Pipelining Multicast Scheduling in All-Optical Packet Switches with Delay Guarantee

Zhiyang Guo and Yuanyuan Yang
Department of Electrical and Computer Engineering, Stony Brook University, Stony Brook, NY 11794, USA

***Abstract*—In this paper, we study multicast scheduling in all-optical packet switches. First, we propose a novel optical buffer called multicast-enabled Fiber-Delay-Lines (M-FDLs), which can provide flexible delay for copies of multicast packets using only a small number of FDL segments. We then present a Delay-Guaranteed Multicast Scheduling (DGMS) algorithm that considers the schedule of each arriving packet for multiple time slots. We also discuss some desirable features of DGMS in detail, such as guaranteed delay upper bound and adaptivity to transmission requirements. To relax the time constraint of DGMS, we further propose a pipelining technique that distributes the scheduling tasks among a sequence of sub-schedulers. The combinatorial logic circuit design of each sub-scheduler, which further reduces time complexity, is also provided. The performance of DGMS is tested extensively against statistical traffic models and real Internet traffic, and the results show that the proposed DGMS algorithm can achieve ultra-low average packet delay with minimum packet drop ratio.**

***Index Terms*—Optical packet switching, multicast scheduling, optical buffer, delay guaranteed, pipeline, hardware implementation.**

I. Introduction and Related Work

Optical networking has been widely adopted to transport high volume traffic in backbone networks due to the huge bandwidth of optics. All-optical packet switches are considered as a very appealing solution for high-throughput, energy-efficient and transparent forwarding in backbone networks. During the past few years, as driven by the increasing multicast applications requiring high-bandwidth transmission from one source to multiple destinations, such as video conference, video-on-demand (VoD) and IP-based Television (IPTV) [1], [2], optical multicast packet switching has attracted much research effort. A series of all-optical switching architectures and technologies have been proposed to support multicast at the switch/router level, such as wavelength-assisted switching [4], [13], Broadcast-and-Select (BS) switching [3], [9], etc. Despite of the considerable amount of work on multicast-capable optical packet switching architectures, however, relatively little attention has been paid on multicast scheduling in such switches, which is critical for high-speed all-optical packet switches. Motivated by this observation, in this paper we consider multicast scheduling in optical packet switches.

A major challenge for multicast scheduling in optical packet switches is the resolution of *output contention* [5], which occurs when multiple optical packets simultaneously go to the same output. In particular, the fact that multicast packets usually have more than one destination outputs intensifies output contention. Since a practical "optical RAM" able to mimic the buffers used in electronic switches is still not available currently, various contention resolution techniques have been proposed. Buffer-less approaches such as wavelength conversion and deflection routing [5] resolve contentions by sending conflicted packets to different wavelengths or other outputs. However, they have been shown to be ineffective for avoiding packet loss under congested network conditions and demand a lot of network resources. Electronic buffers have been considered to be used in OPS, creating "hybrid" electronic/optical switches [6], [7]. Yet such an approach requires Optical-to-Electronic-to-Optical (OEO) conversion, which leads to undesirable power consumption and additional cost in high speed switching. On the other hand, fiber-delay-lines (FDLs) [8], a passive device able to delay optical packets for a fixed period of time, provide a practical resolution for output contention in OPS, due to its transparency to traffic bit-rate and low power dissipation.

Multicast scheduling in optical packet switches with FDLs buffer is significantly different from the well-studied multicast scheduling in electronic switches [10], [11], for the reason that all the approaches for electronic switches rely on electronic RAM to resolve output contention, while FDLs can merely delay packets for a fixed period of time. Several multicast scheduling schemes using FDLs as buffer have been proposed [4], [12], [13]. Among them, a widely adopted scheme is output buffering [12], in which an output buffer consisting of an $N \times B$ switch and B FDL segments with fixed incremental delay is placed at each output. During each scheduling cycle, the scheduler assigns a FDL segment with proper delay in each output buffer to the packets destined for that output. To achieve a reasonable packet loss ratio and network link utilization, the output buffering scheme requires a large amount of FDL segments to construct output buffers. Since FDLs are bulky, such an approach is not scalable for large switches. Another commonly used multicast scheduling scheme is wavelength-assisted routing [4], [13], in which multicast packets are sent to multicast modules, a FDL loop device used to generate copies of the packet and provide necessary delay. However, wavelength-assisted routing can provide only limited multicasting ability and its performance deteriorates quickly under congested traffic conditions, as each multicast module cannot be shared by multiple packets simultaneously. Moreover, wavelength-assisted routing cannot provide delay guarantee since a packet may have to be recirculated for many times in the multicast module before being sent out.

In this paper, we propose a Delay-Guaranteed Multicast Scheduling (DGMS) algorithm for all-optical packet switches with FDLs buffer. In order to efficiently utilize FDLs, we also present a novel optical buffer called multicast-enabled FDLs (M-FDLs) that provides flexible delay for multicast packets by using only a small number of FDLs. The main features of the proposed DGMS can be summarized as follows.

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

978-0-9836283-0-9

This paper was peer reviewed by subject matter experts for publication in the Proceedings of ITC 2011

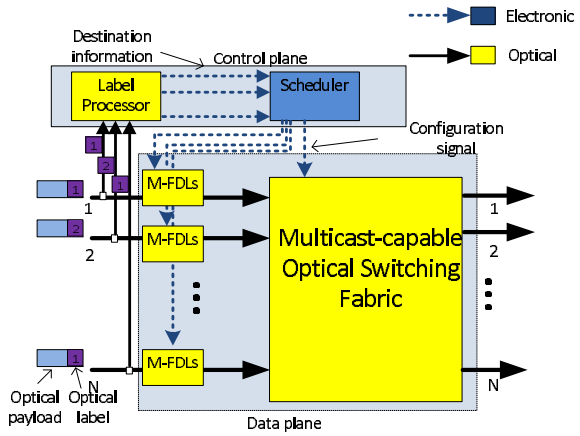

Fig. 1. The architecture of a single-wavelength, input-buffered $N \times N$ optical multicast packet switch.

