected packet to the output fibers, also under the control of SOA gates, such that the packets in the buffer may be sent out.

In this chapter, we follow the same assumptions as other optical switch designs that the switch works in time slots, and packets of fixed length fit in exactly a time slot. The length of a time slot is about 50 ns, similar to that in the OSMOSIS switch [8, 11]. Before receiving the packets, the switch is informed about the destinations of the packets. This can be achieved, for example, by letting a group of processors share an electronic connection to send the packet headers to the switch. The headers are sent to the switch before the packet is sent to allow the switch to make routing decisions and to configure the connections. Note that the cost associated with this electronic connection is likely to be small, because the link is shared by multiple processors and the link speed can be much lower than the data link. At the beginning of each time slot, up to one packet may arrive in optics at each input port. We define a flow as the stream of packets from the same input port and destined for the same output port. Unlike other optical switches with electronic buffers in which every packet goes through the O/E/O conversions, in the OpCut switch packets are converted between optics and electronics only when necessary. Whenever possible, arrived optical packets are directly sent to their desired output port, or, cut-through the switch. A packet that cannot cut-through is picked up by one of the receivers and sent to the electronic buffer. Later when its destined output port is not busy, the packet can be fetched from the electronic buffer, converted back into optics, and scheduled to the switch output.

In each time slot, a receiver picks up at most one packet, and a transmitter sends out at most one packet. In other words, there is no speedup requirement. The cost of the OpCut switch is mainly determined by the number of transmitters, the number of receivers, and the size of the switching fabric. It can be seen that the OpCut switch needs N transmitters and N receivers. The switch fabric is $N \times 2N + N \times N$, where the $N \times 2N$ part connects the inputs to the outputs and receivers, while the $N \times N$ part connects the transmitters to the outputs. To connect more processors, multiple OpCut switches can be connected according to a certain topologies, similar to the Infiniband switch.

4.3 The Basic Packet Scheduler for the OpCut Switch

The key challenge to achieve low latency in a switch is the design of the packet scheduling algorithm. Due to its feed-back buffer structure, existing scheduling algorithms cannot be directly applied to the OpCut switch. Keeping packet order also becomes more challenging in the OpCut switch. For example, in input-queued switches, packets belonging to the same flow are stored in the same Virtual Output Queue (VOQ), thus packet order is preserved as long as each VOQ works as a FIFO. In the OpCut switch, however, packets from the same flow may be picked up by different receivers.

The scheduling algorithm for an OpCut switch should give answers to the following three questions:

- Question 1. For the newly arrived packets, whether they may go to the output port directly or they should be buffered.
- Question 2. For a packet that should be buffered, which receiver should be used to pick it up.
- Question 3. For the output ports that are not receiving the new packets, which of the buffered packets may be sent to the output port.

This section is organized around how the three questions can be answered. We start with the notations and the basics of the scheduler.

4.3.1 Notations and Basics of the Scheduler

In an OpCut switch, input i is denoted as I_i, output j is denoted as O_j, and receiver r is denoted as R_r. Flow ij is defined as the stream of packets arrived at I_i destined to O_j. We also refer to the time slot in which a packet arrives at the switch input as the *timestamp* of that packet. Among all packets of a flow currently at the switch input or in the buffer, the one with the oldest timestamp is referred to as the *head-of-flow* packet. Maintaining packet order means that a packet must be a head-of-flow packet at the instant it is being transmitted to the switch output.

The OpCut scheduler adopts round-robin scheduling when the scheduler has multiple choices and has to choose one. Similar to [5], the round-robin scheduler takes a binary vector as input, and maintains a pointer to make the decisions. Let $[r_1, r_2, \ldots, r_N]$ be the input binary vector and let g be the current round-robin pointer g where $1 \leq g \leq N$. The scheduler picks the *highest priority element* defined as the first '1' encountered when searching the elements in the vector from r_g in an ascending order of the indices, while wrapping around back to r_1 when reaching r_N. Incrementing the round-robin pointer g by one beyond x means that $g \leftarrow x + 1$ if $x < N$ and $g \leftarrow 1$ if $x = N$.

4.3.2 Queueing Management

For each output port, the scheduler of the OpCut switch keeps the information of the packets that are in the buffer and are destined to the output port in a "virtual input queue" (VIQ) style. Basically, for output O_j, the scheduler maintains N queues denoted as F_{ij} for $1 \leq i \leq N$. For each packet arrived at I_i destined for O_j and are currently being buffered, F_{ij} maintains its timestamp, as well as the index of the buffer the packet is in. Note that F_{ij} does not hold the actual packets.

Packets are stored at the receiver buffers. It would make the scheduling much easier if each receiver maintains a dedicated queue for each flow. However, this will result in N^3 queues over all receivers and will lead to much higher cost which is unlikely to be practical when N is large. Instead, no queue is maintained in any receiver buffer, and an auxiliary array is adopted in each buffer to facilitate the locating of a specific packet. The auxiliary array is indexed by (partial) timestamps to store the location of packets in the buffer. Since in each time slot a receiver picks up at most one packet, it is able to locate a packet in constant time given the timestamp of the packet and the auxiliary array. Note that some elements of the array may be empty but the packets can always be stored continuously in the buffer.

As an implementation detail, the auxiliary array can be used in a wrap-around fashion thus does not need to have very large capacity. For instance, if the index of the array is 8-bit long, then the array stores the location of up to 256 packets. Consequently, only the lower 8 bits of the timestamp is needed to locate a packet in the buffer. A conflict occurs only if a packet is routed to a receiver, and another packet picked up by the same receiver at least 256 time-slots earlier is still in the buffer. When this is the case, it usually indicates heavy congestion. Hence it is reasonable to discard one of the packets.

4.3.3 The Basic Scheduling Algorithm

Next we describe a basic scheduling algorithm for the OpCut switch. To maintain packet order, the basic algorithm adopts a simple strategy. Basically, it *allows a packet to be sent to an output only if this packet is a head-of-flow packet*. The basic algorithm consists of three parts, each for answering one of the three questions.

Part I – Answering Question 1

For Question 1, the basic scheduling algorithm consists of two steps.

- *Step 1: Request*. If a packet arrives at I_i destined to O_j, the scheduler checks F_{ij}. If it is empty, I_i sends a request to O_j; otherwise, I_i does not send any request.
- *Step 2: Grant*. If O_j receives any requests, it chooses one to grant in a round-robin manner. That is, it will receive a binary vector representing the requests sent by the inputs. It picks the highest priority element based on its round-robin pointer and grants the corresponding input. Then it increments its round-robin pointer by one beyond the granted input.

In each time slot, since there is at most one packet arriving at each input, an input needs to send a request to at most one output and will receive no more than one grant. Therefore, the input will send the packet (or let the packet cut-through) as long as it receives a grant. The entire cut-through operation can be done by a single iteration of any iterative matching algorithm.

Part II – Answering Question 2

For question 2, the basic scheduling algorithm simply connects the inputs to the receivers according to the following schedule:

- At time slot t, the packet from I_i will be sent to R_r where $r = [(i + t) \mod N] + 1$.

Note that instead of a fixed one-to-one connection, the inputs are connected to the receivers in a round-robin fashion for better load balancing. As an example, according to our simulation, in an 8×8 OpCut switch, when there is a fully-loaded input port, the maximum overall throughput is around 0.85 if the inputs are connected to the receivers in the above way, versus 0.70 with fixed connection.

Part III – Answering Question 3

For Question 3, the scheduler requires one decision making unit for each output and one decision making unit for each buffer. It then runs the well-known iSLIP algorithm [5] between the receivers and the outputs. Each iteration of the algorithm consists of three steps:

- *Step 1: Request.* Each unmatched output sends a request to every buffer that stores a head-of-flow packet destined to this output.
- *Step 2: Grant.* If a buffer receives any requests, it chooses one to grant in a round-robin manner. That is, it will receive a binary vector representing the requests sent by the outputs. It picks the highest priority element based on its round-robin pointer and grants the corresponding output. The pointer is incremented to one location beyond the granted, if and only if the grant is accepted in Step 3.
- *Step 3: Accept.* If an output receives any grants, it chooses one to accept in a round-robin manner. That is, it will receive a binary vector representing the grants sent by the buffers. It picks the highest priority element based on its round-robin pointer and accepts the grant from the corresponding buffer. Then it increments its round-robin pointer by one beyond the granted buffer.

At the end of the algorithm, the scheduler informs each buffer which packet to transmit. It does so by sending the portion of the packet's timestamp that is needed for the buffer to locate the packet. With that information, the targeted packet can be found in constant time and sent through the transmitter. The switch is configured accordingly to route the packets to their destined output port.

4.4 Pipelining Packet Scheduling

Our simulation results show that the basic scheduling algorithm introduced above can achieve satisfactory average packet delay. However, in a high speed or ultra high speed environment, as the length of a time slot shrinks with the increase in line card rate it may become difficult for the scheduler to compute a schedule in each single time slot. In such a case, we can pipeline the packet scheduling to relax the time constraint. Furthermore, by pipelining multiple low-complexity schedulers, we may achieve performance comparable to a scheduler with much higher complexity. In this section we present such a pipeline mechanism.

Pipelined scheduling in electronic switches has been studied in previous works. In [3], a pipelined version of the RRGS algorithm was reported, in which each input port is assigned a scheduler. The scheduler of an input port selects an idle output port, and passes the result to the next input port. The process goes on until all input ports have been visited. However, this approach introduces an extra delay equal to the switch size. In [59] the pipelined maximal-sized matching algorithm (PMM) was proposed, which employs multiple identical schedulers. Each of these schedulers independently works towards a schedule for a future time slot. As pointed out in [60], PMM is "more a parallelization than a pipeline" since there is no information exchange between schedulers. [60] further proposed to pipeline the iterations of iterative matching algorithms such as PIM and iSLIP by adopting multiple sub-schedulers, each of which taking care of one single iteration and passing the intermediate result to the next sub-scheduler in the pipeline. One problem with this approach is that it may generate grants for transmission to an empty VOQ since at any time a sub-scheduler has no idea about the progress at other sub-schedulers and may try to schedule a packet that has already been scheduled by other sub-schedulers. As a result, the service a VOQ receives may exceed its actual needs and is wasted.

4.4.1 Background and Basic Idea

With pipelining, the computing of a schedule is distributed to multiple sub-schedulers and the computing of multiple schedules can be overlapped. Thus, the computing of a single schedule can span more than one time slot and the time constraint can be relaxed. Another consideration here is related to fairness. By adopting the iSLIP algorithm in the third step (i.e., determining the matching between electronic buffers and switch outputs), the basic scheduling algorithm ensures that no connection between buffers and outputs is starved. However, there is no such guarantee at the flow level. In addition, as mentioned earlier, a packet that resides in the switch for too long may lead to packet dropping. To address this problem and achieve better fairness, it is generally a good idea to give certain priority to "older" packets during scheduling.

Combining the above two aspects, the basic idea of our pipelining mechanism can be described as follows. We label each flow based on the oldness of its head-of-flow packet. Among all flows destined to the same output, a flow whose head-of-flow packet has the oldest timestamp is called the oldest flow of that output. Note that there may be more than one oldest flow for an output. Similarly, the flows with the i_{th} oldest head-of-flow packets are called the i_{th} oldest flows. Instead of taking all flows into consideration, we consider only up to the k_{th} oldest flows for each output when scheduling packets from the electronic buffer to the switch output. This may sound a little surprising but later we will see that the system can achieve good performance even when k is as small as 2. Then the procedure of determining a schedule is decomposed into k steps, with step i handling the scheduling of the i_{th} oldest flows. By employing k sub-schedulers, the k steps can

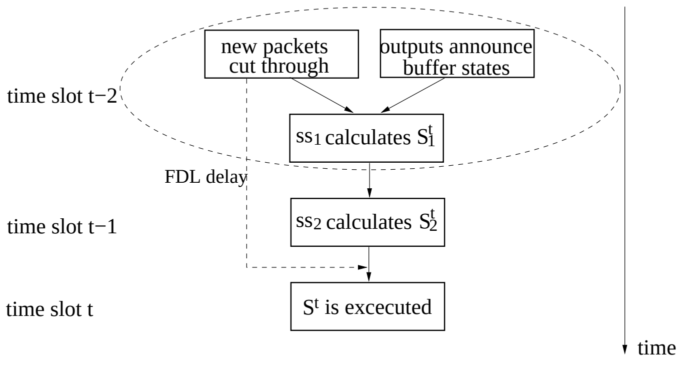


Figure 4.2: Timeline of calculating schedule S^t for time slot t.

be pipelined. Like the basic scheduling algorithm, the pipelined algorithm maintains packet order since only head-of-flow packets are qualified for being scheduled.

Next we will present the pipeline mechanism in more detail. Basically, like in prioritized-iSLIP [5], the flows are classified into different priorities. In our case the prioritization criterion is the oldness of a flow. By pipelining at the priority level, each sub-scheduler deals with only one priority level and does not have to be aware of the prioritization. Furthermore, since each sub-scheduler only works on a subset of all the scheduling requests, on average it converges faster than a single central scheduler. To explain how the mechanism works, we will start with the simple case of $k = 2$, that is, using only the oldest flows and second oldest flows when scheduling. We will also show that when $k = 2$, a common problem in pipelined scheduling, called duplicate scheduling, can be eliminated in our mechanism. Later we will extend the mechanism to allow an arbitrary k, and discuss potential challenges and solutions.

4.4.2 Case of $k = 2$

With $k = 2$, two sub-schedulers, denoted as ss_1 and ss_2 are needed to pipeline the packet scheduling. ss_1 tries to match buffers with the oldest flows to the output ports, while ss_2 deals with buffers with the second oldest flows. The timeline of calculating the schedule to be executed in time slot t, denoted as S^t, is shown in Fig. 4.2. The calculation takes two time slots to finish, from the beginning of time slot $t-2$ to the end of time slot $t-1$. When time slot t starts, S^t is ready and will be physically executed during this time slot. In time slot $t-2$, the cut-through operation for t is performed and the result is sent to the sub-schedulers, so that the sub-schedulers know in advance which output ports will not be occupied by cut-through packets at time t. To provide the delay necessary to realize pipelining, a fiber delay line with fixed delay of two time slots are appended to each input port. As a result, newly arrived packets are attempted for cutting-through at the beginning of time slot $t-2$, but they do not physically cut-through and take up corresponding output ports until time slot t. Later in Section 4.4.4 we will discuss how this extra delay introduced by pipelining may be minimized. As mentioned in Section 4.3.3, since the calculation of cutting-through is very simple and can be done by iSLIP with one iteration, or 1SLIP, there is no need to pipeline this step.

At the same time of cutting-through operation, each output port checks the buffered packets from all flows and finds its oldest and second oldest flows, as well as in which buffer these flows are stored. The outputs then announce to each buffer its state. The state of a buffer consists of two bits and has the following possible values: 0 if this buffer contains neither oldest nor second oldest flow for the output; 2 if the buffer contains one second oldest flow but no oldest flow; 1 otherwise. A buffer is said to contain an i_{th} flow of an output if it contains the head-of-flow packet of that flow. Note that the state being 1 actually includes two cases, i.e. the buffer has an oldest flow only, or has both an oldest and a second oldest flow. The point here is that we do not need to distinguish between these two cases. This is due to the fact that in a time slot at most one packet can be transmitted from a buffer to the switch output. Then if a buffer has an oldest flow for an output and a packet is scheduled from this buffer to the output, no more packets from other flows can be scheduled in the same time slot; on the other hand, if no packet from the oldest flow is scheduled to the output, no packet from the second oldest flows can be scheduled either since otherwise a packet from the oldest flow should have been scheduled instead. Thus as long as a buffer contains an oldest flow for an output, we do not need to know whether it contains a second oldest flow for that output or not.

