Low-Latency Multicast Scheduling in All-Optical Interconnects

Zhiyang Guo and Yuanyuan Yang

***Abstract*—Optical interconnects are considered as a very appealing solution for future high speed interconnections in core networks and parallel computers. In this paper, we study multicast scheduling in all-optical packet interconnects/switches. We first 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 *Low Latency Multicast Scheduling (LLMS) Algorithm* that considers the schedule of each arriving packet for multiple time slots. We show that LLMS has several desirable features, such as a guaranteed delay upper bound and adaptivity to transmission requirements. To relax the time constraint of LLMS, we further propose a pipeline and parallel architecture for LLMS that distributes the scheduling task to multiple pipelined processing stages, with N processing modules in each stage, where N is the size of the interconnect. Finally, by implementing it with simple combination circuits, we show that each processing module can complete the packet scheduling for a time slot in $O(1)$ time. The performance of LLMS is evaluated extensively against statistical traffic models and real Internet traffic traces, and the results show that the proposed LLMS algorithm can achieve superior performance in terms of average packet delay and packet drop ratio.**

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

I. Introduction and Related Work

OPTICAL interconnects are considered as a very appealing solution for high-throughput, energy-efficient and transparent packet forwarding in core networks and parallel computers. During the past few years, driven by the increasing multicast applications requiring high-bandwidth transmission from one source to multiple destinations, such as video conference, video-on-demand (VoD) [1] and IP-based Television (IPTV) [2], optical multicast switching has attracted much research effort. A series of all-optical switching architectures and techniques have been proposed to support multicast at the interconnect/router level, such as wavelength-assisted switching [3], [4], Broadcast-and-Select (BS) switching [5], [6], and wavelength-division-multiplexing (WDM) switching [7]–[9], etc. However, despite the considerable amount of work on multicast-capable optical packet switching architectures, relatively little attention has been paid to multicast packet scheduling in such interconnects, which is critical for high-speed all-optical packet interconnects. Motivated by this observation, in this paper we study multicast scheduling in all-optical packet interconnects/switches.

Manuscript received June 9, 2013; revised September 15 and December 30, 2013. The editor coordinating the review of this paper and approving it for publication was C. Assi.

The authors are with the Department of Electrical and Computer Engineering, Stony Brook University, Stony Brook, NY 11794, USA (e-mail: {zhiyang.guo, yuanyuan.yang}@stonybrook.edu).

Digital Object Identifier 10.1109/TCOMM.2014.021614.130424

Since a practical "optical RAM" able to mimic the buffers used in electronic interconnects is still not available currently, how to resolve *output contention*, which occurs when multiple optical packets simultaneously go to the same output, poses a serious challenge for multicast scheduling in optical packet interconnects. Various contention resolution techniques have been proposed [10]–[16]. Bufferless approaches such as wavelength conversion and deflection routing [10], [11] resolve contentions by sending conflicted packets to different wavelengths or other outputs. However, these approaches have been found ineffective for avoiding packet loss under congested network conditions and demanding a lot of network resources. Electronic buffers have been used to create "hybrid" electronic/optical interconnects [12]–[15]. Yet such an approach requires optical-to-electronic-to-optical (OEO) conversion, which leads to undesirable packet delay, power consumption and additional cost in high speed switching. On the other hand, fiber-delay-line (FDL) [16] provides a viable solution to store optical packets due to its transparency to traffic bit-rate and low power dissipation.