- *Guaranteed delay upper bound for all transmitted packets.* By considering the schedule of each arriving packets for multiple time slots, DGMS allows more efficient packet transmission than scheduling algorithms that resolve output contention for a single time slot. When network is highly congested, DGMS can also immediately detect it and promptly drop packets with overlong delay to let upper layer protocols quickly respond to the network condition. Overall, we show that DGMS is able to guarantee a fixed delay upper bound for all transmitted packets regardless of traffic pattern, while keeping packet drops at a minimum level even under the most congested network condition. Such a feature is especially desirable in delay-sensitive multicast applications, such as video conferences and IPTV.
- *Enable pipelined scheduling and simple hardware implementation.* DGMS can be pipelined to reduce time complexity. In addition, certain procedures in DGMS can be implemented by simple combinatorial logic circuits to further relax the timing constraints.
- *Efficient buffer management.* Through extensive simulations, we demonstrate that with only a small number of FDLs, DGMS can achieve ultra-low average packet delay with minimal packet loss under various traffic conditions.

The reminder of the paper is organized as follows. Section II presents the architecture of the adopted optical multicast packet switch and optical buffer. Section III describes the details of the delay guaranteed multicast scheduling (DGMS). Section IV presents a pipelining technique and the corresponding hardware implementation that reduces the time complexity of DGMS. The performance of DGMS is evaluated through extensive simulations and the results are presented in Section V. Finally, Section VI concludes the paper.

II. Switch Architecture and Buffer Management

In this section, we briefly describe the adopted switch architecture and the operation of the proposed optical buffer called multicast-enabled FDLs (M-FDLS). We consider a simple single-wavelength, input-buffered optical multicast packet switch, the high level view of which is shown in Fig. 1. The adopted switch consists of optical multicast switching fabric

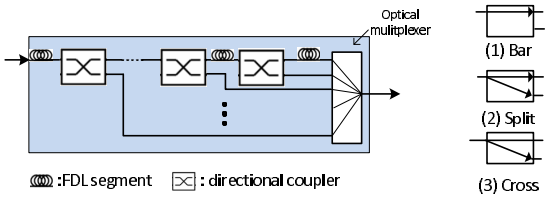

Fig. 2. Multicast-enabled FDLS (M-FDLs). Left: The structure of M-FDLs. Right: Three possible states of the directional coupler: bar, split and cross.

and M-FDLs as input buffers in the data-plane, and optical label processors and electronic scheduler in the control plane.

We assume the switch operates in a time-slotted manner and all optical packets have the same length. Each optical packet consists of two parts: payload and label (or header). The optical label contains the destination outputs of the packet, and is much shorter than the optical payload. When an optical packet arrives at an input, its label is stripped off and sent to the label processor, which can be performed passively by the optical correlation technique. The label processor then converts all optical headers to the electronic form, and sends them to the electronic scheduler, which calculates the schedule for each packet. Note that the optical label can adopt a lower bit rate than the optical payload to facilitate electronic scheduling [14]. Based on scheduling results, control signals are issued to the FDL buffers and switching fabric to properly configure the switch.

Next, we present a novel optical buffer called multicast-enable FDLs (M-FDLs) that provides flexible delay for each incoming multicast packet. Fig. 2 shows the buffer structure. The M-FDLs buffer consists of cascaded couplers and FDL segments, and each segment can provide a delay of T, the duration of a time slot. To provide flexible delays ranging from T to dT, a total of d FDL segments are needed. For each coupler, there are three states: when it is in "bar" state (the default state), packets simply go through it and move to the next FDL segment; when the state is "split," a copy of packet will be sent to the switch for transmission though an optical multiplexer, while the packet continues to move forward in the M-FDLs; when the state is "cross," packets will move out of the M-FDLs completely and be sent to the switch for transmission. The compensation for the power loss and noise incurred from optical splitting are not shown in the figure.

Let's use an example to illustrate how the M-FDLs operates. Assume that a multicast packet arrives at time slot t, and is scheduled to deliver a copy of it to some of its destination outputs in the $(t+i)^{th}$ time slot and to the rest of its destination outputs in the $(t+j)^{th}$ time slot ($i < j \leq d$). At the beginning of the $(t+i)^{th}$ time slot, the packet moves to the i^{th} coupler, and the scheduler sets the i^{th} coupler to "split," such that a copy of the packet will be sent to the switching fabrics. At the beginning of the $(t+j)^{th}$ time slot, the scheduler sets the j^{th} coupler to "cross" state, thus the packet moves out of the M-FDLS and be transmitted completely.

Compared with existing FDL buffers for multicast packet switching, M-FDLs has some clear advantages. On one hand, M-FDLs does not have the problem of limited multicast processing capability in the recirculating loop buffer in the wavelength-assisted routing scheme [4], [13], since it can be

shared by all the incoming multicast packets in a pipelined fashion. On the other hand, M-FDLs uses much less FDL segments compared to the output buffer used in the output buffering scheme [12]. For example, to achieve flexible delays ranging from $1T$ to $16T$, each output buffer in [12] requires $(1+2+\cdots+16=138)$ FDL segments, while each M-FDLs only requires 16 FDL segments, which is a substantial saving.

Based on the input-buffered optical multicast packet switch and M-FDLs buffer described above, we will present the delay-guaranteed multicast scheduling algorithm (DGMS) in the next section.

III. Delay Guaranteed Multicast Scheduling

A. Preliminaries

In this subsection, we introduce some commonly used terms in multicast scheduling. In multicast scheduling, the vector of destinations of a multicast packet is called its *fanout*. For clarity, an arriving *input packet* is usually distinguished from its corresponding *output copies*, i.e., the copies of the input packet destined for the outputs in its fanout.