Fig. 4.3 provides a simple example with $N = 3$ that shows how the announcing of oldest and second oldest flows works. In this example, we focus on one tagged output and three flows associated with it. As shown in the figure, packets p_1 and p_2 arrive in the same time slot but from different flows. p_3 arrives following p_2. A few time slots later, p_4 belonging to flow 3 arrives. We assume that some time later p_1, p_2 and p_4 become the head-of-flow packet for the three flows, respectively. It can be seen that flows 1 and 2 are the oldest flows, and flow 3 is the second oldest flow. As shown in the figure, assume that p_1 and p_2 are stored in buffers 1 and 2, respectively, and both p_3 and p_4 are in buffer 3. Then the tagged output will make the announcement as "1" to buffers 1 and 2, and "2" to buffer 3, which informs the sub-schedulers that buffers 1 and 2 have an oldest flow for this output , and buffer 3 has a second oldest flow but no oldest flow for this output.

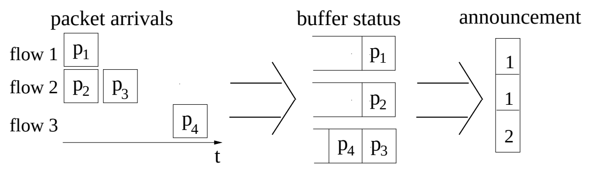


Figure 4.3: An example of how an output makes the announcement. The information of all packets that are in the buffer and destined for the output port is maintained for each output port. Based on that information, an output can find the oldest and second oldest flows, and where the head-of-flow packets are buffered. Then it can make the announcement accordingly.

After receiving the result of cutting-through operation, and the announcements from the outputs, sub-scheduler ss_1 is now set to work. Note that while the sub-schedulers work directly with buffers, they essentially work with flows, in particular, head-of-flow packets, since they are the only packets eligible for transmission for the sake of maintaining packet order. Denote the set of output ports that will not be occupied by cut-through packets at time slot t as O^t. What ss_1 does is to match the output ports in O^t to the buffers containing an oldest flow of these output ports. Theoretically, this process can be done by any bipartite matching algorithm. For simplicity, the iSLIP algorithm is adopted. In each iteration of the iSLIP algorithm, if there is more than one buffer requesting the same output port, ss_1 decides which of them the output should grant. Then, in case a buffer is granted by multiple output ports, ss_1 determines which grant the buffer should accept. The decisions are made based on the round-robin pointers maintained for each output port and buffer. The number of iterations to be executed depends on many factors, such as performance requirement, switch size, traffic intensity, etc. Nevertheless, as mentioned earlier, it can be expected that the result will converge faster than that of a single central scheduler since the sub-scheduler handles only a subset of all the scheduling requests.

ss_1 has one time slot to finish its job. At the beginning of time slot $t-1$, ss_1 sends its result to the output ports so that the output ports can update the VIQs and announce the latest buffer state. Meanwhile, ss_1 relays the result to ss_2. The functionality of ss_2 is exactly the same as ss_1, i.e. matching output ports to buffers according to some pre-chosen algorithm. The difference is that, ss_2 only works on output ports that are in O^t and are not matched by ss_1, and buffers that are announced with state 2 by at least one of these output ports. When ss_2 finishes the job at the end of time slot $t-1$, the matching based on which the switch will be configured in time slot t is ready. Meanwhile the packets that arrived at the beginning of time slot $t-2$ have gone through the two-time-slot-delay FDLs and reached the switch input. In time slot t, the buffers are notified which packet to send, and the switch is configured accordingly. Packets are then transmitted to the switch output, either directly from the switch input or from the electronic buffer.

The complete picture of the pipeline packet scheduling for $k = 2$ is shown in Fig. 4.4. As mentioned earlier, S^t is the schedule executed in time slot t. S_i^t denotes the part of S^t that is computed by sub-scheduler ss_i during time slot $t-i$.

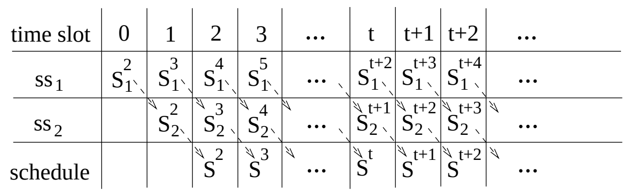


Figure 4.4: The pipelined scheduling procedure for $k = 2$.

A potential problem with pipelined scheduling algorithms is that it is possible for a packet to be included in multiple schedules, or, being scheduled for more than once. This is called duplicate scheduling. It could occur under two different conditions: 1) in the same time slot, different schedulers may try to include the same packet to their respective schedule, since a scheduler is not aware of the progress at other schedulers in the same time slot; 2) with pipelining, there is usually a delay between a packet being included in a schedule and the schedule being physically executed. During such interval the packet may be accessed by another scheduler that works on the schedule for a different time slot. In other words, a scheduler may try to schedule a packet that was already scheduled by another scheduler but has not been physically transmitted yet.

Duplicate scheduling of a packet leads to waste of bandwidth resources, which consequently causes underutilization of bandwidth and limits throughput. In an input-queued switch, when a packet p is granted for transmission more than once by different sub-schedulers, extra grants may be used to transmit the packets behind p in the same VOQ if the VOQ is backlogged. On the other hand, if the VOQ is empty, all but one grants are wasted. With the OpCut switch architecture, the consequence of duplicate scheduling is even more serious, in that extra grants for a packet cannot be used to transmit packets behind it in the same buffer. This is due to the fact that in an OpCut switch packets from the same flow may be distributed to different buffers, and a buffer may contain packets from different flows.

Duplicate scheduling is apparently undesirable but is usually difficult to avoid in pipelined algorithms. For example, the algorithms in [3] [59] [60] all suffer from this problem, even with only two-step pipelining. It was proposed in [60] to use pre-filter and post-filter functions to reduce duplicate scheduling. However, on one hand, these functions are quite complex, and on the other hand, the problem cannot be eliminated even with those functions. The difficulty roots in the nature of pipelining, that schedulers may have to work with dated information, and the progress at one scheduler is not transparent to other schedulers. Fortunately, as will be seen next, when $k = 2$ our mechanism manages to overcome this difficulty and completely eliminates duplicate scheduling.

First of all, it is worth noting that the "oldness" of a flow is solely determined by the timestamp of its head-of-flow packet. Thus we have the following simple but important lemma.

Lemma 3. *Unless its head-of-flow packet departs, a flow cannot become "younger."*

Next we deal with the first condition that may lead to duplicate scheduling. That is, we show that in any time slot the two sub-schedulers will not include the same packet in their respective schedule. In fact we have a even stronger result here, as shown by the following theorem:

Theorem 2. *During any time slot, sub-scheduler ss_1 and ss_2 will not consider the same flow when computing their schedule. In other words, if we denote F_i^t as the set of flows that ss_i takes into consideration in time slot t, then $F_1^t \cap F_2^t = \emptyset$ for any $t \geq 0$.*

Proof. First note that for $t = 0$ there is no second oldest flow, $F_2^t = \emptyset$, thus the theorem holds. Now assume for some $t > 0$, the theorem held up to time slot $t - 1$ but not in time slot t. In other words, there exists a flow f such that $f \in F_1^t$ and $f \in F_2^t$. Note that $f \in F_2^t$ indicates f was not an oldest flow at time $t - 1$. Thus at $t - 1$ there existed at least one flow that was older than f and destined to the same output as f. Denote such an flow as f', then $f' \in F_1^{t-1}$ since it was an oldest flow at that time. Besides, it can be derived that no packet from f' was scheduled by ss_1 in time slot $t - 1$. Otherwise, the corresponding output port should be matched, and at time t ss_2 would not consider any flow associated with that output, including f.

Furthermore, since $f' \in F_1^{t-1}$, it follows that $f' \notin F_2^{t-1}$, given that the theorem held in time slot $t-1$. Then neither ss_1 nor ss_2 could schedule any packet belonging to f' in time slot $t-1$. According to Lemma 3, f' is still older than f at time slot t. Consequently, f is not an oldest flow at t, and $f \in F_1^t$ cannot hold, which contradicts the assumption. This implies that the theorem must hold for time slot t if it held for time slot $t-1$. That proves the theorem for $t \geq 0$. $\square$

Next we consider condition 2. It is possible for condition 2 to occur between S_2^t and S_2^{t+1} due to the existence of a time glitch: the buffer states based on which S_2^{t+1} is calculated are announced at the beginning of time slot t. At that time S_2^t is not calculated yet. Thus it is possible that a packet is included in both S_2^t and S_2^{t+1}. In contrast, S_2^t and S_1^{t+1} can never overlap, since the latter is calculated based on the information announced after being updated with S_2^t. For the same reason, sub-schedules S_i^t and S_j^{t+x} would never include the same packet for any $t \geq 0, i, j \in \{1, 2\}$, as long as $x > 1$. Thus the task of eliminating condition 2 reduces to making sure that S_2^t and S_2^{t+1} do not overlap, which can be achieved as follows.

When an output makes its announcement, instead of three possible states as introduced earlier in this section, each buffer may be in a forth state denoted by value 3 (this is doable since the state of a buffer is 2-bit long), which means that this buffer contains a third oldest flow and no oldest or second oldest flow for this output. Furthermore, we call a flow a *solo flow* if it is the only i_{th} oldest flow, and a buffer a *solo buffer* for an output port if it contains a solo flow of that output port. Now suppose ss_2 matched an output port *op* to a buffer bf in S_2^t based on the announcements in time slot $t-2$. Then when S_2^{t+1} is being computed, bf is excluded from S_2^{t+1} if *op* again announced bf as a state-2 buffer. On one hand, if there exists at least one buffer other than bf that was announced with state 2 by *op* in time slot $t-1$, ss_2 will work with these buffers. On the other hand, if bf was a solo buffer for *op* based on the announcement at time slot $t-1$, ss_2 will work on state-3 buffers instead. Consequently, we have the following theorem.

Theorem 3. *The method introduced above ensures that S_2^t and S_2^{t+1} will not introduce duplicate scheduling of a packet.*

Proof. First, S_2^t and S_2^{t+1} may include the same packet only if ss_2 matches a buffer to the same output port in both S_2^t and S_2^{t+1}. Hence it is assumed that buffer bf is matched to output port *op* in both time slots $t-1$ and t by ss_2 (As a reminder, S_2^t is calculated in time slot $t-1$ based on output announcements made in time slot $t-2$). For this to occur, the state of bf announced at time slot $t-1$ can only be 3 according to the above method. Besides, bf cannot be a state-1 buffer of *op* for time slots $t-2$ and $t-1$, since otherwise bf should not be considered by ss_2. Then the states of bf announced by *op* at time slots $t-2$ and $t-1$, based on which S_2^t and S_2^{t+1} are calculated respectively, have only two possible combinations: 2 at time slot $t-2$ and 3 at time slot $t-1$ ($\{2, 3\}$), or 3 at time slot $t-2$ and 3 at time slot $t-1$ ($\{3, 3\}$). We will show that under neither of the combinations could duplicate scheduling occur.

- *$\{2, 3\}$:* In this case, by matching bf to *op*, S_2^t actually schedules to *op* the head-of-flow packet of some second oldest flow announced by *op* at time slot $t-2$. The head-of-flow packet is buffered in bf. Similarly, S_2^{t+1} schedules to *op* the head-of-flow packet of a third oldest flow announced at time slot $t-1$. Denote the two head-of-flow packets as p_a and p_b, and the two flows as f_a and f_b. On one hand, if flow f_a and flow f_b are different, packet p_a and packet p_b must be different. On the other hand, if flow f_a and flow f_b are the same flow, packet p_a and packet p_b are still different according to Lemma 3, since the flow becomes "younger" (second oldest at time slot $t-2$ and third oldest at time slot $t-1$).

- *$\{3, 3\}$:* Given that the state of bf is announced as 3 at time slot $t-1$ but ss_2 takes it into consideration when computing S_2^{t+1}, it must be true that in S_2^t ss_2 grants a buffer with a second oldest flow of *op* announced at time slot $t-2$ and that buffer is a solo buffer of *op*, which cannot be bf whose state announced at time slot $t-2$ is 3. This contradicts with the assumption that bf is matched to output port op in both time slots by ss_2.

Combining the two cases, the theorem is proved. $\square$

By now, duplicate scheduling is completely ruled out in our mechanism.

4.4.3 Case of $k > 2$

We now extend our result for $k = 2$ to the case that k is an arbitrary integer between 3 and N. The system performance can be improved at the cost of extra subschedulers. While the basic idea remains the same as $k = 2$, there are a few implementation details that need to be addressed when k becomes large. Duplicate scheduling can no longer be eliminated with an arbitrary k due to the increased scheduling complexity. Nevertheless we will propose several approaches to reducing it.

The basic pipelined scheduling procedure is given in Fig. 4.5. An FDL of length k is attached to each input port to provide the necessary delay for computing the schedules. k identical sub-schedulers, ss_1, ss_2, ..., ss_k are employed, ss_i dealing with buffers that contain an i_{th} oldest flow of some output port. Intermediate results are passed between adjacent sub-schedulers and used to update the VIQ status. The computing of the schedule to be executed in time slot t spans k time slots, from the beginning of time slot $t-k$ to the end of time slot $t-1$. The announcement of buffer states from an output port to the sub-schedulers can be done exactly the same way as that for $k = 2$, except that the state of a buffer for an output is now of length $\log(k+1)$ bits.

We have addressed the solo buffer problem for $k = 2$ to eliminate duplicate scheduling. Namely, if sub-scheduler ss_2 matched a buffer bf to an output port op in S_2^t, it will not consider bf as a state-2 buffer for op when computing S_2^{t+1} even if it was announced so. In case bf is the solo buffer of op, i.e. the buffer announced by op to contain the only second oldest flow of it, ss_2 will work on state-3 buffers for op trying to keep work conserving. For an arbitrary k, the rule is still

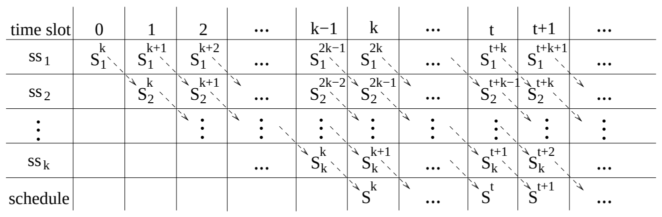


Figure 4.5: Pipelined scheduling procedure for an arbitrary k.

kept, that if ss_i matched a buffer bf to an output port op in S_i^t, it will not consider bf as a state-i buffer for op when computing S_i^{t+1}. However, if bf is the solo buffer of op, ss_i will *not* turn to buffers with state $i+1$. The reason is that, while this method involves only ss_2 when $k = 2$, it may cause a chain effect when $k > 2$: if ss_i sets to work on buffers with state $i+1$ at some time, then ss_{i+1} needs to work on buffers with state $i+2$ for the same schedule. In case there is only a solo buffer with state $i+1$ and is matched by ss_i again, then in the next time slot, ss_i may have to work on buffers with state $i+2$ and ss_{i+1} has to work on buffers with state $i+3$. The process could go on and become too complicated to implement. Therefore, if an output announced the same buffer as the solo buffer in two consecutive time slots, say, $t-1$ and t, and ss_i matched this buffer to the output in S_i^{t+k-i}, it will not try to match the output to any buffer in $S_i^{t+k+1-i}$. In other words, we will let ss_i be *idle* for the output in time slot $t+i$ in that case.