Multicast scheduling in optical packet interconnects with FDL buffer is fundamentally different from the well-studied multicast scheduling in electronic interconnects [17]–[20], for the reason that all the approaches for electronic interconnects rely on electronic RAM to resolve output contention, while FDLs can merely delay packets for a fixed period of time. Two major challenges must be properly addressed in multicast scheduling with FDL buffer. First, since FDL is very bulky in general, only a few can be used in a single optical interconnect. Therefore, efficient optical buffer is needed to avoid the performance degradation resulted from limited buffering capability of FDL. Second, scheduling in optical packet interconnects requires electronic processing that involves calculating the delay for each packet in the FDL buffer and configuring the switching fabric, where a complex scheduler may pose a bottleneck in high-speed switching. For example, a scheduler with $O(N)$ time complexity, where N is the interconnect size, requires the electronic processing component to run N times faster than the port speed normalized by the packet length on the fiber. Suppose we have an optical packet switch with 100 Gbit/s port speed and the packet length is 64 bytes. Then, the number of packets arriving at each input port would peak at around 200 million per second (i.e., each time slot lasts around 5 ns). To process packets at such a rate, a scheduler with $O(N)$ time complexity would have to work at a clock frequency much higher than state-of-art FPGAs can accommodate even for small switches with $N = 8$ ports. This means that we will face a serious scalability problem as the port count and port speed increase. It is therefore critical to design a scheduler of lower time complexity for high speed optical packet interconnects.

Most existing multicast scheduling schemes require $O(N)$ time complexity. Wavelength-assisted routing [3], [4] is a

commonly used multicast scheduling scheme, in which each multicast packet is sent to a multicast module, a FDL loop device used to generate copies of the packet and provide necessary delay. However, wavelength-assisted routing based schemes are generally quite complex and provide only limited multicasting ability, 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 many times in the multicast module before being sent out. Output buffering [22], [23] is another widely adopted scheme. In [22], a buffer consisting of an $N \times B$ switch and B FDL segments is placed at each output of an $N \times N$ interconnect. The length of these FDL segments increases linearly, in which the shortest segment is able to delay optical signal for one time slot and the longest one can delay optical signal for B time slots. During each time slot, the scheduler assigns packets to FDL segments with proper delay in each output buffer, such that they will exit the optical buffer at different times. A shared segment buffer was proposed in [23], which significantly reduces the total length of FDL segments in the output buffering scheme. However, it demands a very large switching fabric. Though output buffering delivers optimal performance in terms of average delay and packet drop ratio, the stringent hardware requirement makes output buffering unscalable for large interconnects. In [14], a multicast scheduling algorithm called GLMS was proposed for hybrid optical/electronic interconnects that uses electronic buffers to store packets that fail output contention. The algorithm cannot be adopted for all-optical interconnects that rely on FDLs for output contention resolution.

To avoid performance bottleneck caused by slow scheduling in high speed electronic and optical switches, many previous works focused on designing scheduler architectures with low time complexity through parallel and/or pipeline processing [24]–[26]. For example, [24] presents a multi-processing scheduler with $O(N)$ time complexity for input-queued electronic switches, which uses multi-input-queue (MIQ) and parallel arbitration to speedup the scheduling process. A scheduler for output-queued switches was proposed in [25], which achieves $O(\log^2 N)$ time complexity using a parallel prefix-sum operation. Its time complexity was further reduced to $O(1)$ in [22], [26] through a pipeline processing architecture consisting of $(\log_2 N + 1)$ pipeline stages. However, the pipeline processing incurs $O(\log_2 N)$ delay overhead.

In this paper, we systematically address the above challenges in multicast scheduling for high speed optical packet interconnects. Our contributions can be summarized as follows.

- We propose an efficient optical buffer called multicast-enabled FDLs (M-FDLs) that enables flexible packet duplication and controllable delay using much shorter FDL segments than the incremental buffer [22]. Such optical buffer is very helpful to multicast scheduling in optical packet interconnects.
- Using the M-FDLs buffer, we present an algorithm called *Low Latency Multicast Scheduling (LLMS) Algorithm* for all-optical packet interconnects. By considering the schedule of each arriving packet for multiple time slots, LLMS allows more efficient packet transmission than scheduling algorithms that resolve output contention for a single time slot. LLMS can also immediately detect the congestion and promptly drop packets with overlong delay to let upper layer protocols quickly respond to the network condition. Such a feature is desirable for delay-sensitive multicast applications. LLMS is able to achieve the performance very close to the optimal performance of output buffering [22] in terms of average delay and packet drop, while requiring orders of magnitude shorter FDL segments.
- We propose a pipeline and parallel processing architecture that distributes the scheduling task to multiple pipelined stages, with N processing modules operating in parallel in each stage. Combined with a simple combination circuit implementation, the time complexity for each processing module can be reduced to $O(1)$. The proposed architecture enables the switch to schedule packets at the line rate, and solves the dilemma that the processing speed of electronic scheduler struggles to catch up with the ever-increasing port speed in optical packet interconnects. Also, it does *not* incur additional pipeline overhead, which is very desirable compared to the scheduler proposed in [22], [26] that has $O(\log_2 N)$ pipeline overhead.
- The LLMS algorithm can be easily extended to provide differentiated Quality-of-Service (QoS), which presents a desirable extra feature. We show that the prioritized LLMS scheduling, though based on a simple preemptive strategy, achieves good QoS differentiation in traffic and, more importantly, can be implemented by the pipeline and parallel processing architecture with very little modification.