The most straightforward multicast solution was the use of copy networks, in which all output copies are delivered by unicast. However, since optical switching fabrics such as Broadcast-and-Select (BS) has a *intrinsic multicasting capability*, i.e., the ability to transmit packets from one input port to multiple output ports simultaneously, treating multicast as multiple unicasts wastes bandwidth and prolongs packet delay. There are several service disciplines to transmit multicast packets from input ports to output ports, which can be roughly divided into two categories: *one shot* and *fanout splitting*. With one shot, all the output copies of an input multicast packet must be sent to the corresponding output ports in one time slot, whereas in fanout splitting, a multicast packet could be delivered to the outputs in multiple time slots, and in each time slot, only some of the outputs in its fanout receive the packet copy. It has been shown that one shot discipline may severely limit the throughput of the switch. Thus, in this paper, we adopt fan-out splitting discipline.

B. Delay Guaranteed Multicast Scheduling (DGMS)

In this subsection, we describe in detail the delay guaranteed multicast scheduling (DGMS) algorithm. Consider a switch of size $N \times N$ as shown in Fig.1, and assume that packets arrive at the beginning of each time slot.

According to the operations of the adopted switch, the label of an arriving packet will be sent to the scheduler to be processed the moment it enters the corresponding M-FDLs, and its copy can leave the M-FDLs and enter the switch as early as it reaches the first coupler (i.e., after a delay of T, where T is the time slot length). Therefore, the scheduler must determine the following within a time slot: (1) whether or not the packet is allowed to be switched in the next time slot; (2) if it is to enter the switch, which outputs will it be delivered to, such that no output contention could occur. Likewise, when the packet reaches the i^{th} coupler, both the corresponding coupler and the switching fabric should be properly configured to avoid output contention. Note that all arriving packets will be delayed for at least a period of T, which is the scheduling overhead.

DGMS considers the schedule for the next D time slots by keeping D scheduling vectors, indexed by $1, 2, \ldots, D$, with each vector corresponding to the scheduling results in a future time slot. Note that D cannot be larger than the maximum delay each M-FDLs can provide. For example, a scheduling vector of index i is denoted by S_i, which is used for keeping track of scheduling results of the time slot that is i time slots after the current time slot. S_1 is used to record scheduling results of the next time slot. A scheduling vector has N entries, indexed by $1, 2, \ldots, N$, with each corresponding to an output. The k^{th} entry of S_i is denoted by $S_i(k)$. If a copy of some packet for output k is scheduled to be transmitted in the i^{th} time slot after the current time slot, we say that output copy k of the packet is *assigned* to entry $S_i(k)$.

Each entry can be represented by a four-tuple $(full, input, location, split)$, where the one-bit field $full$ is set to 1 if this entry has been assigned, otherwise it is 0. $input$ is used to record the corresponding input index of the packet in that entry. $location$ is used to record the index of the scheduling vector that the copy is assigned to initially. For example, if a copy is assigned to the i^{th} scheduling vector, the $location$ field of its entry is set to i. $split$ is used to configure the state of couplers in M-FDLs. For example, if a packet is completely scheduled within next D time slots, and its last output copies are assigned to entries in the k^{th} scheduling vector, then the $split$ fields in all its entries in the k^{th} scheduling vector are set to 0 (indicating that the k^{th} coupler will be set to "cross" and the packet will exit the M-FDLs), while the $split$ fields in the rest of its assigned entries are set to 1 (indicating that a copy will be created while the packet stays in the M-FDLs).

Since the switching fabric can only run as fast as the input line rate, at most one packet can come out of the same M-FDLs in one time slot. Also, each output can receive at most one packet in each time slot to avoid output contention. Therefore, an entry of index i in a scheduling vector is said to be *eligible* for an output copy of some packet from input k if and only if all the following three conditions are met.

1) The entry is not full, i.e., no packet has been previously assigned to this entry.
2) The output copy is destined for the i^{th} output.
3) No packets from input k have been previously assigned to this scheduling vector, such that at most one packet from the same input is scheduled to be transmitted in the same time slot.

To ensure that the third condition is satisfied, we use D one-bit mask vectors of length N, each corresponding to one scheduling vector. The k^{th} entry of the i^{th} mask vector is denoted as $M_i(k)$, which is set to 1 if some packet from the k^{th} input has been assigned to the i^{th} scheduling vector. The scheduler will check mask vectors before assigning output copies to make sure no packet from the same input has been previously assigned to this scheduling vector, and all the copies with the same input index in each scheduling vector are copies of the same packet.

The basic operation of the scheduler is to find the *earliest possible* eligible entries for arriving packets in each time slot. For the reason of fairness, the Round-Robin scheduling

TABLE I
DELAY GUARANTEED MULTICAST SCHEDULING (DGMS)

Input to DGMS: Arriving packets P, scheduling vectors S, mask vectors M, priority register pr
// Packet Transmission
Configure couplers and switching fabric according to the 1_{st} scheduling vector S_1
For $i = 2$ to D **Do**
  $S_{i-1} = S_i$;
  $M_{i-1} = M_i$;
EndFor
Clear S_D, M_D;
// find the earliest eligible entries for the arriving packets
For packet P_i from input i,
$i = [pr, pr+1, \ldots, (pr+N-1) \mod N+1]$. **Do**
  For scheduling vector $S_k, k = [1, 2, \ldots, D]$ **Do**
    For each output O_j in P_i's fanout, **Do**
      If $S_k(j)$ is empty **and** $M_k(i) == 0$
        Assign the entry to the output copy of P_i;
      EndIf
    EndFor
    If some copies of P_i get assigned in S_k
      set $M_k(i) = 1$;
    EndIf
  EndFor
  If there are still outputs in P_i's fanout left undelivered
    Drop these output copies of P_i;
  EndIf
EndFor

scheme is used to allocate priority to the packets arriving at different inputs to be scheduled in each time slot. To indicate which input has the highest priority, a register pr is used. We update pr according to a cyclic-priority rule, i.e., the value of pr changes to $(pr+1) \mod N$ at the end of each time slot. The scheduler checks the optical label of each packet in the order of their input index $[pr, (pr+1) \mod N, \cdots, (pr+N-1) \mod N+1]$, and tries to assign their output copies to the earliest eligible entries among D scheduling vectors. Only when an output copy cannot be assigned after searching all D scheduling vectors will it be dropped by the scheduler. Note that some copies of the packets arriving later can be scheduled to be transmitted prior to copies of the earlier packets if there is no output contention, to reduce the average delay.