By allowing a sub-scheduler to be idle for some output port in certain time slot, we prevent the possibility that the sub-scheduler schedules a packet that was already scheduled and blocks other sub-schedulers behind it in the pipeline from scheduling a packet to that output port. Unfortunately, the cost is that Theorem 2 does not hold for $k > 2$. To see this, first note that F_i^t is essentially the set of the i_{th} oldest flows of every output port at the beginning of time slot $t+1-i$. For instance, F_1^t is the set of the oldest flows at time slot t and F_3^t is the set of the third oldest flows at time slot $t-2$. If there is a flow f such that $f \in F_i^t$, then it is one of the i_{th} oldest flows for some output port at time slot $t+1-i$. During the time interval, denoted as T, from time slot $t+1-i$ to time slot $t - j$ for some $j < i$, at most $i - j$ flows for that output can be scheduled. Therefore, at time slot $t + 1 - j$, f is at least the $i - (i - j) = j_{th}$ oldest flow. If f is indeed the j_{th} oldest flow, which can occur if and only if $i - j$ flows that are "younger" than f have been scheduled during T and all of them are solo flows, $f \in F_j^t$ holds. In that case, $f \in F_i^t \cap F_j^t$ holds, and ss_i and ss_j may schedule the same packet during time slot t. Nevertheless, as can be seen, the possibility that F_i^t overlaps with F_j^t is rather small and should not significantly affect the overall system performance. In fact, if we let P_r denote the probability that an output port op announces a buffer bf as the buffer which contains the solo second oldest flow and bf is later matched to op by ss_2 based on the announcement, then according to our simulations for $k = 4$, when the traffic intensity is as high as 0.9, P_r is less than 2%. The probability for the case of multiple solo flows is roughly exponential to P_r and thus is even smaller.

4.4.4 Adaptive Pipelining

We have discussed the mechanism to pipeline packet scheduling in the OpCut switch for any fixed k. In the following we will enhance it by adding adaptivity. The motivation is that, in our mechanism, the extra delay introduced by pipelining is equal to the number of active sub-schedulers, or k. When traffic is light, a small number of sub-schedulers may be sufficient to achieve satisfactory performance, or pipeline is not necessary at all. In this case, it is desirable to keep k as small as possible to minimize the extra delay . On the other hand, when the traffic becomes heavy, more sub-schedulers are activated. Although the delay of pipelining increases, now more packets can be scheduled to the switch output since more packets are taken into consideration for scheduling due to the additional sub-schedulers.

The first step towards making the pipelined mechanism adaptive is to introduce flexibility to the FDLs attached to the switch output ports. Since k sub-schedulers working in pipeline require

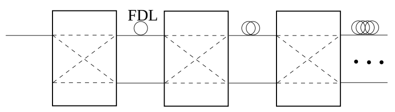


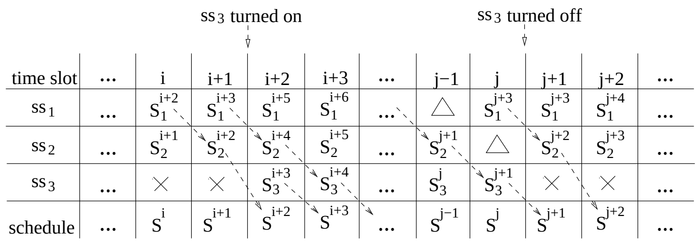
Figure 4.6: A possible implementation of an FDL that can provide flexible delays to fit the needs of pipeline with different number of sub-schedulers. There are $\lfloor \log K \rfloor + 1$ stages. The i_{th} stage is able to provide either zero delay or 2^i time slot delay.

a k time slot delay of the newly arrived packets, the FDL needs to be able to provide integral delays between 0 and K time slots, where K is the maximum number of sub-schedulers that can be activated. Clearly, $K \leq N$.

A possible implementation of such an FDL is shown in Fig. 4.6. The implementation adopts the logarithmic FDL structure [61] and consists of $\lfloor \log K \rfloor + 1$ stages. A packet encounters no delay or 2^i time slot delay in stage i, depending on the input port it arrives at the switch of stage i and the state of the switch. Through different configurations of the switches, any integral delay between 0 and K can be provided.

The number of packet arrivals in each time slot is recorded, and the average over recent W time slots is calculated and serves as the estimator of current traffic intensity. This average value can be efficiently calculated in a sliding window fashion: let A_i denote the number of packet arrivals in time slot i, then at the end of time slot t, A is updated according to $A = A - (A_{t-w+1} - A_t)/W$. An arbitrator decides whether a sub-scheduler needs to be turned on or off based on A. If during certain consecutive time slots, A remains larger than a preset threshold for the current value of k, an additional sub-scheduler will be put into use. Similarly, if A drops below some threshold and does not bounce back in certain time interval, an active sub-scheduler can be turned off.

The value of W can be adjusted to trade-off between sensitivity and reliability: if W is large, the averaging of traffic intensity is over a relatively long time period, and it is less likely that a small jitter will trigger the activation of an additional sub-scheduler. However, more time is needed for



time slot	...	i	i+1	i+2	i+3	...	j−1	j	j+1	j+2	...
ss_1	...	S_1^{i+2}	S_1^{i+3}	S_1^{i+5}	S_1^{i+6}	...	△	S_1^{j+3}	S_1^{j+3}	S_1^{j+4}	...
ss_2	...	S_2^{i+1}	S_2^{i+2}	S_2^{i+4}	S_2^{i+5}	...	S_2^{j+1}	△	S_2^{j+2}	S_2^{j+3}	...
ss_3	...	×	×	S_3^{i+3}	S_3^{i+4}	...	S_3^{j}	S_3^{j+1}	×	×	...
schedule	...	S^{i}	S^{i+1}	S^{i+2}	S^{i+3}	...	S^{j-1}	S^{j}	S^{j+1}	S^{j+2}	...

Figure 4.7: An example of sub-schedulers being turned on and off.

it to detect a substantial increase in traffic intensity, and vice versa.

An example of adaptive pipelining is given in Fig. 4.7. The basic idea is the same for any k value, thus we only show the process from two sub-schedulers to three sub-schedulers and then back to two to keep it neat. The "$\times$" state in the figure indicates the sub-scheduler is off, and a "Δ" means the sub-scheduler is on but will be idle in the time slot. The arrows in the figure illustrate how the intermediate results are relayed among sub-schedulers at transition points when a sub-scheduler is being turned on or off.

4.5 Performance Evaluation

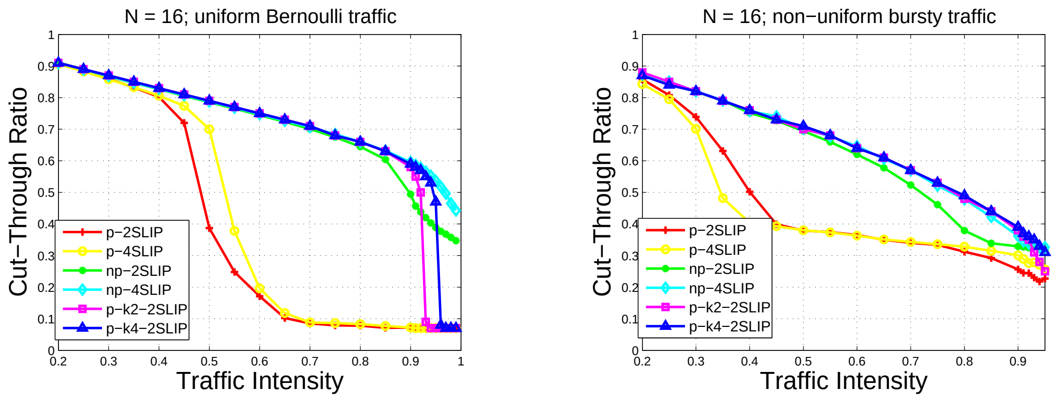
In this section, we evaluate the performance of the switch under two both the uniform Bernoulli traffic and the non-uniform bursty traffic. Both models assume that the arrival at an input port is independent of other input ports. See 2.6 for a more detailed description of the two models. For non-uniform traffic, it is assumed that a packet arrived at I_i is destined to O_i with probability $\mu + (1-\mu)/N$, and to O_j with probability $(1-\mu)/N$ for $j \neq i$, where μ is the "unbalance factor" and is set to be 0.5 which is the value that results in the worst performance according to [62]. We have evaluated OpCut switches of different sizes with both non-pipelined and pipelined schedulers. Each simulation was run for 10^6 time slots.

We implemented two instances of the proposed pipelining mechanism, denoted as p-k2-2SLIP and p-k4-2SLIP, respectively. Both of them are built on sub-schedulers executing two steps of iSLIP in each time slot. p-k2-2SLIP runs two such sub-schedulers and covers up to the second oldest flows of each input port, while p-k4-2SLIP runs four sub-schedulers and covers up to the fourth oldest flows. For comparison purpose, we implemented the basic non-pipelined scheduler running iSLIP as well, denoted as np-iSLIP. Also included in the simulations are the straightforwardly pipelined iSLIP scheduler (denoted as p-iSLIP) - i sub-schedulers, each of which executes one iteration of iSLIP in a time slot. Unlike the proposed pipelined schedulers, the straightforward approach is not aware of the duplicate scheduling problem.

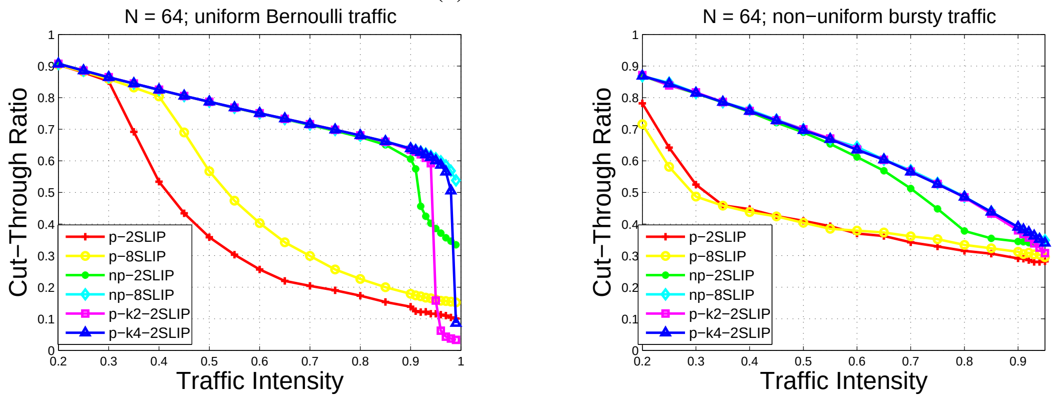
4.5.1 Cut-Through Ratio

First we investigate the packet cut-through ratio, which indicates how much portion of packets can cut-through the switch without experiencing electronic buffering. Apparently, if only a tiny portion of packets could cut-through, or packets could cut-through only when the traffic intensity is light, the OpCut switch would not be very promising. From Fig. 4.8, we can see that when the load is light, the cut-through ratio is high with all schedulers under both traffic models and switch sizes. However, For p-iSLIP schedulers, the ratio drops sharply with the increment in traffic intensity. For all the other simulated schedulers, the ratio decreases much slower, and stays above 60% under Bernoulli uniform traffic and 30% under bursty non-uniform traffic even when the load rises to 0.9.

We notice that under Bernoulli uniform traffic, there is a sharp drop in the cut-through ratio for both pipelined schedulers. For the 16×16 switch, the drop occurs at 0.93 load for p-k2-2SLIP and 0.95 load for p-k4-2SLIP. For the 64×64 switch it occurs at slightly higher loads. As will be confirmed shortly by the average packet delay, these are the points at which the OpCut switch is saturated with the respective pipelined scheduler. However, it is worth pointing out that higher

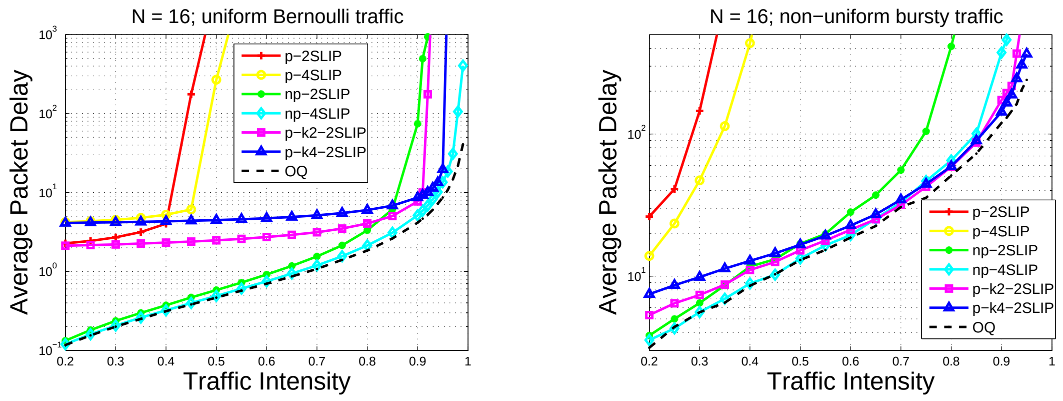


(a) 16×16 switch

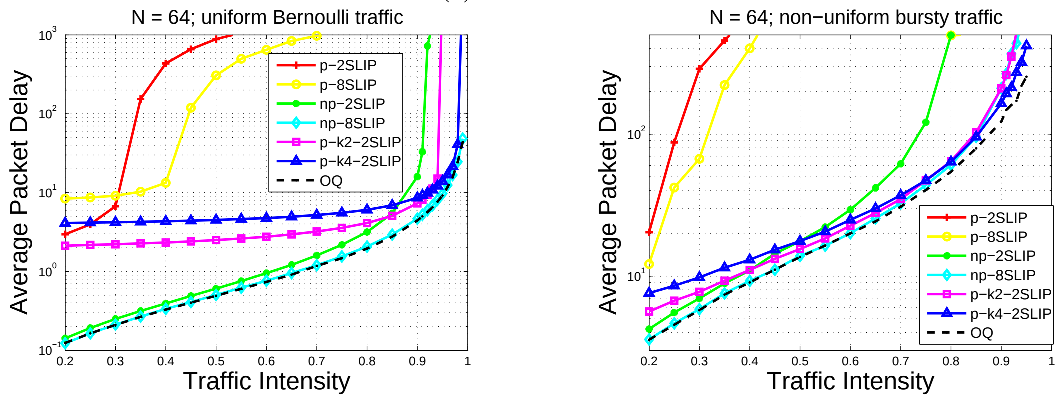


(b) 64×64 switch

Figure 4.8: Packet cut-through ratio with non-pipelined and pipelined schedulers under different traffic models and switch sizes. p-iSLIP: pipelined iSLIP with i sub-schedulers, each executing 1SLIP. np-iSLIP: non-pipelined scheduler executing iSLIP in each time slot. p-ki-2SLIP: pipelined scheduling that takes up to the i_{th} oldest flows into consideration, each sub-scheduler executing *2*SLIP.