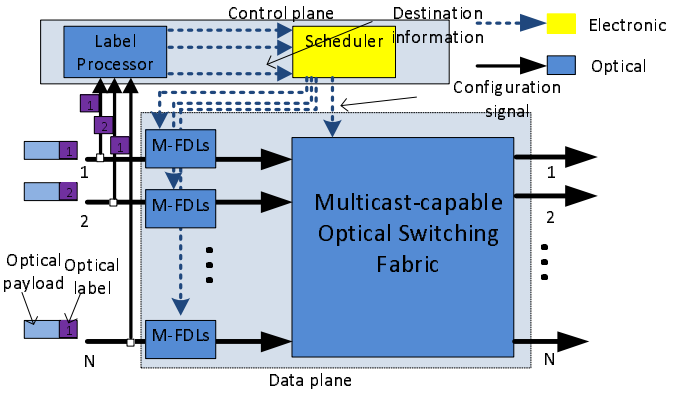


Fig. 1. Architecture of a single-wavelength, input-buffered $N \times N$ optical multicast packet interconnect.

The remainder of the paper is organized as follows. Section II presents the architecture of the adopted all-optical multicast interconnect and optical buffer. Section III describes the details of the all-optical multicast scheduling (LLMS) algorithm. Section IV presents the pipeline and parallel technique and the corresponding hardware implementation that reduces the time complexity of LLMS. Section V presents the performance evaluation results. Finally, Section VI concludes the paper.

II. Interconnect Architecture and Buffer Management

In this section, we briefly describe the adopted interconnect architecture and the operation of the proposed optical buffer, multicast-enable FDLs (M-FDLS).

A. Interconnect Architecture

We consider a simple single-wavelength, input-buffered optical multicast packet interconnect, whose high level view is

depicted in Fig. 1.

We assume the interconnect operates in a time-slotted manner and uses optical packets of the same duration with low bit rate headers to facilitate processing at the scheduler. 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. It is also encoded at a low fixed bit rate to allow easy optoelectronic conversion and electronic processing. The payload duration is fixed to a time slot, such that its data volume is proportional to the user-defined bit-rate ranging from Mbs per second to hundreds of Gbs per second. In an all-optical packet switched network, the payload of each packet transmits across the network transparently, and is only electronically recovered at end points.

The adopted interconnect consists of the optical switching fabric and M-FDLs as input buffers in the data-plane, and optical label processors and electronic scheduler in the control plane. When an optical packet arrives at an input port, the input port aligns the incoming packet related to the switch master clock in order to synchronize packet flows, which is necessary for packet header recovery. Then, the packets label is stripped off and sent to the label processor, which can be performed passively by the optical correlation technique [28]. The label processor then converts all-optical headers to electronic form, and sends them to the electronic scheduler, which calculates the schedule for each packet. Based on scheduling results, control signals are issued to the FDL buffers and switching fabric to properly configure the interconnect. Finally, updated headers are reinserted, and the packets are sent out of the switch.