At the beginning of each time slot, the scheduler configures the corresponding couplers in the M-FDLs buffers and switching fabric for packet transmission, according to the first scheduling vector S_1. Next, the scheduler shifts all the scheduling vectors and mask vectors forward by one position, i.e., the content of S_i is moved to $S_{i-1}, i = 2, 3, \ldots, D$, and empties the last scheduling vector and mask vector, then starts the scheduling process for the current time slot. For example, assume that entry $S_1(2)$ has the value of $(1, 1, 3, 1)$, which indicates that the packet entered the M-FDLs three time slots ago from the 1_{st} input and now reaches the 3^{rd} coupler (because the scheduling vectors are shifted forward by one position each time slot). The scheduler sets the 3^{rd} coupler to "split," and connects input 1 with output 2 at the beginning of the time slot, such that a copy of the packet is delivered to output 2. The coupler will be reset to the default state "bar" after the packet goes through. The detailed description of DGMS is given in Table I.

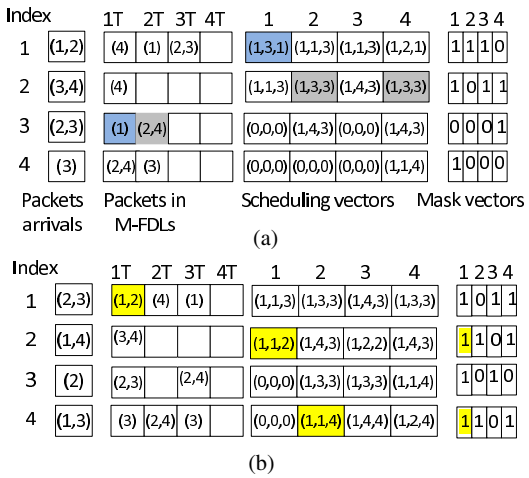

Fig. 3. A scheduling example for a 4×4 switch. (a) The initial switch state at the beginning of the current time slot. (b) The switch state at the beginning of the next time slot, obtained after the scheduler rotates the scheduling vectors and mask vectors, then schedules all arriving packets. The schedule for packet $(1, 2)$ is highlighted.

C. A Scheduling Example

A scheduling example of DGMS algorithm for a 4×4 switch is shown in Fig. 3. The number of scheduling vectors D is set to 4. Packets are denoted by their fanouts, e.g., the packet destined for outputs 3 and 4 is denoted as $(3, 4)$. The packets in M-FDLs are denoted by the time they have been delayed, and the maximum delay each M-FDLs provides is $4T$, where T is the length of a time slot. Entries in scheduling vectors shown in the figure are 3-tuple recording the $(full, input, location)$ information of the assigned packets. The $split$ field is omitted here, as it does not participate in the scheduling process. The initial content of scheduling vectors and mask vectors at the beginning of the current time slot is depicted in Fig. 3(a). Note that DGMS allows the packets arrived later to be transmitted before the packets arrived earlier, thus eliminates the Head-of-Line (HOL) blocking. For example, packet (1) from input 3 (blue block) arrived one time slot later than packet $(2, 4)$ (grey block), yet is scheduled to be transmitted earlier. Such a feature enables more efficient buffer management and reduces packet delay. However, it is worth mentioning that in-order transmission of packets from the same flow (i.e., packets sharing the same input and fanout) is guaranteed in DGMS, as it is impossible to deliver a later packet prior to its predecessor in the same flow in DGMS.

The scheduler then configures the switch and M-FDLs according to the first scheduling vector, and rotates the scheduling vector and mask vector forward by one position. Packets transmitted completely will be removed from the buffer, while those with remaining fanout stay in the M-FDLS and will be delayed by another T. Assume the current priority indicator pr is 1. The scheduling results of the arriving packets are shown in Fig. 3(b). Take packet $(1, 2)$ (yellow block) as an example. Its two output copies are scheduled for transmission in the 2^{nd} and 4^{th} scheduling vectors, respectively. The packet will reach the second coupler when the 2^{nd} scheduling vector is rotated to the front, and accordingly the scheduler will change the 2^{nd} coupler to "split" and connects input 1 with output 1, such that a copy of the packet will be delivered to output 2.

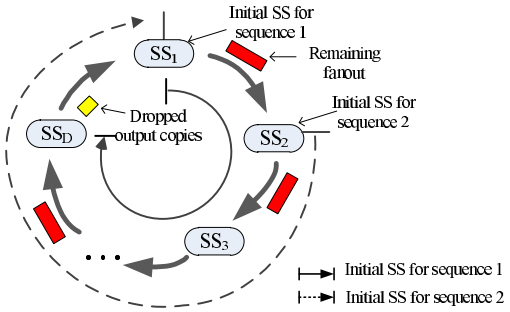

Fig. 4. Ring of cascaded schedulers. The solid line and dashed line indicate the sequence of sub-schedulers packets go through in different time slots.

The packet will move out of M-FDLs in four time slots when all its output copies are transmitted.

D. Discussions

We now discuss some useful properties and the time complexity of DGMS. One desirable feature of DGMS is that the maximum packet delay for any transmitted packets is bounded by D, where D is the number of scheduling vectors, regardless of traffic conditions. If output copies of some packets cannot be scheduled for the next D time slots, the scheduler can promptly drop them. As will be seen in the performance evaluation section, packet drop only occurs when network is highly congested and can be kept at a minimum level with a reasonable number of scheduling vectors D.

In addition, DGMS is highly adaptive to traffic conditions, meaning that the trade-off between the packet drop ratio and the average delay can be easily adjusted by changing the number of scheduling vectors D. If traffic is delay-sensitive, a smaller D can filter out packets with long delay and achieve ultra-low latency, whereas a larger D makes the scheduler more tolerant to packet delay and prone to reliable packet transmission.