(a) 16 × 16 switch



(b) 64 × 64 switch

Figure 4.9: Packet delay with non-pipelined and pipelined schedulers under different traffic models and switch sizes. The notations of schedulers are the same as in Fig. 4.8. OQ: ideal output-queued switch.

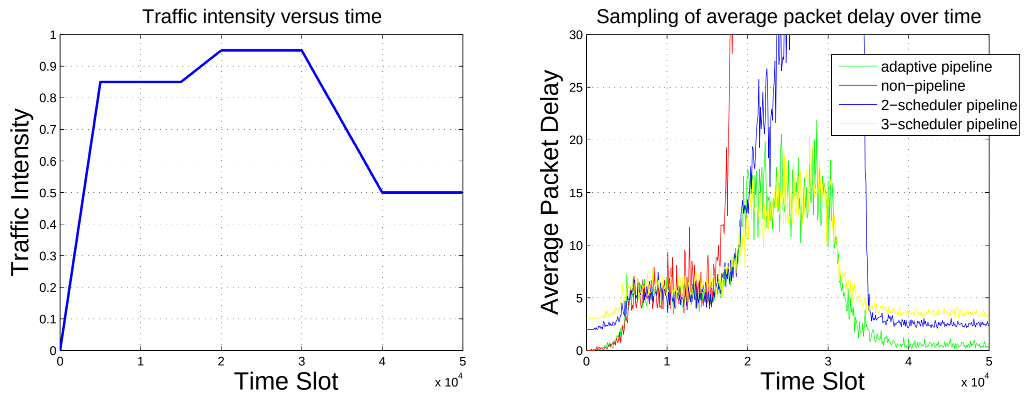
cut-through ratio does not necessarily imply better overall performance. In particular, throughput is not directly related to cut-through ratio, as packets can always be transmitted from the buffers to the switch output. Thus while non-pipelined scheduler np-2SLIP results in a higher cut-through ratio than p-k2-2SLIP and p-k4-2SLIP under higher-than-0.93 uniform Bernoulli traffic, it can be seen from Fig 4.9 that the pipelined schedulers actually achieve better delay and higher throughput than np-2SLIP under that traffic model.

An interesting observation is that for bursty non-uniform traffic, such sharp drops in cut-through ratio do not exist. This is likely due to the nature of non-uniform traffic that, except for those hotspot flows, most flows contains much fewer packets. For those packets, cutting-through becomes easier compared to the uniform-traffic scenario, since whether a packet can cut-through is independent of packets from other flows.

4.5.2 Average Packet Delay

Fig. 4.9 shows the average packet delay of the OpCut switch under different schedulers, traffic models and switch sizes. The ideal output-queued (OQ) switch is implemented to provide the lower bound on average packet delay. It can be seen that the straightforward pipelining approach, p-iSLIP, performs very poor due to underutilization of bandwidth caused by the duplicate scheduling problem. On the other hand, as instances of the proposed pipelining mechanism, p-k2-2SLIP and p-k4-2SLIP lead to substantially improved performance. The maximum throughput p-k2-2SLIP can sustain is about 0.93 under uniform Bernoulli traffic and 0.9 under non-uniform bursty traffic, which outperforms np-2SLIP by about 5% and 15%, respectively. In other words, using the same 2-SLIP scheduler, the system throughput can be improved through pipelining.

In fact, except for under light uniform Bernoulli traffic where the extra delay introduced by pipelining is comparatively significant (for which we have proposed the adaptive pipelining scheme), the performance of p-k4-2SLIP is very close to that of np-4SLIP when $N = 8$ and np-8SLIP when $N = 64$, which is in turn very close to that of the OQ switch in terms of average packet delay. That is, the performance of a non-pipelined scheduler that executes 8 iterations of iSLIP in each time slot can be well emulated by four schedulers working in pipeline, each of which executes 2-iteration iSLIP only. The time constraint on computing a schedule is relaxed by four times and the system performance is hardly affected, which illustrates the effectiveness of the proposed pipelining mechanism in reducing scheduler complexity.



(a) (b)

Figure 4.10: An example of adaptive pipeline. (a) The traffic model under which the traffic intensity changes with time. (b) Average packet delay over time under different pipelining strategies.

4.5.3 Adaptive Pipelining

Next we examine the effectiveness of adaptive pipelining. To illustrate the point, we consider a simple synthetic traffic model given in Fig. 4.10(a). The performance of adaptive pipelining is obtained and compared with that of non-pipelining and pipelining with a fixed number of sub-schedulers. All schedulers, pipelined or not, are assumed to run 1SLIP. The average packet delay is sampled every 100 time slots and is plotted against time in Fig. 4.10(b). It can be seen that in the first 2×10^4 time slots, while non-pipelining and 2-subscheduler pipelining eventually lead to very large delay, 3-subscheduler pipelining and adaptive pipelining successfully sustain the increase in traffic intensity. Moreover, when traffic intensity drops back to around 0.5, adaptive pipelining outperforms 3-subscheduler pipelining as it does not have a fixed 3 time-slot pipelining delay. In fact, according to Fig. 4.10(b), adaptive pipelining always has the minimum delay among all of these schedulers regardless of the change in traffic load. It achieves both high throughput under heavy traffic and low pipelining delay under light traffic by adjusting the number of sub-schedulers according to the traffic load.

4.6 Conclusions

In this chapter, we have considered pipelining the packet scheduling in the *OpCut* switch. The key feature of the OpCut switch is that it allows packets to cut-through the switch whenever possible, such that packets experience minimum delay. Packets that cannot cut-through are received by receivers and stored in the electronic buffer, and can be sent to the output ports by the transmitters. We have presented a basic packet scheduler for the OpCut switch that is simple to implement and achieves satisfactory performance. We then proposed a mechanism to pipeline packet scheduling in the OpCut switch by employing k sub-schedulers. The i_{th} sub-scheduler handles the scheduling of the i_{th} oldest flows of the output ports. We have respectively discussed the implementation details for $k = 2$ and an arbitrary k. For the case of $k = 2$, we have shown that our mechanism eliminates the duplicate scheduling problem. With arbitrary k, duplicate scheduling can no longer be eliminated, but we have proposed approaches to reducing it. We have further proposed an adaptive pipelining scheme to minimize the extra delay introduced by pipelining. Our simulation results show that the OpCut switch with the proposed scheduling algorithms achieve close performance to the ideal output-queued (OQ) switch in terms of packet latency, and that the pipelined mechanism is effective in reducing scheduler complexity and further improving switch throughput.

Chapter 5

Packet Scheduling in the OpCut Switch - WDM Scenario

Previous chapter studied packet scheduling in single-wavelength OpCut switches, including the basic scheduling algorithm and a pipelined scheduling mechanism. In this chapter, 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 the 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.

The rest of the chapter is organized as follows. Section 5.1 introduces the WDM OpCut switch, as well as notations and queuing management in this switch. Section 5.2 describes the goal of an optimal schedule and the basic scheduling procedure. Section 5.3 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 5.4 presents simulation results. Section 5.5 concludes the chapter.

5.1 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.

5.1.1 Switch Architecture

Like many other proposed switches in the literature [8, 11], the OpCut switch works in time slots. Packets are of fixed length and fit in exactly one time slot, which is approximately 50 ns and similar to that in the OSMOSIS switch [8, 11]. 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. 5.1. 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

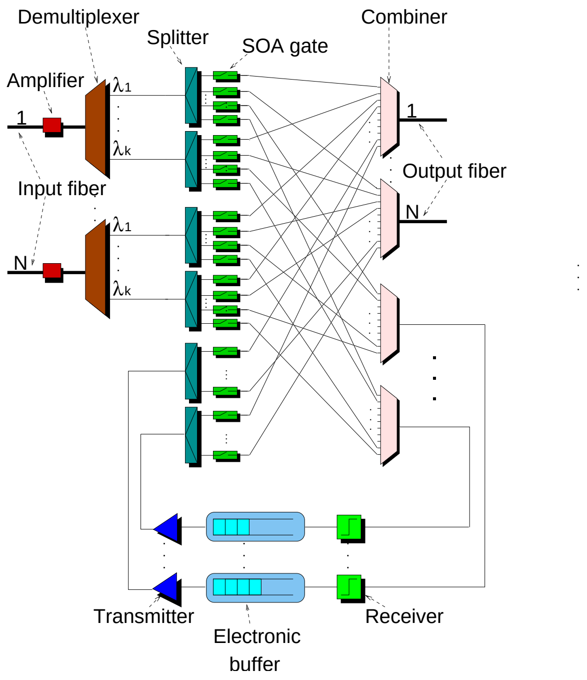


Figure 5.1: A possible implementation of the WDM OpCut switch.

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 and packet queuing at the switch output. A packet arriving at an output fiber will be picked up by its destined processor connected to the output fiber, or be routed to the next stage switch.

5.1.2 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 packet's *timestamp*. 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).

5.1.3 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 [63], 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 follows.

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. 5.2 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).

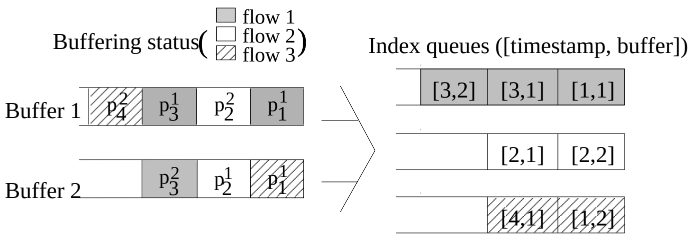


Figure 5.2: The relationship between the actual buffer status and the index queues, assuming packet p_i^λ arrived at the switch on wavelength λ in time slot i. The index queues keep the timestamps and buffer indices of the packets being buffered.

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.

5.2 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 chapter, we define an optimal schedule to be a schedule that satisfies the 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.

While the basic idea is similar, the basic scheduling procedure in a WDM OpCut switch differs significantly from the single-wavelength case due to dependence between wavelengths. The first stage 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 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

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

Similar to the single-wavelength scenario, the inputs are connected to the receivers in a round-robin fashion such that better load balancing among the receivers can be achieved. In fact, 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, according to our simulation.

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, such that the order is preserved when they arrive at next stage switch. Finding good matchings while maintaining packet order is the main challenge of the scheduling problem, which will be discussed in detail next.

5.3 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. Next we show that an optimal schedule can be found by computing the two matchings sequentially, i.e., first finding an optimal matching between input and output wavelength channels, then an optimal matching between transmitters and output channels that are still available. In the following we denote such two maximum matchings as M_c and M_b, respectively.

Theorem 4. *$M_c \cup M_b$ is an optimal schedule.*

Proof. It is clear that $M_c \cup M_b$ does not violate packet order since only head-of-flow packets are considered. Thus it remains to prove that $|M_c \cup M_b|$ is the maximum number of packets that can be transmitted without violating packet order, where $|M|$ is the cardinality of a matching M.

Assume M_c' and M_b' are the corresponding two matchings found with an arbitrary strategy under the precondition that packet order is preserved. For an arbitrary output port, denote the total number of wavelength channels as k, out of which o_c is matched to a packet in M_c, o_c' is matched in M_c', o_b is matched in M_b, and o_b' in M_b'. It is not difficult to see that the packet cut-through for different output ports are independent. Therefore it must be true that $o_c \geq o_c'$, given that M_c is an optimal matching.

On the other hand, full wavelength conversion is available when transmitters send packets to switch output. In other words, any two wavelength channels on an output port are equivalent when M_b is being computed. Hence

$$\begin{aligned} o_b' - o_b &\leq (k - o_c') - (k - o_c) \\ &= o_c - o_c' \end{aligned}$$

or, $o_b' + o_c' \leq o_b + o_c$. This implies that, for any output port, $M_c \cup M_b$ schedules more packets to it than $M_c' \cup M_b'$ does. Since M_c' and M_b' are matchings obtained under an arbitrary strategy that preserves packet order, this completes our proof that $|M_c \cup M_b|$ is the maximum number of packets that can be transmitted without violating packet order. As a result, $M_c \cup M_b$ is an optimal schedule. □

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

5.3.1 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.

Finding M_c

To compute M_c, what we have are the packet arrivals on each input wavelength channel in 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 current time slot. If there is no packet arrival for a flow, the corresponding sequence is then 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.

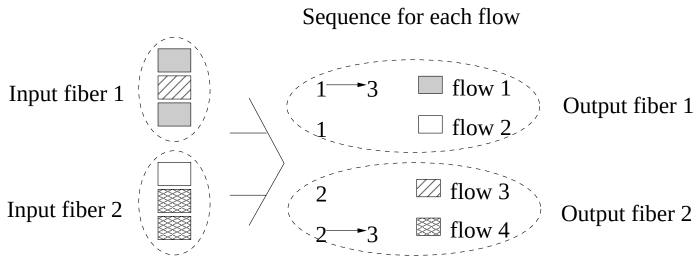


Figure 5.3: An example of converting packet arrivals to sequences. Flow 1 contains packet from input fiber 1 to output fiber 1. There are two packet arrivals for flow 1 on wavelength 1 and 3, respectively. Therefore the sequence for flow 1 is $\langle 1 \rightarrow 3 \rangle$. Similarly, the sequence for flow 2 is $\langle 1 \rangle$, since its sole packet arrival in this time slot is on wavelength 1.

In the simple example shown in Fig. 5.3, there are two input fibers and two output fibers, each containing three wavelength channels. We assume flow 1 contains packet from input fiber 1 to output fiber 1. There are two packet arrivals for flow 1 on wavelength 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 the rest three flows.

As mentioned earlier, the packet cut-through for different output fibers are 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 current time slot to 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 current time slot. For example, for flow 1 in Fig. 5.3, 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.

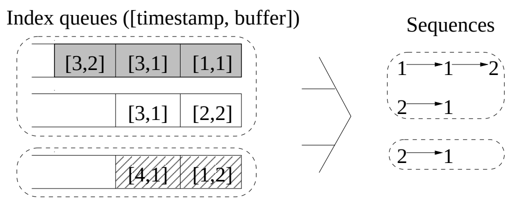


Figure 5.4: The relationship between index queues and sequences. An index queue keeps the timestamps and buffer indices of the packets being buffered. A sequence keeps the buffer indices only. Both the index queues and sequences are grouped according to the destined output fiber of the packets.

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 to 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 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.

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. 5.4 continues the example in Fig. 5.2 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. And 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. 5.4, 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, $\langle 1 \to 1 \to 2\rangle$ 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 $c_1, c_2, \ldots, c_N$, all in the range of $[0, k]$. 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. Plus, 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 are 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.

5.3.2 NP-Hardness and Inapproximability Proof

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

Theorem 5. *The MCP problem is NP-hard.*

Proof. We show that the MCP problem is NP-hard by reducing the set packing problem [64] 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, $\mathcal{S} = \{S_i \mid S_i \subseteq U, i = 1, 2, \ldots, n\}$, 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 $i = 1, 2, \ldots, n$, 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.