There are two common types of multicast-capable optical switching fabrics. The first one is SOA-based broadcast-and-select (BS) optical switching fabric [5], [6]. The maximum size of a BS switching fabric is limited by the large number of SOA gates required and considerable splitting power losses. By interconnecting smaller switching modules using a Clos architecture, the BS switching fabric can scale up to 32×32 currently [29]. The second type is active vertical coupler (AVC)-based optical cross point switch matrix (OXS). Also, limited by signal noise accumulation and gain saturation, a multicast OXS switch can scale up to around 40×40 [30].

Similar to many existing works in the literature, such as European KEOPS switch [5], [6], we adopt the broadcast-and-select optical switching fabric. Since couplers and SOAs inevitably introduce signal distortion, after packets are delivered to corresponding output ports, they will go through proper regeneration (e.g., reamplification and reshaping) to reduce signal degradation [5].

B. Buffer Management

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 *unit-length* FDL segments. Each unit-length FDL segment can provide a delay of T, which is the duration of a time slot. To provide flexible delays ranging from T to dT, a total of d FDL segments are needed.

The FDL segments are connected by 1×2 *switching modules*. To support controllable delay and flexible packet duplication for multicast packets, each switching module can be set to one of three possible states: when it is in "bar" state (the default state), packets simply go through it and move to the next FDL segment; when it is in "split" state, a copy of packet will be sent to the interconnect for transmission through an optical multiplexer, while the packet continues to move forward in the M-FDLs; when it is in "cross" state, packets will move out of the M-FDLs completely and be sent to the interconnect for transmission.

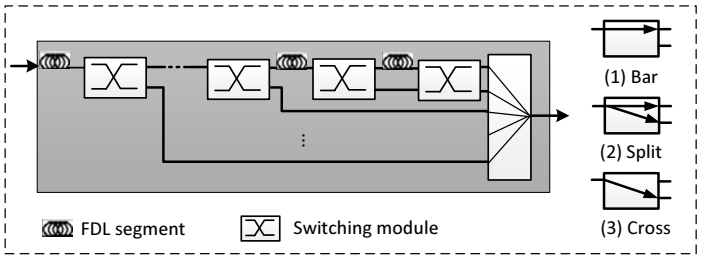


Fig. 2. Multicast-enabled FDLS (M-FDLs). Left: Structure of M-FDLs. Right: Three possible states of switching modules: bar, split and cross.

Duplicating packets in a switching module causes considerable splitting power loss, which is an important concern in the design of M-FDLs. Here, we borrow some ideas from the design of switching cells in multicast optical cross point switches (OXS) [30], and present an implementation of switching modules that uses active vertical couplers (AVC) to achieve lossless packet duplication.

As shown in Fig. 3, a switching module is built using two AVCs (called AVC1 and AVC2) perpendicularly intersecting each other, which are formed using a light-amplifying active waveguide layer grown on top of the passive waveguide. When a switching module is at "bar" state, optical signal simply passes through the bottom passive waveguide with negligible insertion loss and SNR degradation, as shown in Fig. 3(a). When a switching module is at "cross" state, optical signal is coupled into the upper active waveguide. By adjusting the injecting carrier density at a proper level, the signal power is completely transferred from AVC1 to AVC2, amplified in the process, and as a result, the packet exits the M-FDLs buffer completely, as shown in Fig. 3(b). Finally, when a switching module is set to "split" state, the optical signal is coupled into active waveguide with about equal parts of input signal power at the end of both couplers, as shown in Fig. 3(c). The gain of the active waveguide is set to be sufficient to overcome the split loss, therefore realizing lossless packet duplication.

Even though insertion loss caused by packets splitting is no longer an issue, each splitting operation decreases the optical signal-to-noise ratio (OSNR) of a packet. However, the noise accumulation is very slow, since a high OSNR of 25 dB can still be achieved after more than 40 stages of splitting [30]. This is sufficient for current multicast optical packet switches, because a packet can be split at most N times, where N is the switch size, and as mentioned before, the size of multicast optical switching fabrics is limited in practice.

Meanwhile, it should be mentioned that the ultra-high bit rate of state-of-art OPS (e.g., 640 Gbit/s OPS has been demonstrated [27]) implies that each optical packet can be fit into a time slot of very short duration, say, a few ns, such that a FDL segment of reasonably short length can provide sufficient buffer depth.