DGMS can also provide fairness guarantee, since each arriving packet is assigned according to its input index in a round robin fashion. Finally, the number of FDL segments DGMS requires is much smaller compared to existing multicast scheduling algorithms for OPS.

The time complexity of DGMS is $O(N^2D)$ in the worst case, where N is the switch size and D is the number of scheduling vectors. Next we will introduce a pipelining technique to further reduce the complexity to $O(N)$.

IV. Pipelining the Scheduling

In this section, we present a pipelining technique that distributes scheduling tasks to a sequence of sub-schedulers, and show that this technique can reduce the time complexity of DGMS to $O(N)$.

The most time consuming part in DGMS involves a nested loop of three layers, when the scheduler tries to find the earliest eligible entries among D scheduling vectors for at most N arriving packets, each with a fanout of cardinality up to N. Also, it takes $O(ND)$ time to shift all scheduling vectors. To schedule packets among D scheduling vectors, we construct D sub-schedulers (SS), indexed by $1, 2, \ldots, D$, and concatenate them to a directional cascaded ring, as shown in Fig. 4.

Each $SS_i, i \in [1, 2, \ldots, D]$, has a built-in scheduling vector S_i and mask vector M_i, and takes the input index and the remaining fanout of the processed packet as inputs. The processing component ($COMP$) inside each SS is responsible for the scheduling of the processed packet according to the built-in scheduling vector and mask vector. Each SS also passes the input index and the remaining fanout of the processed packet as outputs to the next SS. In each time slot, the arriving packets are processed through a sequence of sub-schedulers along the ring in a pipelined fashion, till all the output copies of the arriving packets are either scheduled or dropped. We denote the first sub-scheduler in the sequence as the *initial* SS. Starting from the initial SS, the i^{th} SS in the sequence is responsible for the scheduling for the time slot which is i time slots after the current time slot.

In DGMS, all the scheduling vectors and mask vectors need to be shifted forward by one position at the beginning of each time slot, which involves massive data transfer among the vectors. To simplifies the operation, we clear the initial SS then "rotate" clockwise by one position at the beginning of every time slot, that is, choose the one next to the initial SS as the initial SS in the next time slot. For example, as depicted in Fig. 4, assume SS_1 is the initial SS for the scheduling sequence in the current time slot, and the arriving packets go through the sequence of SS's along the solid line. In the next time slot, SS_2 is chosen as the initial SS. In this way, SS_1 becomes the last SS in the sequence and all other SS's are one position closer to the initial SS, indicated by the dashed line in Fig. 4. The simple rotation avoids massive data transfer and only takes $O(1)$ time, while producing equivalent results to that of shifting all the vectors forward.

To relax the time constraints of DGMS, all SS's operate in a pipelined fashion. Fig. 5 illustrates the pipeline example for a 4×4 optical multicast switch with $D = 4$ scheduling vectors, in which the time for each SS to process a packet is denoted as a *microslot*. Assume SS_1 is chosen as the initial sub-scheduler in the first time slot. The packets arriving from input i in the k^{th} time slot is denoted as P_i^k. In the first time slot, SS_1 starts to process packet P_1^1. When it finishes, P_1^1 is passed to SS_2 while the packets from input 2 (P_2^1) is fed to SS_1. As shown in Fig. 5, the scheduling of all the packets arriving in the first time slot P_i^1 can be completed in $7 (= N + D - 1)$ microslots. However, due to the fact that only the result of the initial SS needs to be ready for transmission at the beginning of the next time slot, we do not have to wait till all the scheduling for the first time slot completes before starting the scheduling for the second time slot. In other words, the scheduling in consecutive time slots can also be pipelined. As shown in Fig. 5, it takes four microslots for SS_1 to finish the scheduling for packets arriving in the first time slot. For packets arriving in the 2^{nd} time slot, P_i^2, the initial SS is set to SS_2 after rotation, and the scheduling can begin as early as the 6^{th} microslot (yellow block) rather than the 8^{th} microslot (grey block). Therefore, it only takes $O(N)$, instead of $O(N+D)$, microslots to complete the scheduling process in each time slot with the pipelining technique.

Each sub-scheduler has to find the eligible entry for the processed packet in each microslot, which is a non-trivial task. Therefore, we present a simple combinatorial circuit design for the processing component (COMP), and show that it only

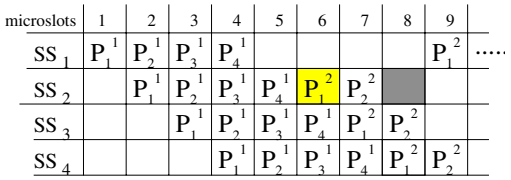

Fig. 5. Example of the pipeline operation. The microslot when the scheduling for the 2^{nd} time slot can begin without consecutive time slot pipelining is marked in grey, while that with consecutive time slot pipelining is marked in yellow.

takes $O(1)$ time for each SS to complete the scheduling in one microslot. The inputs of COMP include following.

1) The destination vector of the processed packet, denoted by $O(i), i = 1, 2, \ldots, N$, where $O(i) = 1$ if output i is in the fanout of the packet;
2) The $full$ bit of the entries in the scheduling vector, denoted by $F(i)$, where $F(i) = 1$ if the i^{th} entry of the scheduling vector is occupied;
3) The mask bit from the mask vector, denoted by $M(i)$, where i is the receiver index of the processed packet. $M(i) = 1$ if the packet from the same input of the processed packet has been assigned to the scheduling vector, otherwise it is 0.

The outputs of COMP include following.

1) $R(i)$, the remaining outputs in the fanout that cannot be scheduled, which is passed to the next SS, where $i = 1, 2, \ldots, N$;
2) $Q(i)$, the scheduling results of the destination outputs of packets, where $i = 1, 2, \ldots, N$ and $Q_i = 1$ if the i^{th} output copy is successfully scheduled in the current SS;
3) $M'(i)$, the updated mask vector after the scheduling, where i is the receiver index of the processed packet.