Sets	Sequences
{a, b, c}	$\langle\rangle$
{c, a}	$\langle\rangle$
{b, d}	$\langle\rangle$
{e}	$\langle\rangle$

(a)

Sets	Sequences
{a, b, c}	$\langle\rangle$
{c, a, z}	$\langle\rangle$
{b, d, y}	$\langle\rangle$
{e, x, w}	$\langle\rangle$

(b)

Sets	Sequences
{a, b, c}	$\langle i_{12} \to i_{13} \rangle$
{c, a, z}	$\langle i_{12} \rangle$
{b, d, y}	$\langle i_{13} \rangle$
{e, x, w}	$\langle\rangle$

(c)

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

(d)

Figure 5.5: 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.

- *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. 5.5 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, $i = 1, 2, \ldots, n$. 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 $\mathcal{P} = \{P_i^* \mid i = 1, 2, \ldots, n\}$, 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 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, 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 6. *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 5. 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 [64]. 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.

5.3.3 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 $2n_h$ [65]. 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 [64].

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

5.3.4 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 [5]. 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.

5.3.5 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.

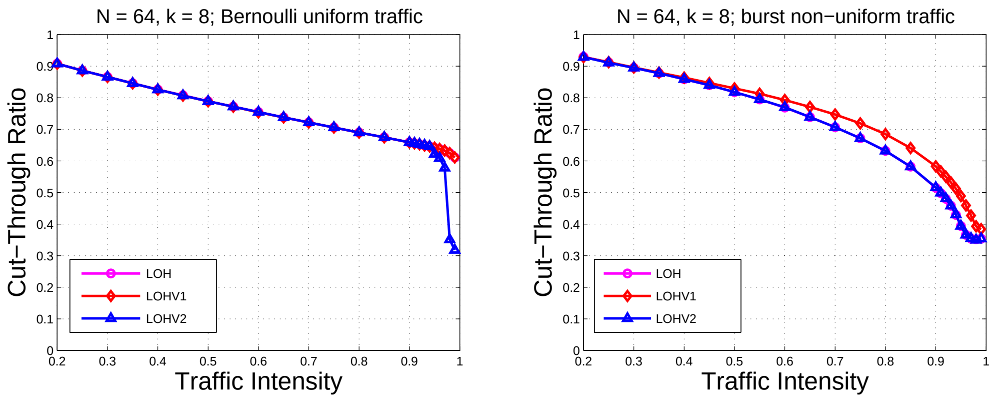
To incorporate this idea into the original LOH algorithm, the first and the third step of each iteration of the LOH algorithm are modified as follows.

- *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.

5.4 Performance Evaluation

In this section we present the simulation results. The basic simulation setup such as traffic models are similar to that in 4.5. The WDM OpCut switch is simulated with $N = 64$ and $k = 8$.



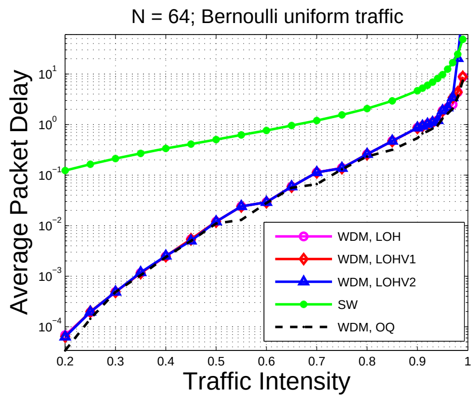
(a) (b)

Figure 5.6: 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.

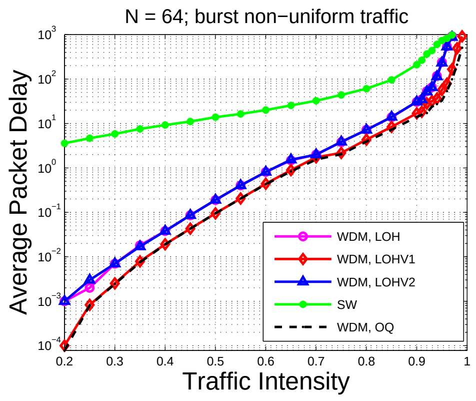
The LOH algorithm, as well as its variations as discussed in Section 5.3.5, is implemented with 8-iteration iSLIP. Each simulation was run for 10^6 time slots. The main performance criteria considered include the packet cut-through ratio and the average packet delay.

5.4.1 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. 4.8 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.



(a)



(b)

Figure 5.7: Average packet delay under different schedulers and traffic models. The notations of schedulers are the same as in Fig. 4.8. WDM: the multi-wavelength scenario, as opposite to single wavelength (SW). OQ: ideal output-queued switch.

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.

5.4.2 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 on each input wavelength channel and N-speedup output memory are assumed, such that a newly arrived packet can be 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 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. 3.6 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 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.

5.5 Conclusions

In this chapter, 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 by reducing the set packing problem to it. 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, which is in line with the best known approximation algorithm for set packing. 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.

Chapter 6

Energy-Aware Routing in Hybrid Optical Networks-on-Chip

In this chapter, we will study a special type of interconnect networks, the Network-on-Chip. In recent years, with the development of Multi-Processor System-on-Chip (MPSoC), the intra-chip communication is becoming more and more important to the performance of the entire system [66]. The NoC approach has been proposed to overcome this bottleneck.

A NoC example is shown in Figure 6.1. The fundamental idea of NoC, as summarized by [67], is to "route packets, not wires." Each processing core is connected to a local router. All the routers and the links interconnecting the routers, form the NoC. The communication between two processing cores in a NoC is very similar to that between two nodes in a computer network. As a result, a lot of existing research in computer networks has been leveraged for NoCs. However, due to the tight constraints in space, time and power, NoCs significantly differ from traditional off-chip networks. In particular, given the power consumption budget, it is unclear whether traditional electronic interconnects will be sufficient for NoC communications as the demand on bandwidth and latency keeps increasing [68] [69] [70].

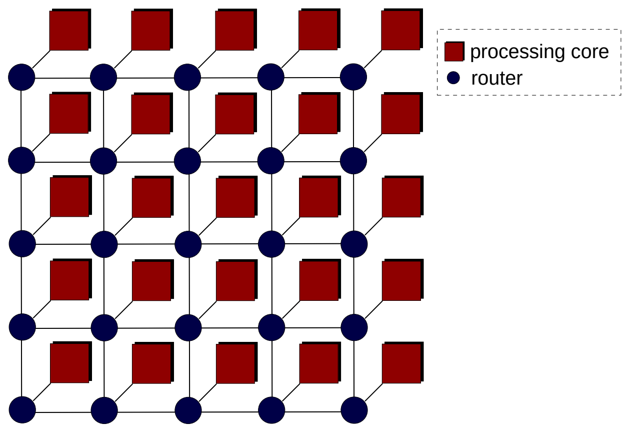


Figure 6.1: A typical architecture of a NoC.

In this chapter, we study the routing problem in optical NoCs with arbitrary topologies. To be more specific, we will consider the energy-aware routing problem in such networks. Currently, minimum hop count routing schemes are dominant in both electronic and optical NoCs. However, while the minimum-hop-count path is likely to be the most energy-efficient path in electronic NoCs, the situation is different in the optical domain, as will be shown later. We will investigate the trade-off between energy consumption and the average path length in optical NoCs, as well as how to reduce energy consumption without introducing extra transmission latency.

The rest of the chapter is organized as follows. Section 6.1 briefly describes the basic architecture of existing optical NoCs, and a key characteristic of optical routers proposed for optical NoCs. Section 6.2 discusses the energy-aware routing problem in optical NoCs and two approaches to addressing this problem, as well as its effectiveness in various topologies. Section 6.3 proposes possible extensions to current hybrid optical NoCs, including the application of optical burst switching (OBS), and energy-aware adaptive routing in optical NoCs. Finally, Section 6.4 concludes the chapter.

6.1 Optical NoC

In this section we briefly describe the architecture of optical NoCs considered in this chapter. Due to the fact that buffering and packet header processing are currently difficult to implement in optical domain, especially at on-chip level, most existing optical NoCs are based on a hybrid approach consisting of two layers [68] [71]. The control layer is in electronic domain, which is used to exchange control information and other short messages. An optical interconnection network layer, comprised of optical routers interconnected by waveguides, is responsible for transmitting the actual data.

Similar to the wavelength-routed networks, a light-path must be set up before any data can be transmitted in the optical layer. This is done by a control packet in the electronic layer, which traverses along the path to be established and makes reservations. In case that some reservation cannot be satisfied, the control packet is dropped and a corresponding path-blocked packet is generated. The path-blocked packet is backtracked along the path to release the reservations already made, and to notify the potential sender that a path cannot be established at this moment. If the path is ready, the message is transmitted in optics without being buffered at any intermediate node. When the transmission is completed, a packet is sent to tear down the path. In the rest of this chapter we will focus on this type of NoC and refer to it as the hybrid optical NoC.

A major advantage of using the light-path mechanism, as pointed out in [68], is the bit-rate transparency property [72]. Traditional electronic routers switch every bit of the data, thus the power dissipation scales with the bit rate. On the other hand, optical routers switch at the message level, hence the energy dissipation is unrelated to the bit rate. This nice property makes it possible in optical domain to achieve very high-bandwidth at a relatively low power budget.

Currently, most of the proposed optical NoC architectures use microresonators as the basic building block for optical routers, which can work at a very high speed as a 30ps switching time

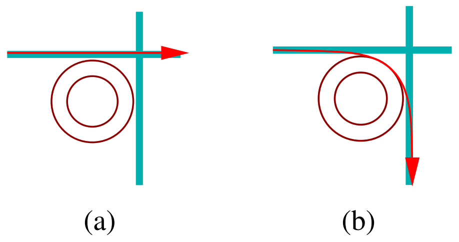


Figure 6.2: Basic operations of a microresonator. (a) Off state. The signal goes through without turning. (b) On state. The signal is forced to change direction.

has been demonstrated in [73]. The basic operations of a microresonator is illustrated in Figure 6.2. When a microresonator is not turned on, its off-state resonance wavelength, which is determined by the material and the internal structure, is different from that of the optical signal. As a result, the signal passes through uninterrupted. When the microresonator is powered on, the resonance wavelength is shifted to that of the optical signal and forces the turning of the signal. The key point here is that the microresonator only consumes energy when it needs to be turned on to change the direction of the incoming signal.

Different optical router architectures have been proposed based on microresonators, such as the 4×4 switch in [68], the OXY router in [74], and the optical turnaround router in [71]. While these architectures differ in the number of input/output ports, the number of microresonators used, and the way the microresonators are organized, they do share one common characteristic: the energy required to switch a message is determined by the number of "on" microresonators, which is in turn determined by the incoming and outgoing ports of the message. If no microresonator needs to be turned on to switch a message, then there is no energy consumption by the router ; otherwise, the energy consumption is proportional to the number of microresonators that are on. For example, the 4×4 switch in [68] does not consume any energy when switching a message between east port and west port, or between south and north, but needs to turn on at least one microresonator in all other cases.

6.2 Energy-Aware Routing in Optical NoCs

In this section, we discuss the energy-aware routing problem in hybrid optical NoCs. We first quantify the energy required to route a message in both an electronic NoC and a hybrid optical NoC, then discuss how the energy-aware routing problem can be addressed in optical NoCs.

6.2.1 Energy Consumption for Routing a Message

In a network where the components are built in electronics, shortest path routing is usually adopted, because both the latency and the energy consumption of routing a packet from the source to the destination is proportional to the distance that the packet traverses. Similarly, since it is usually assumed that the distances between neighbor nodes in a NoC are identical, a minimum hop count routing policy is usually employed in NoCs. However, as discussed above, due to the special architecture of current optical on-chip routers, such a minimum-hop path may not be energy-wise optimal. To put it more formally, let's first consider the energy that will be needed in an electronic NoC to route a packet from the source to the destination:

$$E_e = E_e^0 \cdot h$$

where E_e^0 is the energy consumed by the packet at a single router and h is the hop count of the path from the source to the destination. E_e^0 includes, for example, the energy required to transmit the packet across the internal crossbar and the inter-router link, to temporarily buffer the packet, and to make routing decisions. It is roughly proportional to the size of the packet [68]. On the other hand, for hybrid optical NoCs, the energy consumption consists of two parts, namely, the control packets in electronics and the actual data in optics. That is,

$$E_o = E_{ctrl} + E_{data}$$

There is no significant difference in the way of being routed between an electronic control packet in a hybrid optical NoC and a regular packet in an electronic NoC. The only difference is that the former , which contains no payload, is of a fixed size and is much shorter than the latter on average. Thus, if we denote the energy consumed by a control packet at an intermediate router as E^0_{ctrl}, then E^0_{ctrl} is fixed and typically much smaller than E^0_e, and

$$E_{ctrl} = E^0_{ctrl} \cdot 2h$$

The factor 2 in front of h is due to the fact that typically there will be two control packets accompanying each optical message. The most dramatic difference between energy consumption of electronic NoCs and hybrid optical NoCs, however, lies in E_{data}, which can be expressed as

$$\begin{aligned} E_{data} &= N_{on} \cdot E^0_{mr} + E_i \\ &= N_{on} \cdot P^0_{mr} \cdot \frac{d}{c} + E_i \end{aligned}$$

where E_i is the energy consumed by the E/O and O/E interfaces and is linear to the data size, N_{on} is the total number of microresonators that need to be turned on along the path in all intermediate optical routers, E^0_{mr} and P^0_{mr} are respectively the energy and power consumed by a single "on" microresonator, d is the distance the data need to traverse inside a router, and c is the speed of light in silicon optical waveguide. It can be seen that the E_{data} has nothing to do with the hop count. In an extreme case, E_{data} can be zero for a long path, as long as there is no microresonator turned on.

If we further let $\sigma = E^0_{ctrl}/E^0_{mr}$, then we have

$$\begin{aligned} E_o &= E^0_{ctrl} \cdot 2h + N_{on} \cdot E^0_{mr} + E_i \quad (6.1) \\ &= (2\sigma h + N_{on}) E^0_{mr} + E_i \quad (6.2) \end{aligned}$$

According to the data in [71] and [68], E^0_{ctrl} and E^0_{mr} are roughly comparable, while they change rapidly with the technique development. In the rest of this paper we focus on the cases of $0.1 \le \sigma \le 10$.

Furthermore, for a given data message, E_i usually can be modeled as a constant and is not directly affected by the routing policy. Therefore, in a hybrid optical NoC, an energy-wise optimal path to route a message is the path that minimizes $2\sigma h + N_{on}$. Such a path is not necessarily a path with minimum hop count. Next we will dig more into this problem and discuss how it can be converted into a shortest path problem.

6.2.2 The Energy-Aware Routing Problems

Given Equation (6.2), it is straightforward to address energy-aware routing in a hybrid optical NoC as the following problem:

- Problem 1. Find a path between two processing cores that is the most energy-efficient, that is, with $\sigma h + N_{on}$ minimized.

A potential problem with the above formalization is that a resulted path may be very long. On one hand, in an extreme case, the assumption that E_i is constant for a given data message may no longer hold, since higher laser power may be required to compensate for the higher loss. On the other hand, a hop-constrained shortest path problem is NP-hard. Therefore, in the cases where the loss may become a problem, we can try to address the energy-aware routing problem in another way:

- Problem 2. Find a path between two processing cores that has the minimum N_{on} among all paths that have the minimum hop count.