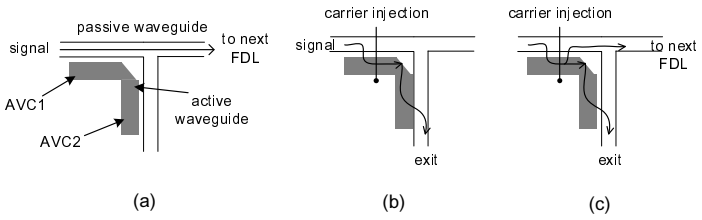


Fig. 3. Schematic illustrations of a switching module: (a) bar state; (b) cross state; (c) split state.

For example, a FDL buffer that can provide variable delay with a maximum delay of 30 time slots has been developed for error-free buffering in high-speed OPS [31]. The buffer uses 30 FDL segments with linear incremental lengths, in which the shortest segment can provide one time slot delay and the longest one can delay optical packets for 30 time slots. Overall, one optical buffer like this demands FDL segments with a total length that can delay optical packets for $(1 + 2 + \cdots + 30 = 465)$ time slots. Therefore, we estimate that placing hundreds of unit-length FDL segments in each M-FDLs is feasible with current technology.

We now use an example to illustrate how the M-FDLs works. 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 fabric. 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 [3], [4], since each M-FDLs can be shared by all the incoming multicast packets in a pipelined fashion. On the other hand, M-FDLs can achieve the same buffer depth using much shorter FDL segments, compared to the incremental buffer used in the output buffering scheme [22], [31]. For example, as mentioned earlier, to achieve flexible delays ranging from $1T$ to $30T$, an incremental optical buffer in [22], [31] requires FDL segments with a total length that can delay packets for 465 time slots, while each M-FDLs only requires 30 unit-length FDL segments, which is a substantial saving.

There has been some previous work [23], [32] on cascading couplers and FDLs to build optical buffers with variable delay. The novelty in the proposed M-FDLs compared to these buffer structures is that the M-FDLs buffer not only provides variable delay, but can duplicate packets at the right time, hence greatly facilitates multicast scheduling. Based on the input-buffered optical multicast packet interconnect and M-FDLs buffer described above, we will present the Low Latency Multicast Scheduling (LLMS) Algorithm in the next section.

III. Low Latency Multicast Scheduling (LLMS)

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 fabric such as Broadcast-and-Select (BS) has an *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 interconnect. Thus, in this paper, we adopt fan-out splitting discipline.

B. General Description

In this subsection, we give a general description on how the Low Latency Multicast Scheduling (LLMS) algorithm works.

As mentioned earlier, the optical packet interconnect we consider is input-buffered, where a M-FDLs is placed in each input port. Each M-FDLs is capable of providing flexible delay within a range as well as producing duplicated copies for each entering packet. On the other hand, there can be at most one packet exiting from each output in each time slot. Hence, the schedule in each time slot must be contention-free, that is, no more than one packets are sent towards the same output port in a time slot. Therefore, the main objective of the proposed scheduling algorithm is to interleave arriving packets onto contention-resolved schedules in one or more future time slots, such that they are delivered with low transmission latency.

According to the operations of the adopted interconnect, the label of an arriving packet will be sent to the scheduler to be processed the moment the packet enters the corresponding M-FDLs. Every packet has to go through the first FDL segment inside the corresponding M-FDLs buffer. When a packet reaches the first coupler, it has the option to send a copy to the interconnect or exit the buffer. In this way, all arriving packets will be delayed for at least a period of T (where T is the time slot length), which allows the scheduler to make proper decisions. We adopt a centralized scheduler, which keeps track of all the arriving packets in the current time slot and all the packets currently inside M-FDLs. The scheduler makes the following decisions for each arriving packet: (1) in which future time slot will the packet be sent to the switching fabric; (2) if it is to enter switching fabric, which outputs the packet will be delivered to, such that no output contention could occur. Then, according to the schedule, the scheduler sends

coordinated control signals to the switching fabric and M-FDLs for contention-free packet transmission. For example, suppose a packet is scheduled to be delivered to a subset of its destination output ports t time slots after it enters the M-FDLs. Then, right before the packet reaches the t^{th} coupler of the M-FDLs, the corresponding coupler duplicates the packet and a copy is sent to the switching fabric, which is properly configured to deliver the packet to the corresponding output ports.