The logic expressions for R_i, Q_i and M'_i are given below.

$$\begin{aligned} R(i) &= O(i) \cap (M(i) \cup F(i)) \quad \forall i = 1, 2, \ldots, N \\ Q(i) &= \overline{M(i)} \cap \overline{F(i)} \cap O_i \quad \forall i = 1, 2, \ldots, N \\ M'(i) &= M(i) \cup Q(1) \cup Q(2) \cup \cdots \cup Q(N) \end{aligned}$$

From the above logic expressions, we can see that each SS can complete the scheduling in $O(1)$ time with $O(N)$ hardware cost (the number of logic gates used). Overall, with the pipelined scheduling and combinatorial circuit design, the time complexity of DGMS can be reduced to $O(N)$.

Note that since arriving packets need to be processed sequentially, each sub-scheduler needs to compute the schedule N times faster than the line rate, which may not be scalable in high speed switching. Therefore, we have also developed a parallel scheduling technique, which further distributes the scheduling to a set of processors running in parallel. This way, each processor only takes $O(1)$ time to perform its task. Due to limited space, we omit the parallel scheduling in this paper.

V. Performance Evaluations

We have conducted extensive simulations to evaluate the performance of DGMS. In this section, we present the simulation results. The simulation consists of two parts. In the first part of the simulation, we evaluate the effect of the number of scheduling vectors D (i.e., the maximum delay) on the packet drop ratio, which is defined as the percentage of dropped output copies among the total output copies of all packets arrived during the simulation period. In the second part, we evaluate the average delay performance of the proposed DGMS algorithm, which is calculated by the average interval between the arrival and departure of all successfully transmitted output copies.

As mentioned earlier, there has been little previous work on the multicast scheduling on optical packet switches with FDL input buffer. On the other hand, there is some similarity between the adopted switching architecture and the input queued (IQ) electronic switch. Therefore, we compare the performance of DGMS with several well-known multicast scheduling algorithms for IQ electronic switches, including FIFOMS [10] and MCMS [11] algorithms. The results for FIFO scheduling on the output queued multicast switch (OQ-FIFO) are also presented as a performance benchmark. These algorithms can be briefly described below.

- **FIFOMS** is an iterative multicast scheduling algorithm. In an iteration, each unmatched input scheduler selects the HOL packet in each VOQ with the smallest time stamp and sends the requests to the corresponding outputs. The process continues till there is no possible match between inputs and outputs. FIFOMS was shown to be superior to many well-known scheduling algorithms in terms of packet delay.
- **MCMS** considers the scheduling of the HOL packet in each input queue for multiple time slots (the number of time slots considered is set to 64 in our simulation). It was demonstrated that the delay performance of MCMS outperforms most of previous scheduling algorithms such as WSPLIT, Revision scheme and Windows-based algorithms.
- The output queued switch is known to be superior to the input queued switch in terms of performance, but requires N times faster switching ability. Despite its much stronger hardware requirement, in our simulation, we include a simple FIFO scheduling algorithm on the output queued switch (**OQFIFO**) as a performance benchmark to show how close our algorithm can be to the performance of the output queued switch while not requiring the speed-up.

In each simulation run, there is a sufficient warmup period (typically one fourth of the total simulation time) to obtain stable statistics. The simulation runs for a fixed amount of simulation time (10^6) unless the scheduling algorithms become unstable (i.e., the switch reaches a stage where it cannot sustain the offered load). We have simulated DGMS for different switch sizes under both statistical traffic models, such as Bernoulli traffic, mixed traffic, bursty traffic. etc., and real Internet traffic traces. Due to limited space, we only give some representative results under the mixed traffic model and the real Internet traffic traces in the paper.

A. Performance Evaluation under Mixed Traffic

Internet traffic is a mixture of unicast and multicast packets. In this subsection, we show that DGMS is capable of dealing with both traffic types efficiently. In the mixed traffic pattern, arriving packets can be either unicast or multicast. Packets

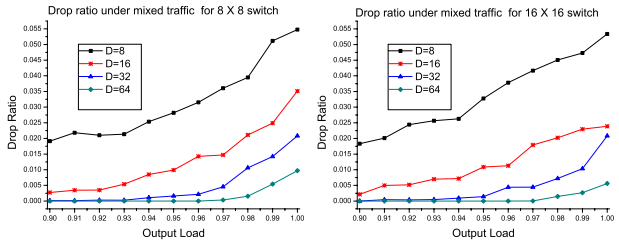

Fig. 6. The effect of the number of scheduling vectors D on the packet drop ratio under mixed traffic (half the traffic is multicast). (a) 8×8 switch; (b) 16×16 switch.

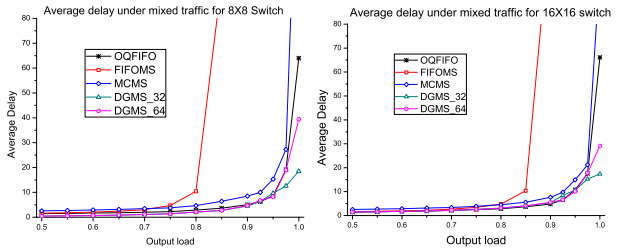

Fig. 7. The average delay under mixed traffic (half the traffic is multicast). (a) 8×8 switch; (b) 16×16 switch.

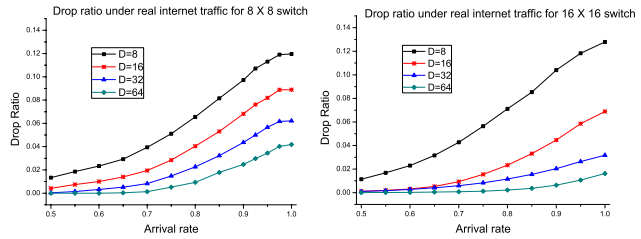

Fig. 8. The effect of the number of scheduling vectors D on the packets drop ratio under real Internet traffic. (a) 8×8 switch; (b) 16×16 switch.

arrive at an arrival rate λ, which is the probability that there is a packet arrival in a time slot. The traffic is a combination of unicast and multicast packets, with a multicast fraction (f_m) and a unicast fraction (f_u), where $f_m = 1 - f_u$. For a unicast packet, it has an equal probability ($1/N$) being destined for each output port, while for a multicast packet, we assume it has a probability of b being destined to each output port. When $f_u = 1$, the traffic only consists of unicast packets. The relationship between output load μ and arrival rate λ can be expressed as $\mu = \lambda(f_u + b \times N \times f_m)$. In our simulation, the multicast traffic fraction is set to 0.5, i.e., half of the traffic is multicast.