This formalization also makes sense for latency-sensitive data, such that the energy consumption can be reduced without introducing any extra latency. The two problems are closely related and can be solved in a similar way. In the following we will mainly focus on Problem 1, and will show how Problem 2 can be solved at the end of this section.

Solving Problem 1

As has been shown in Section 6.1, for a microresonator-based optical router, the number of microresonators that need to be turned on for routing a packet is solely determined by the incoming port and outgoing port of the message. As a result, any existing microresonator-based optical router, no matter how many ports it has, or what its internal fabric is, can be modeled as an undirected, complete graph in terms of power consumption.

To be more specific, since all transmissions are completely bidirectional, we will assume all the ports of an optical router are fully duplex, which is true for all existing architectures. In other words, each port is used as both an input port and an output port at the same time. Assuming an optical router contains N ports, then it can be modeled by a complete graph with N nodes, each denoting a port of the router. There is an edge between any pair of nodes. The cost of the edge equals the minimum number of microresonators that need to be turned on to route a packet from one node of the pair to the other. We refer to this graph as the energy consumption graph of the corresponding optical router in the rest of the chapter.

Figure 6.3 shows some existing optical router architectures proposed for optical NoCs and their corresponding energy consumption graphs. As mentioned earlier, in the 4×4 router [68],

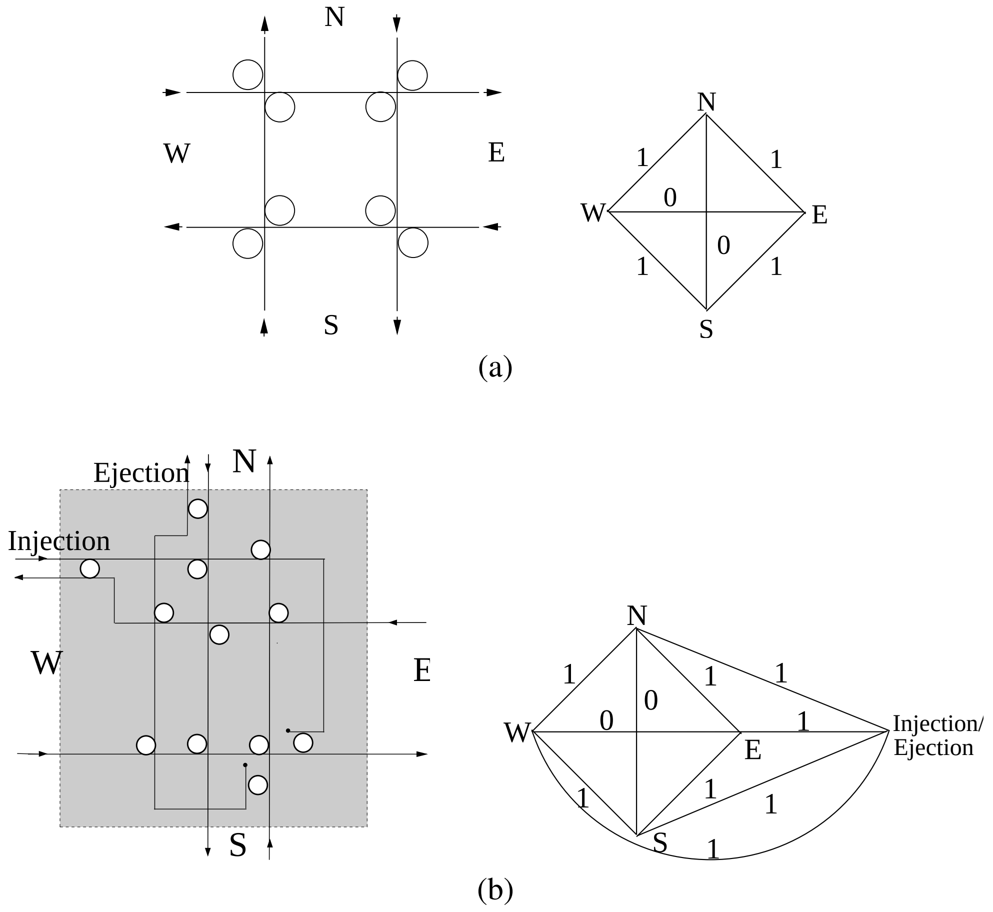


Figure 6.3: (a) The 4 $\times$ 4 router in [68] and its energy consumption graph. The numbers denote the cost of the corresponding edges. (b) The OXY router in [74] and its energy consumption graph.

no microresonator needs to be in “on” state to route packets between east port and west port, or between north port and south port, i.e. the energy cost is 0. However, one microresonator has to be turned on for any of other cases. The energy consumption graph of OXY router is similar to that of the 4 $\times$ 4 router, except that there is an additional port for injecting / ejecting local packets. The cost of transmitting packets between this port and any other port is 1.

Having illustrated how to convert an optical router into an energy consumption graph, now solving the energy-efficient problem takes only one step further. In the graph that represents an optical NoC, we replace each node that denotes an optical router with the energy consumption graph of the router. A link between two adjacent routers is represented by an edge between the corresponding ports of the two routers. We assume that the cost of all such edges is identical in terms of energy consumption, i.e., 2σ. As a result, the problem of finding a path between two processing cores that minimizes energy consumption is equivalent to finding a shortest path in the converted graph, between the two nodes representing the local ports of the corresponding processing cores. This problem can then be solved by any existing shortest-path algorithms, for example, Bellman-Ford algorithm, or, since all costs are positive, Dijkstra's algorithm [65]. Similarly, algorithms such as Floyd-Warshall algorithm can be used to solve the all-pair version of the problem.

As a shortest path problem, the time complexity of the energy-aware routing problem is determined by the number of nodes and edges in the converted graph. Assume an arbitrary NoC contains R optical routers and L inter-router links, with the i_{th} router containing p_i ports, $i = 0, 1, 2, \ldots, R-1$. Then the total number of nodes in the converted graph is given by

$$V = \sum_{i=0}^{R-1} p_i$$

and the total number of edges is given by

$$E = L + \sum_{i=0}^{R-1} \frac{p_i(p_i - 1)}{2}$$

since the i_{th} router is replaced by a complete graph with p_i nodes in the converted graph.

If Dijkstra's algorithm is used, and is implemented with a Fibonacci heap [65], then the time complexity of finding an energy-wise optimal path between a single pair of processing cores is

$$\begin{aligned} T_s &= O(E + V \log V) \\ &= O(L + \sum_{i=0}^{R-1} {p_i}^2) \end{aligned}$$

To find an energy-wise optimal path between all pairs of processing cores, the time complexity is

$$\begin{aligned} T_a &= O(V^3) \\ &= O((\sum_{i=0}^{R-1} p_i)^3) \end{aligned}$$

As a comparison, a minimum-hop path on such a NoC (with R optical routers and L inter-router links) can be found by the breath first search or depth first search algorithm, which takes $O(R + L)$ for a single pair of processing cores and at most $O(R(R + L))$ for all pairs. It can be seen that the time complexity of energy-aware routing is higher than that of minimum-hop routing. However, since the computation can be performed off-line and needs to be done only once (instead of being required by each message), such increase in complexity is unlikely to become an issue.

Solving Problem 2

As mentioned earlier, theoretically, a path that minimizes energy consumption in optical NoC could be very long. This is especially the case when σ is small and the overall energy consumption for routing a message is dominated by E_{data}, which is independent of the hop count. In this case, the minimization of energy consumption is at the cost of introducing extra delay to the messages. When this is undesirable, we can try to solve Problem 2 instead. For Problem 2, the energy consumption is minimized only under the precondition that the hop count is minimized.

Next we show how to solve Problem 2 by converting it to a shortest path problem. For an arbitrary optical NoC, let N_m denote the total number of microresonators inside all routers in the NoC. We set the cost of all the inter-router links to σ' such that $\sigma' > N_m$, then run any shortest path algorithm in the converted graph. It can be shown that the path found is then a path that satisfies the requirement of Problem 2.

Lemma 4. *A path found according to the above procedure requires the minimum energy among* *all paths with the minimum hop count.*

Proof. Denote the path returned by the shortest path algorithm as P and its hop count as h_P, and denote the number of "on" microresonators along P as $N_{on,P}$. First we prove that P is a path with the minimum hop count. Assume that there is another path P' with a smaller hop count, i.e., $h_{P'} < h_P$. Then

$$\begin{aligned} C_P &= \sigma' h_P + N_{on,P} \\ &\geq \sigma' h_P \\ &\geq \sigma'(h_{P'} + 1) \\ &> \sigma' h_{P'} + N_m \\ &\geq C_{P'} \end{aligned}$$

where C_P and $C_{P'}$ are the cost of P and P', respectively. Since $C_P > C_{P'}$ contradicts the fact that P is a shortest path, such P' does not exist. Hence h_P is minimum.

It is straightforward to prove that $N_{on,P}$ is the minimum among all paths that have the minimum hop count, since otherwise P cannot be a shortest path. That completes the proof. $\square$

Therefore, similar to Problem 1, Problem 2 can be solved by running shortest path algorithms on the converted graph. Since the only difference is the cost of inter-router links, the time complexity of solving Problem 2 is identical to that of Problem 1 as analyzed in the previous subsection.

6.2.3 Energy-Aware Routing in Different Topologies

In this section we apply the idea of energy-aware routing discussed above to different optical NoC topologies and investigate its effectiveness.

Mesh / Torus Networks

Mesh / Torus are popular topologies for NoC networks due to their regularity and simplicity. The simplest routing algorithms for these topologies are based on dimension order, for example, the XY routing. With XY routing, a packet is first routed in the horizontal direction, then in the vertical direction, until the destination is reached. It is interesting to note that, for an optical mesh / torus NoC, XY routing is optimal in terms of energy consumption. The optimality consists of two aspects: (1) Being an oblivious algorithm, XY routing requires minimal logic hence may simplify the router architecture. (2) The path found by XY routing has the minimum hop count and turns on the minimum number of microresonators. It is true because XY routing makes at most one turn from the source to the destination.

As a contrast, other routing algorithms, for example, the Odd-Even routing algorithm [75], may make multiple turns to route a packet, hence is not optimal in terms of energy consumption.

As a simple example, Figure 6.4 shows two paths between two nodes in a torus network that may be used according to XY routing and Odd-Even routing, respectively. We can observe that while both paths are minimum hop-count paths, the XY routing path makes only one turn, while the other algorithm makes three turns. If we assume the 4×4 optical router in [68] is used, then the former turns on only one microresonator, whereas the latter turns on three.

Gaussian Networks

Gaussian networks [76] [77] are another type of topology of interest to optical NoCs. A Gaussian network corresponds to a Gaussian integer $a + bi$, where a and b are both integral. Figure 6.5 illustrates a Gaussian network of $3 + 4i$, where there are 25 nodes in total, each labeled by its real and imaginary coordinates.

Similar to that in torus networks, each node in a Gaussian network has a degree of 4. However,

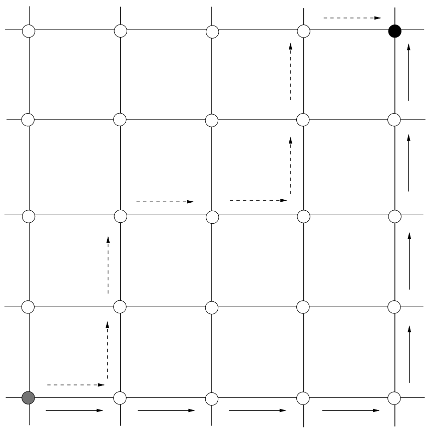


Figure 6.4: One possible path by XY routing (solid line) and one possible path by Odd-Even routing (dashed line). If the 4×4 optical router in [68] is used, the former turns on only one microresonator, while the latter turns on three.

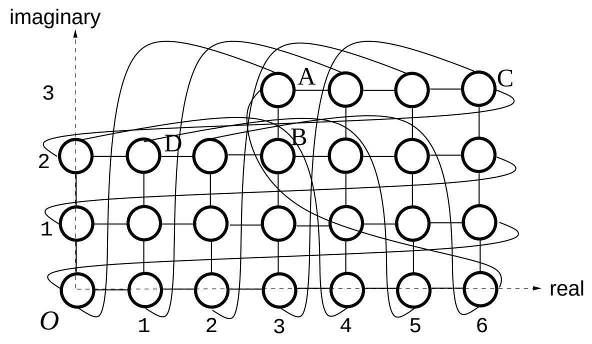


Figure 6.5: A Gaussian network of $3 + 4i$ with 25 nodes.

the wraparound edges are defined differently. A Gaussian network of $a + bi$ contains the following edges [76]:

- x is connected to $(x + a) + (b - 1)i$ if $0 \leq x \leq b - 1$;
- x is connected to $(x - b) + (a - 1)i$ if $b \leq x \leq a + b - 1$;
- yi is connected to $(a + b - 1) + (y + b - a)i$ if $0 \leq y \leq a - 1$; and
- $a + yi$ is connected to $(a + b - 1) + (y - a)i$ if $a \leq y \leq b - 1$.

Gaussian networks attract increasing attentions in recent years as they outperform mesh / torus networks in terms of diameter when the network size is sufficiently large. For example, the diameter of the Gaussian Network $30 + 40i$ is 40, while for a torus network with the same number of nodes (50×50), the diameter is 50.

Now consider using the Gaussian network as the underlying topology for optical NoCs. Due to the way the wraparound edges are defined in the Gaussian network, a node may be directly connected to another node that is in a different row and different column, which is not possible in a torus network. For example, in Figure 6.5, there is an edge between node i and node $6 + 2i$. An interesting consequence is that, for some Gaussian networks (for example, the one in Figure 6.5), there is a path between any two nodes that does not make any "turns," in other words, does not require any microresonator to be turned on in any intermediate router to route messages. For example, in Figure 6.5, there is a three-hop path from node A $(3 + 3i)$ to D $(1 + 2i)$ through B $(3 + 2i)$, which needs to turn on a microresonator at B. Meanwhile, there is also another path from A to D through C $(6 + 3i)$, which is one hop longer but does not require any microresonator to be turned on.

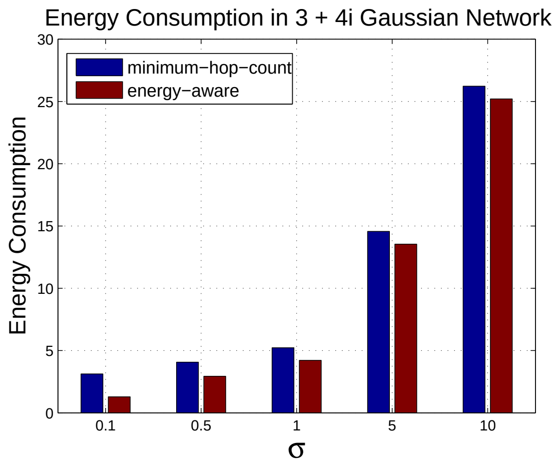


Figure 6.6: Energy consumption in the $3+4i$ Gaussian network under minimum hop count routing and energy-aware routing. σ varies between 0.1 and 10.

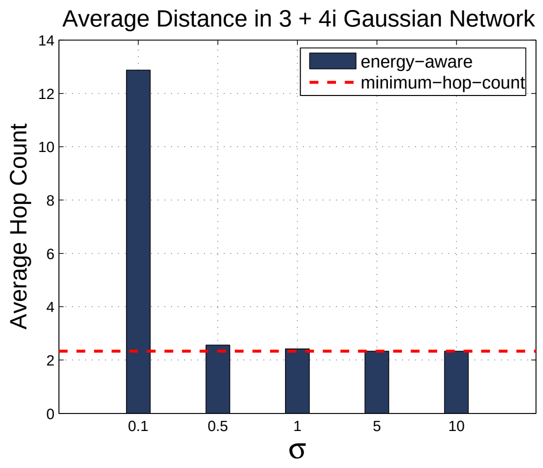


Figure 6.7: Average distances in the $3 + 4i$ Gaussian network under minimum hop count routing and energy-aware routing.

Figure 6.6 shows the energy consumption in the Gaussian network in Figure 6.5 under both minimum hop count routing and energy-aware routing. The results are based on synthetic traces, similar to other studies carried out in this field [68] [74]. While real traces are highly desirable, synthetic traces provide a convenient way to gain some insight into the problem. In addition, we follow the common assumption that the traffic is uniform. That is, the destinations of data messages from a processor are uniformly distributed over all other processors. It can be seen that when σ is smaller than 1, the percentage of energy saved by selecting a minimum-energy path instead of a shortest path is significant. As σ becomes larger, unsurprisingly, minimum-energy paths gradually converge to shortest paths as E_{ctrl} becomes dominating.

The average path lengths under energy-aware routing and minimum hop count routing are compared in Figure 6.7. It can be seen that when $\sigma = 0.1$, the paths that are most energy efficient tend to be rather long. In fact, in this case the control overhead contributes to all the energy consumption. It can be considered as a trade-off between the latency and the energy consumption. For the rest of the cases, however, the average distance under energy-aware routing is very close to that under minimum hop count routing. In other words, some energy can be saved without introducing noticeable extra delay to the messages.

Arbitrary Topologies

Most existing research on optical NoCs assumes a regular network topology. However, as pointed out in [78], there are reasons why customization is also desired. The processing cores may have different sizes and communication requirements, which calls for a heterogeneous network topology. The NoC may be application-specific hence the workload is known beforehand, which makes a full customization of the network topology possible.

In this section we investigate the effectiveness of applying the idea of energy-aware routing to arbitrary topologies. We consider connected graphs generated randomly. The number of microresonators that need to be turned on to connect two ports in an optical router is assumed to be a random integer between 0 and 4, and σ varies between 0.1 and 10. We have conducted simulations on different random networks. The results are shown in Figure 6.8 and Figure 6.9.

Two random networks are simulated. Network 1 contains 10 processing cores, while network 2 has 100 processing cores. From Figure 6.8, it can be seen that when σ is small ($= 0.1$), the energy consumed on switching a packet can be reduced by as much as about 40% if energy-aware routing is adopted instead of minimum hop count routing. As σ increases, the energy saving effect becomes less significant. This is similar to that in the Gaussian network, that when the control overhead dominates the energy consumption, the paths with overall minimum energy consumption converge to the minimum hop count paths. Figure 6.8 also shows that the energy saving effect is more obvious in network 2 compared to that in network 1, indicating that the energy-aware routing algorithm is more effective in large networks.

Figure 6.9 compares the average distance between nodes under energy-aware routing and minimum hop count routing. When σ is small, it is possible that an energy-wise optimal path may travel many extra hops to avoid turning on microresonators, thus the average distance is longer

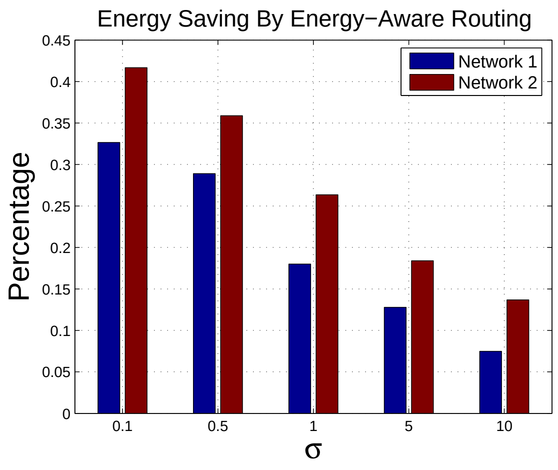


Figure 6.8: Energy saving by energy-aware routing in random topologies, compared to minimum hop count routing. σ varies between 0.1 and 10.

than that of minimum hop count routing by a clear margin. When σ becomes large, as discussed earlier, minimum hop count paths also become minimum-energy-consumption paths, hence the difference in hop count diminishes.

6.3 Some Extensions

In this section we discuss some possible extensions beyond current hybrid optical NoCs and the routing algorithms. The basic requirement is that these extensions should be easy to implement and have the potential to significantly improve the system performance.

6.3.1 Optical Burst-Switched NoC

As mentioned earlier, currently most proposed optical NoCs are based on circuit switching. To send information from one node to another, a light path has to be established. Apparently, the disadvantage of circuit-switching is that it cannot well adapt to dynamic traffic, because the

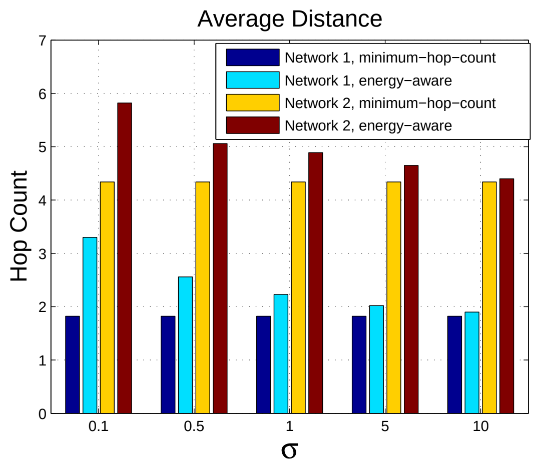


Figure 6.9: Average distances between nodes in random topologies under minimum hop count routing algorithm and energy-aware routing algorithm, respectively.

entire bandwidth on the light path is reserved by the source-destination pair and cannot be used by others even when there is no traffic between the pair, leading to low resource utilization. Similar situation has occurred in off-chip networks. To overcome this problem, two other techniques have been proposed for optical networks, namely, the Optical Packet Switched (OPS) network and the Optical Burst Switched (OBS) network.

An OPS network is like the electronic packet switched network and can better adapt to dynamic traffic. There has been some attempt to implement OPS NoC [79]. However, currently OPS is difficult to implement because it needs optical memories at the switches, given that fiber-delay-line based buffers are limited, yet slow light is still in its laboratory stage 1.2.

Given all the conditions, optical burst switching (OBS) [80] [81] appears to be a more viable choice for data transmission in optical NoCs. It has higher bandwidth utilization compared to circuit-switching, and it does not need a large optical buffer in the routers compared to OPS. While there are different OBS schemes, the basic idea is the same: using one-way reservation instead of two-way. With OBS, data are first assembled into bursts at the source node. A control packet is still necessary for light path setup, which traverses the path to be established in an electronic control plane, or in a dedicated control wavelength. The control packet makes reservation along the path. No confirmation packet is sent back to the source node. Instead, after the control packet is sent, the source node waits for a certain amount of time and sends out the message, or data burst. The burst does not have to be buffered at intermediate routers and cuts through the routers without delay. Moreover, no control packet is required to tear down the light path. The bandwidth is released as soon as the burst leaves a router.

Note that typical OBS will not try to inform the source node if a bandwidth reservation request cannot be satisfied. In other words, the data burst will be sent out without any knowledge of whether the control packet has successfully reserved the needed bandwidth along the path or not, and may have to be dropped at any intermediate node. Therefore, effective contention resolution techniques are critical to OBS networks to prevent excessive bandwidth wasting. Various techniques have been proposed for such a purpose. For example, in the context of optical NoCs, data bursts can be temporarily buffered by FDLs, or deflected to another output port at intermediate nodes when contention arises. Yet there is a unique contention resolution approach in optical domain – wavelength conversion. In a WDM environment, a burst can be converted and transmitted on a different wavelength channel of its desired output port without being deflected.

Figure 6.10 compares the timeline of circuit-switching and OBS. Depending on the relative size of a control packet and a burst, the saving by OBS in path setup time overhead could be significant compared to circuit-switching. Energy-wise, OBS also has its advantage as it requires fewer control packets than circuit-switching does.

Two of the most important parameters of OBS are the burst size and the interval between a control packet and the corresponding burst. The former affects the assembly time at the source node hence the latency of the data. The latter affects the burst blocking probability and QoS support [82]. In the context of an optical NoC, the burst size cannot be too large since the requirement on

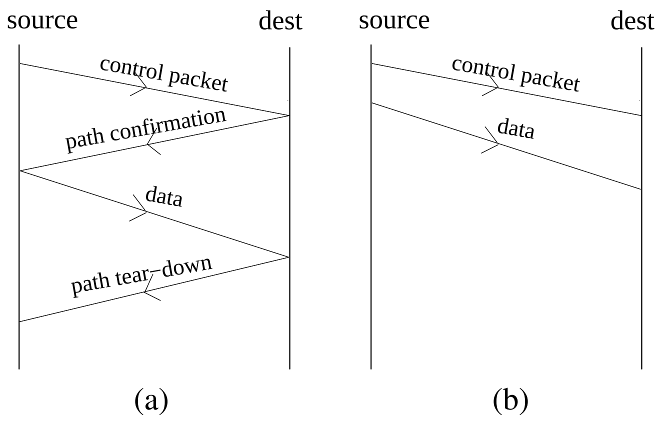


Figure 6.10: The timeline of (a) circuit-switching (b) OBS. OBS does one-way reservation thus does not need a confirmation packet for path setup, nor does OBS require the path tear-down process as the duration of the data burst is known to the routers when the path is being established.

latency is stringent and the data cannot wait for too long at the source node, hence any fancy assembly algorithm may not be viable.

On the other hand, compared to Internet routers, the routers in NoC have very limited electronic buffer due to space restriction. For example, a popular electronic NoC router architecture based on input-queued crossbar is equipped with about 1k bit buffer at each input port [83]. While the buffer size is obviously too small to leverage any sophisticated scheduling algorithm for traditional crossbar switches, it does provide some room for the scheduling of control packets at each intermediate router. To be more specific, the control packets can be temporarily buffered at an intermediate router for a certain amount of time. Then, instead of making a scheduling decision for each control packet as soon as it arrives, the decision can be made based on the information of all control packets that arrive during this time interval. Since more information is available, it can be expected that a schedule that leads to better resource utilization can be achieved. Meanwhile, the per-hop delay of a control packet cannot be too long, otherwise, either the offset between a control packet and its corresponding data burst must be made very large, or the burst may become ahead of the control packet and has to be dropped.

6.3.2 Energy-Aware Adaptive Routing in Optical NoC

Opposite to static routing, adaptive routing describes the capability of the system to dynamically adjust the path from the source to the destination in response to some condition change. There have been efforts to incorporate adaptive routing into NoC routers, for example, [84] [85]. Combining that with the idea of energy-aware routing, the following energy-aware adaptive routing scheme can be adopted for optical NoCs.

For each destination node, multiple candidate output ports are maintained. All of these candidate ports belong to some path with minimum hop count from the current node to the destination, and they are sorted by their power consumption. When a control packet needs to be scheduled, the candidate output ports are checked in the sorted order until the first available one is found. Then the control packet and the corresponding data message will both be routed through that output port. If none such output port is available, the router either buffers the control packet for a longer time and tries to schedule it later, or simply drops it, depending on the specific policy adopted.

By allowing alternate paths at each router, above adaptive routing algorithm has the potential to reduce energy consumption without introducing additional delay in communication. Instead, since a control packet can be re-routed if the best candidate output port is currently unavailable, it is possible that the buffering delay for a control packet at intermediate routers will be shortened. Consequently, the path setup overhead in circuit-switched or OBS optical NoCs can be reduced. Another potential benefit of the adaptive routing is to balance the workload and eliminate hotspots.

6.4 Conclusions

Network-on-Chip (NoC) has been proposed as a promising technique to support the ever-increasing intra-chip communication load for Multi-Processor System-on-Chip (MPSoC). To overcome the power consumption problem in electronic NoCs, optics based architectures have been proposed. In this chapter, we have studied the routing problem in optical NoCs with arbitrary network topologies. Currently, minimum hop count routing schemes are dominant in both electronic and optical NoCs. However, while the minimum-hop-count path is likely to be the most energy-efficient path in electronic NoCs, we have showed that it is not the case in the optical domain. We studied the architecture of different optical NoC routers proposed recently, and discussed how to model an optical router into an energy consumption graph. By converting the energy-aware routing problem into a shortest-path problem and applying our approach to different NoC topologies, we showed that the energy consumed in data communication in an optical NoC can be significantly reduced. We also proposed several extensions to current optical NoCs, including the use of OBS in optical NoCs to reduce control overhead, and an adaptive routing mechanism to reduce energy consumption without introducing extra latency. The simulation results showed the effectiveness of the proposed algorithms.

Chapter 7

Conclusions

This dissertation studied the packet scheduling and performance evaluation problem in optical switches and interconnects. A suite of algorithms and schemes for packet scheduling in optical switches were presented. For output-buffered optical switches, the proposed Augment to Full and schedule construction algorithms find a schedule that simultaneously maximizes throughput and minimizes packet delay in each time slot. And the complexity of the proposed algorithms asymptotically matches the lower bound of the original scheduling problem. For input-buffered optical switches, we established the requirements on admissible traffic, and proposed a novel fiber delay line based optical input buffering structure with flexible delay. By combining the new buffer with a weight-based packet scheduling algorithm, named Most-Packet Wavelength-Fiber Pair First (MPWFPF), we were able to theoretically prove that an input-buffered optical switch can deliver at its maximum capacity as long as input traffic is admissible and satisfies the strong law of large numbers. We further proposed fast scheduling algorithm, WDM-iSLIP that can efficiently determine an approximate optimal scheduling with much lower time complexity. Also, the dissertation systematically studied the packet scheduling problem in the OpCut Switch, a hybrid optical switching architecture with recirculating electronic buffer. We designed a basic scheduler for Op-

Cut switches, which decomposes the scheduling process into three stages. For single-wavelength OpCut switch, we proposed a mechanism to (adaptively) pipeline the scheduling procedure to reduce scheduler complexity and improve system throughput. For the WDM scenario, we first proved that the optimal scheduling problem is NP-hard and inapproximable in polynomial time within any constant factor, then presented a bounded practical approximating algorithm as well as its variations to further improve performance or reduce implementation complexity. In addition, we studied energy-aware routing in hybrid optical NoCs, and presented approaches to efficiently achieve that.

The research contained in this dissertation combines algorithm design, hardware design, analytical, and simulation techniques. We expect the research result to have a significant impact on switch architecture design, scheduler implementation and performance evaluation for the development of future high and ultra high speed optical switching networks. The outcome of this project is generally applicable to a wide range of interconnect networks, including computer networks, inter-chip switching, intra-chip switching, and networks-on-chip.

Bibliography

[1] M. Karol, M. Hluchyj and S. Morgan, "Input versus output queueing on a space-division packet switch," *IEEE Transactions on Communications*, vol. 35, pp. 1347-1356, Dec. 1987.