The effect of the number of scheduling vectors D (maximum delay) on the packet drop ratio under mixed traffic for both 8×8 and 16×16 switches is illustrated in Fig. 6. We can see that for switches of both sizes, packet drop only occurs under high traffic load (over 0.9) with only $D = 16$ scheduling vectors. The packet drop ratio decreases drastically when we increase D from 8 to 32, but only a slight improvement is observed when we further increase D to 64, indicating that most packets can be scheduled for transmission within 32 time slots. Even under the most extreme traffic condition when the output load is 1, i.e., one packet is destined for each output in every time slot on average, 32 scheduling vectors is sufficient to keep the packet drop ratio at a very low level.

Fig. 7 compares the average delay of DGMS under mixed traffic with other algorithms for both 8×8 and 16×16 switches. To demonstrate the effect of the number of scheduling vectors D on the average delay, we set the value of D to 32 and 64, and label them as DGMS_32 and DGMS_64 in the figure, respectively. It is shown that both MCMS and FIFOMS are saturated before the traffic load reaches 1, which coincides with the theory that the IQ switch cannot maintain sustainability under all admissible multicast traffic conditions [17]. At the same time, we also find the proposed DGMS algorithm closely matches the performance of OQFIFO and outperforms both MCMS and FIFOMS when traffic load is moderate. When the traffic load approximates 1, DGMS even achieves better performance than OQFIFO. The reason is that DGMS can detect and promptly drop output copies with overlong delay instead of keeping them in the buffer, thus significantly reduces the average packet delay.

We can see that DGMS_32 generally performs better than DGMS_64 in terms of average packet delay. The reason is that more scheduling vectors mean that the scheduler is less prone to drop packets and more tolerant to packet latency. For example, an output copy of some packet expected to have a delay of 40 time slots would be dropped by DGMS_32, while it would be scheduled for transmission by DGMS_64. Such trade-off between the packet drop ratio and the average delay can be easily adjusted by changing the number of scheduling vectors, making DGMS highly adaptive to various transmission requirements.

B. *Performance Evaluation under Real Internet Traffic*

Due to the complexity of Internet traffic, it is very difficult, if not impossible, to completely capture its characteristics using statistical traffic models. For this reason, we have also tested the proposed DGMS algorithm under real Internet traffic traces obtained from the backbone network link monitors.

The anonymized traffic traces used here were obtained from the CAIDA's passive $OC192$ network link monitors [15]. All trace files consist of one line per IP packet arrival in the form of *<packets_ index, time_stamp, protocol, source_IP_address, destination_IP_address>*. As in [16], we feed each input by a separate trace file. We assume all packets have a fixed size. Note that due to the lack of traffic regulation, certain outputs are busier than others in real Internet traffic and it is impossible to determine the output load. Therefore, different from the simulation under statistical traffic models, where the traffic is admissible (no oversubscription at outputs) and the scheduling algorithms are evaluated against the output traffic load, we test the algorithms against the packet arrival rate at the inputs in real Internet traffic. To adjust the arrival rate, we change the length of the time slot according to the throughput of the trace, and multiple packets from the same input arriving in the same time slot are placed in consecutive time slots. For example, if the throughput of a trace is 500 Mb/s and all packets have a fixed size of 64 bytes, to achieve the arrival rate of 0.8, the time slot length is $((64 \times 8)/500M) \times 0.8 = 0.8192\mu$s.

With the absence of the forwarding table, determining how to map the destination IP address of packets to output ports of the switch is not a trivial task. Since forwarding tables in the routers are updated relatively infrequently, we can assume that it stays invariable during the simulation period. For unicast

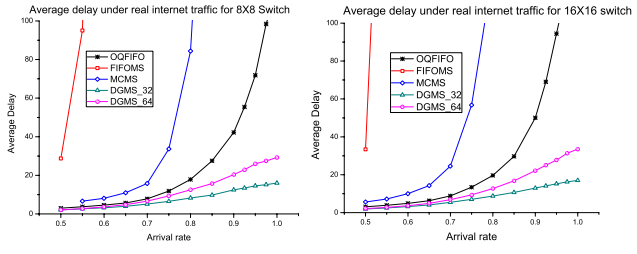

Fig. 9. The average delay under real Internet traffic. (a) 8×8 switch; (b) 16×16 switch.

packets, we use a simple hash function to determine the output port for each packet, which returns the modulus of summation of the four IP address fields in each destination IP to the switch size N. For example, a packet with destination IP address 243.124.121.4 will be sent to port $(243 + 124 + 121 + 4) \mod N + 1$. For multicast packets, we uniformly choose from 1 to N output ports with an equal probability. In this way, all the packets belonging to the same flow (i.e., packets share the same source and destination IP) will be sent to the same destination output port(s).

From Fig. 8, we can see that there is a noticeable increase in the packet drop ratio. The reason is three-fold. First, we use the packet arrival rate at the input instead of the output load in this simulation. As real Internet traffic consists of both unicast flows and multicast flows, the output load is considerably higher than the arrival rate. Second, the real Internet traffic consists of many flows, in which packets arrive in consecutive time slots and share the same output ports. Such traffic bursts are more likely to cause packet drop. Finally, real Internet traffic could be inadmissible during the simulation period, as some output ports could be oversubscribed. Nevertheless, $D = 64$ scheduling vectors are sufficient to keep the packet drop ratio below 1% in the most congested traffic condition when the arrival rate is 1.

As for average packet delay shown in Fig. 9, both FIFOMS and MCMS perform considerably worse under real Internet traffic than under the mixed traffic model. FIFOMS saturates before the packet arrival rate reaches 0.55 and MCMS saturates at the arrival rate of 0.8. Meanwhile, we can see that DGMS manages to achieve ultra-low average delay and exceeds the performance of all other algorithms under all traffic loads, for the same reason as that under mixed traffic.

VI. Conclusions

In this paper, we have studied multicast scheduling problem for input-buffered optical multicast switches. We first proposed an efficient optical buffer called multicast-enabled FDLs (M-FDLs), which can provide flexible delay for output copies of each multicast packet while requiring only a small number of FDL segments. We also designed a delay-guaranteed multicast scheduling (DGMS) algorithm, the main features of which can be summarized as follows.

- Guaranteed delay upper bound for transmitted packets. DGMS can guarantee a fixed delay upper bound for all transmitted packets, while keeping packet drop ratio at a minimum level even under the most congested traffic condition.
- Ultra-low average delay. DGMS can detect and promptly drop output copies with long delay, thus can significantly lower the average delay.
- Adaptive to varying transmission requirements. DGMS can easily adjust the number of scheduling vectors D to balance the trade-off between the packet drop ratio and the packet delay.
- Require very few number of FDLs. compared to existing multicast scheduling schemes for all optical packet switches, DGMS can achieve superior performance using much less FDLs.
- Fairness. All packet arrivals are scheduled in a round robin fashion to ensure fairness.

We also presented a pipelining technique and combinatorial digital circuit design for DGMS, which lowers the time complexity to $O(N)$. Extensive simulations demonstrate that DGMS achieves ultra-low average packet delay under both statistical traffic models and real Internet traffic with a minimal packet drop ratio.

References

[1] Z. Guo, X. Luo, Y. Jin, et al, "Improving Resource Utilization in Hybrid Packet/Circuit Multicasting for IPTV Delivery," *OFC 2008,* 2008.

[2] J. Choi, M. Yoo, and B. Mukherjee, "Efficient Video-on-Demand Streaming for Broadband Access Networks," *IEEE/OSA Journal of Optical Communications and Networking,* vol. 2, no. 1, pp. 38-50, January 2010.

[3] A. Misawa and M. Tsukada, "Broadcast-and-Select Photonic ATM Switch with Frequency Division Multiplexed Output Buffers," *Journal of Lightwave Technology,* vol. 1 pp.1769-1777, 1997.

[4] Q. Huang, S. Member, W. Zhong and S. Member, "A Wavelength-Routed Multicast Packet Switch With a Shared-FDL Buffer," *Journal of Lightwave Technology,* vol. 28, pp. 2822-2829, October 2010.

[5] A.G.P. Rahbar and O.W.W. Yang, "Contention Avoidance and Resolution Schemes in Bufferless All-Optical Packet-Switched Networks: a Survey," *IEEE Comm. Surveys & Tutorials*, vol. 10, no. 4, pp. 94-107, 2008.

[6] L. Liu and Y. Yang, "Packet Scheduling in a Low-Latency Optical Switch with Wavelength Division Multiplexing and Electronic Buffer" *IEEE INFOCOM 2011*.

[7] A.G. Reza and H. Lim, "Hybrid Buffering Architecture for Packet Contention Resolution of an Optical Packet Switch," *International Journal for Light and Electronic Optics,* pp. 3-5, July 2010.

[8] H. Yang and S.J.B. Yoo, "All-Optical Variable Buffering Strategies and Switch Fabric Architectures for Future All-Optical Data Routers," *Journal of Lightwave Technology,* vol. 23, pp. 3321-3330, Oct. 2005.

[9] Y.K. Yeo, Z. Xu, D. Wang, J. Liu, Y. Wang and T. Cheng, "High-Speed Optical Switch Fabrics with Large Port Count," *Optics Express,* vol. 17, no. 13, pp. 10990-10997, 2009.

[10] D. Pan and Y. Yang, "FIFO-Based Multicast Scheduling Algorithm for Virtual Output Queued Packet Switches," *IEEE Transactions on Computers,* vol. 54, no. 10, pp. 1283- 1297, Oct. 2005.

[11] W.T. Chen, C.F. Huang, Y.L. Chang and W.Y. Hwang, "An Efficient Cell-Scheduling Algorithm for Multicast ATM Switching Systems," *IEEE/ACM Transactions on Networking,* vol.8, no.4, pp. 517-525, 2000.

[12] H. Harai and M. Murata, "High-Speed Buffer Management for 40 Gb/s-Based Photonic Packet Switches," *IEEE/ACM Transactions on Networking,* vol. 14, pp. 191-204, Feb. 2006.

[13] Q. Huang and W. D. Zhong, "An Optical Wavelength-Routed Multicast Packet Switch Based on Multitimeslot Multiwavelength Conversion," *IEEE Photonic Technology Letter*, vol. 20, no. 18, pp. 1518-1520, 2008.

[14] L.G. Rau and D.J. Blumenthal, "160 Gb/s Variable Length Packet 10 Gb/s-Label All-Optical Label Switching with Wavelength Conversion and Unicast/Multicast Operation," *Journal of Lightwave Technology,* vol. 23, pp. 211-218, Jan. 2005.

[15] K. Claffy, D. Andersen and P. Hick, "The CAIDA Anonymized 2010 Internet Traces," http://www.caida.org/data/passive/passive_2010_dataset.xml

[16] A. Kos, P. Homan and J. Bester " Performance Evaluation of a Synchronous Bulk Paket Switch Under Real Traffic Conditions" *IEICE Transactions on Communications*, vol. E86-B, May 2003.

[17] M.A. Marsan, A. Bianco, P. Giaccone, E. Leonardi and F. Neri, "Multicast Traffic in Input-Queued Switches: Optimal Scheduling and Maximum Throughput," *IEEE/ACM Transactions on Networking*, vol. 11, no. 3, pp. 465-477, June 2003.