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

[3] A. Smiljanic, R. Fan and G. Ramamurthy, "RRGS-round-robin greedy scheduling for electronic/optical terabitswitches," *GLOBECOM 1999*, pp. 1244-1250, 1999.

[4] 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.

[5] N. McKeown, "The iSLIP scheduling algorithm for input-queued switches," *IEEE/ACM Trans. Networking*, vol. 7, no. 2, pp. 188-201, 1999.

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

[7] H. Ngo, D. Pan and Y. Yang, "Optical switching networks with minimum number of limited

range wavelength converters," *IEEE/ACM Transactions on Networking*, vol. 15, no. 4, pp. 969-979, August 2007.

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

[9] 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.

[10] C. Minkenberg, et. al, "Designing a crossbar scheduler for HPC applications," *IEEE Micro*, vol. 26, pp. 58-71, May 2006.

[11] 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.

[12] 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," *Proc. ACM/IEEE Conference on Supercomputing (SC '05)*, Seattle, WA, Nov. 2005.

[13] Y. Yang and J. Wang, "Designing WDM optical interconnects with full connectivity by using limited wavelength conversion," *IEEE Transactions on Computers*, vol. 53, no. 12, pp. 1547-1556, Dec. 2004.

[14] H. Yang and S.J.B. Yoo,"All-optical variable buffering strategies and switch fabric archi-

tectures for future all-optical data routers," *Journal of Lightwave Technology*, vol. 23, pp. 3321-3330, Oct. 2005.

[15] W.D. Zhong and R.S. Tucker, "A new wavelength-routed photonic packet buffer combining traveling delay lines with delay-line loops," *Journal of Lightwave Technology*, vol. 19, No. 8, Aug. 2001.

[16] J.B. Khurgin, "Light slowing down in moire fiber gratings and its implication for nonlinear optics," *Physics Review A.*, vol. 62, July 2000.

[17] Y. Okawachi, M.S. Bigelow, J.E. Sharping, Z. Zhu, A. Schweinsberg, D.J. Gauthier, R.W. Boyd and A.L. Gaeta, "Tunable all-optical delays via Brillouin slow light in an optical fiber," *Phys. Rev. Lett.*, **94**, 153902, 2005.

[18] C.J. Chang-Hasnain, P.C. Ku, J. Kim and S.L. Chuang, "Variable optical buffer using slow light in semiconductor nanostructures," *Proceedings of the IEEE*, vol. 91, pp. 1884-1897, November 2003.

[19] F. Xia, L. Sekaric and Y. Vlasov, "Ultracompact optical buffers on a silicon chip,", *Nature Photonics* **1**, pp. 65-71, 2007.

[20] R.M. Camacho, C.J. Broadbent, I.Ali-Khan and J.C. Howell, "All-optical delay of images using slow light," *Physics Review Letter*, 98, 043902 (2007).

[21] R.S. Tucker, P.-C Ku and C.J. Chang-Hasnain, "Fundamental limitations of slow-light optical buffers," *Optical Fiber Communication Conference, 2005. Technical Digest. OFC/NFOEC*, vol. 3, Mar. 2005.

[22] L. Liu and Y. Yang, "Optimal packet scheduling in output-buffered optical switches with

limited-range wavelength conversion," accepted for publication in *IEEE/OSA Journal of LightWave Technology*.

[23] L. Liu and Y. Yang, "Achieving 100% throughput in input-buffered WDM optical packet interconnects," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, pp. 237-286, February 2011.

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

[25] L. Liu, Z. Zhang and Y. Yang, "Packet scheduling in a low-latency optical interconnect with electronic buffers, " *IEEE INFOCOM 2011*, Shanghai, China, April 2011.

[26] L. Liu, Z. Zhang and Y. Yang, "Packet scheduling in a low-latency optical switch with wavelength division multiplexing and electronic buffer, " *IEEE INFOCOM 2011*, Shanghai, China, April 2011.

[27] L. Liu and Y. Yang, "Energy-aware routing in hybrid optical network-on-chip for future multi-Processor system-on-chip, " *Proceedings of the 6th ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS'10)*, La Jolla, CA, October 2010.

[28] S. Shimizu, Y. Arakawa and N. Yamanaka, "Wavelength assignment scheme for WDM networks with limited-range wavelength converters," *Journal of Optical Networking*, vol. 5, issue 5, pp. 410-421, 2006.

[29] T. Tripathi and K.N. Sivarajan, "Computing approximate blocking probabilities in wave-

length routed all-optical networks with limited-range wavelength conversion," *IEEE Journal of Selected Areas in Comm.*, vol. 18, pp. 2123-2129, 2000.

[30] X. Qin and Y. Yang, "Nonblocking WDM switching networks with full and limited wavelength conversion," *IEEE Transactions on Communications*, vol. 50, no. 12, pp. 2032-2041, 2002.

[31] Z. Zhang and Y. Yang,"Optimal scheduling in buffered WDM interconnects with limited range wavelength conversion capability," *IEEE Transactions on Computers*, vol. 55, no. 1, pp. 71-82, Jan. 2006.

[32] R.K. Ahuja, T.L. Magnanti and J.B. Orlin, *Network Flows: Theory, Algorithms, and Applications,* Prentice Hall, 1993.

[33] T. Hou and A. Wong, "Queueing analysis for ATM switching of mixed continuous-bit-rate and bursty traffic," *Proc. INFOCOM'90*, pp. 660-667.

[34] S.L. Danielsen, C. Joergensen, B. Mikkelsen and K.E. Stubkjaer, "Analysis of a WDM packet switch with improved performance under bursty traffic conditions due to tunable wavelength converters," *Journal of Lightwave Technology*, vol. 16, no. 5, pp. 729-735, May 1998.

[35] Z. Zhang and Y. Yang, "Optimal scheduling in buffered WDM packet switching networks with arbitrary wavelength conversion capability," *IEEE Transactions on Computers,* vol. 55, no. 1, pp. 71-82, January 2006.

[36] L. Zuchelli, M. Burzio, and P. Gambini, "New solutions for optical packet delineation and synchronization in optical packet switched networks," *Proc. ECOC'96*, vol. 3, pp. 3.301-3.304, Oslo, Norway, 1996.

[37] L. Tassiulas and A. Ephremides, “Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks,” *IEEE Transactions on Automatic Control*, vol. 37, no. 12, pp. 1936-1948, 1992.

[38] J.G. Dai and B. Prabhakar, “The throughput of data switches with and without speedup,” *Proc. IEEE INFOCOM 2000*, pp. 556-564, Mar. 2000.

[39] A.D. Sarwate and V. Anantharam, “Exact emulation of a priority queue with a switch and delay lines,” *Queueing Systems: Theory and Applications*, vol. 53, pp. 115-125, Jul. 2006.

[40] H.-C. Chiu, C.-S. Chang, J. Cheng and D.-S. Lee, “Using a single switch with $O(M)$ inputs/outputs for the construction of an optical priority queue with $O(M_3)$ buffer,” *Proc. IEEE INFOCOM 2007*, pp. 2501-2505, May 2007.

[41] C.-S. Chang, Y.-T. Chen and D.-S. Lee, “Constructions of optical FIFO queues,” *IEEE/ACM Transactions on Networking*, vol. 14, pp. 2838-2843, 2006.

[42] 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.

[43] R. S. Tucker and W. Zhong,“Photonic packet switching: An overview,” *IEICE Transactions on Communications*, vol. E82-B, pp. 254-264, Feb. 1999.

[44] I. Chlamtac, A. Fumagalli and Chang-Jin Suh,“Multibuffer delay line architectures for efficient contention resolution in optical switching nodes” *IEEE Transactions on Communications*, vol. 48, no. 12, pp. 2089-2098, 2000.

[45] “http://www.fiber-span.com/pdf/Fiber-Span_FS_1123-PDL.pdf.”

[46] H. Le Minh, Z. Ghassemlooy, and W.P. Ng,"An ultrafast with high contrast ratio 1x2 all-optical switch based on tri-arm Mach-Zehnder employing All-optical flip-flop," *IEEE International Conference on Communications 2007 (ICC 2007)*, Glasgow, UK, pp. 2257-2262, Jun. 2007.

[47] S. Danielsen, B. Mikkelsen, C. Joergensen, T. Durhuus and K. Stubkjaer, "WDM packet switch architecture and analysis of the influence of turnable wavelength converters on the performance," *Journal of Lightwave Technology*, vol. 15, pp. 219-227, Feb. 1997.

[48] W.D. Zhong and R.S. Tucker, "Wavelength routing-based photonic packet buffers and their applications in photonic packet switching systems," *Journal of Lightwave Technology*, vol. 16, no. 10, pp. 1737-1745, Oct. 1998.

[49] L. Li, S.D. Scott and J.S. Deogun, "A novel fiber delay line buffering architecture for optical packet switching," *Proc. IEEE GLOBECOM 2003*, vol. 5, pp. 2809-2813, 2003.

[50] T. Zhang, K. Lu and J.P. Jue, "Shared fiber delay line buffers in asynchronous optical packet switches," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 4, pp. 118-127, April 2006.

[51] M. Garrett and W. Willinger, "Analysis, modeling, and generation of self-similar VBR video traffic," *Proc. SIGCOMM'94*, pp. 269-280.

[52] A. Huang and S. Knauer, "Starlite: A wideband digital switch," *Proc. GOLBECOM '84*, Nov. 1984.

[53] J.N. Giacopelli, J.J Hickey, W.S. Marcus, W.D. Sincoskie and M. Littlewood, "Sunshine: A high-performance self-routing broadband packet switch architecture," *IEEE Journal on Selected Areas in Communications*, vol. 9, no. 8, pp. 1289-1298, Oct. 1991.

[54] C. Develder, M. Pickavet and P. Demeester, “Assessment of packet loss for an optical packet router with recirculating buffer,” *Optical Network Design and Modeling (ONDM) 2002*, pp. 247-261, Torino, Italy, 2002.

[55] Z. Zhang and Y. Yang, “WDM optical interconnects with recirculating buffering and limited range wavelength conversion,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, no. 5, pp. 466-480, May 2006.

[56] C.S. Chang, D.S. Lee, and Y.S. Jou, “Load balanced Birkhoff-von Neumann switches, part I: one-stage buffering,” *Computer Communications*, vol. 25, pp. 611-622, Apr. 2002.

[57] I. Keslassy and N. McKeown, “Maintaining packet order in two-stage switches,” *IEEE INFOCOM ’02*, New York, June 2002.

[58] 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, Jan. 2008.

[59] E. Oki, R. Rojas-Cessa and H. Chao, “A pipeline-based approach for maximal-sized matching scheduling in input-buffered switches,” *IEEE Communication Letters*, vol. 5, pp. 263-265, Jun. 2001.

[60] C. Minkenberg, I. Iliadis and F. Abel, “Low-latency pipelined crossbar arbitration,” *Proc. IEEE GLOBECOM 2004*, vol. 2, pp. 1174-1179, 2004.

[61] D.K. Hunter, D. Cotter, R.B. Ahmad, W.D. Cornwell, T.H. Gilfedder, P.J. Legg, and I. Andonovic, “Buffered switch fabrics for traffic routing, merging, and shaping in photonic cell networks,” *Journal of Lightwave Technology*, vol. 15, pp. 86-101, Jan. 1997.

[62] 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.

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

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

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

[66] J.D. Owens, W.J. Dally, R. Ho, D.N. Jayasimha, S.W. Keckler and L.-S. Peh, "Research challenges for on-chip interconnection networks," *IEEE Micro*, vol. 27, no. 5, pp. 96-108, September 2007.

[67] W.J. Dally and B. Towles, "Route packets, not wires: on-chip interconnection networks," *DAC 2001*, pp. 684-689.

[68] A. Shacham, K. Bergman, L.P. Carloni, "Photonic networks-on-chip for future generations of chip multiprocessors," *IEEE Transactions on Computers*, vol. 57, no. 9, pp. 1246-1260, September 2008.

[69] J. Fujikata, K. Nishi, A. Gomyo, et al, "LSI on-chip optical interconnection with Si Nano-Opticals," *IEICE Transactions on Electronics*, vol. 91-C, no. 2, pp. 131-137, 2008.

[70] A. Driessen, D.H. Geuzebroek and E.J. Klein, "Optical network components based on microring resonators," *Proceedings. of the 8th International Conference on Transparent Optical Networks*, pp. 210-215, 2006.

[71] H. Gu, J. Xu and W. Zhang, “A low-power fat tree-based optical network-on-chip for multiprocessor system-on-chip,” *IEEE Computer Society Annual Symposium on VLSI*, pp. 19-24, May 2009.

[72] R. Ramaswami and K.N. Sivarajan, *Optical networks: a practical perspective,* second edition, Morgan Kaufmann, 2002.

[73] Q. Xu, B. Schmidt, S. Pradhan and M. Lipson, “Micrometre-scale silicon electro-optic modulator,” *Nature*, vol. 435, no. 7040, pp. 325-327, 2005.

[74] H. Gu, J. Xu and Z. Wang, “A novel optical mesh network-on-chip for gigascale systems-on-chip,” *IEEE Asia Pacific Conference on Circuits and Systems*, pp. 1728-1731, November 2008.

[75] G.-M. Chiu, “The odd-even turn model for adaptive routing,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 7, pp. 729-738, July 2000.

[76] C. Martínez, R. Beivide, E. Stafford, M. Moretó, and E.M. Gabidulin, “Modeling toroidal networks with the Gaussian integers,” *IEEE Transactions on Computers*, vol. 57, no. 8, pp. 1046-1056, August 2008.

[77] M. Flahive and B. Bose, “The topology of Gaussian and Eisenstein-Jacobi interconnection networks,” *IEEE Transactions on Parallel and Distributed Systems*, August 2009.

[78] R. Marculescu, U.Y. Ogras, L.-S. Peh, N.E. Jerger and Y. Hoskote, “Outstanding research problems in NoC design: system, microarchitecture, and circuit perspectives,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 1, pp. 3-21, January 2009.

[79] S. Koohi, S. Hessabi, “Contention-free on-chip routing of optical packets,” *Proceedings of the 3rd ACM/IEEE International Symposium on Networks-on-Chip*, pp. 134-143, 2009.

[80] C. Qiao and M. Yoo, “Optical burst switching (OBS) - a new paradigm for an optical Internet,” *Journal of High Speed Networks*, vol. 8, no. 1, pp. 69-84, 1999.

[81] S. J. Ben Yoo, “Optical packet and burst switching technologies for the future photonic internet,” *Journal of Lightwave Technology*, vol. 24, no. 12, December 2006.

[82] M. Yoo and C. Qiao, “A new optical burst switching protocol for supporting quality of service,” *SPIE Proceedings of Conference on All-optical Networking*, vol. 3531, pp.396-405, 1998.

[83] T.M. Pinkston and J. Shin, “Trends toward on-chip networked microsystems,” *International Journal of High Performance Computing and Networking*, vol. 3, no. 1, pp. 3-18, 2001.

[84] E. Nilsson, M. Millberg, J. Oberg and A. Jantsch, “Load distribution with the proximity congestion awareness in a network on chip,” *Proceedings of the conference on Design, Automation and Test in Europe*, pp. 1126-1127, March 2003.

[85] J. Hu and R. Marculescu, “DyAD - Smart routing for networks-on-chip,” *Proceedings of the 41st annual Design Automation Conference*, pp. 260-263, June 2004.