To ensure low packet latency, the scheduler adopts a greedy strategy that delivers the output copies of a packet as early as possible. If a packet cannot be delivered to all its destination output ports before it reaches the end of the M-FDLs, the remaining output copies will be dropped. We should also consider the limitation of the switching fabric, which can send up to one packet from each input port to outputs. Therefore, the scheduler should prevent multiple packet copies from entering the same input port simultaneously, which can be done by setting the constraint that at most one packet copy is allowed to exit each M-FDLs in a time slot. Next, we describe the implementation details of the LLMS algorithm.

C. Implementation Details

In this subsection, we describe in detail the Low Latency Multicast Scheduling (LLMS) algorithm.

Consider an interconnect of size $N \times N$ as shown in Fig. 1, and assume that packets arrive at the beginning of each time slot. LLMS 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 t is denoted by S_t, which is used for keeping track of scheduling results of the time slot that is t 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 o^{th} entry of S_t is denoted by $S_t(o)$. If a copy of some packet for output o is scheduled to be transmitted in the t^{th} time slot, we say that output copy o of the packet is *assigned* to entry $S_t(o)$.

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*, while the $split$ field is used to configure the state of couplers in M-FDLs. For example, suppose an output copy of some packet is assigned to the t^{th} scheduling vector, then the $location$ field of the assigned entry is set to t. If the packet still has leftover output copies that remain to be scheduled, then the $split$ field of the assigned entry is set to 1, which indicates that a copy will be created by setting the t^{th} coupler to "split" after t time slots while the packet continues to move along the M-FDLs afterwards. Otherwise, the $split$ field is set to 0, indicating that the t^{th} coupler will be set to "cross" and the packet will exit the M-FDLs after t time slots.

Since the switching fabric can only transmit up to one packet from each input to output ports within a time slot, at most one packet can come out of the same M-FDLs in one time slot.

TABLE I
LOW LATENCY MULTICAST SCHEDULING (LLMS)

Input to LLMS: 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 $t = 2$ to D **Do**
  $S_{t-1} = S_t$;
  $M_{t-1} = M_t$;
EndFor
Clear S_D, M_D;
// find the earliest eligible entries for the arriving packets
For packet P_i from input i,
$i = [pr, pr \mod N + 1, (pr + 1) \mod N + 1, \cdots, (pr + N - 2) \mod N + 1]$. **Do**
  For scheduling vector $S_t, t = [1, 2, \ldots, D]$ **Do**
    For each output o in P_i's fanout, **Do**
      If $S_t(o)$ is empty **and** $M_t(i) == 0$
        Assign the entry to the output copy of P_i;
      EndIf
    EndFor
    If some copies of P_i get assigned in S_t
      set $M_t(i) = 1$;
    EndIf
  EndFor
  If there are still outputs in P_i's fanout left undelivered
    Drop these output copies of P_i;
  EndIf
EndFor

Also, each output can receive at most one packet in each time slot to avoid output contention. Therefore, an entry of index o in a scheduling vector is said to be *eligible* for an output copy of some packet from input i 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 o^{th} output.
3) No packets from input i 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 i^{th} entry of the t^{th} mask vector is denoted as $M_t(i)$, which is set to 1 if some packet from the i^{th} input has been assigned to the t^{th} scheduling vector. The scheduler will check mask vectors before assigning output copies to ensure 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.

In order to reduce the delay each packet experiences in buffers, the basic operation of the scheduler is to find the *earliest possible* eligible entries for arriving packets in each time slot. To prevent packets of an input port with a smaller index from always being scheduled earlier, the round robin policy is used to allocate priority to the packets arriving at different inputs to be scheduled in each time slot. We choose round robin policy for two reasons. Firstly, as the packet that is scheduled first will generally be transmitted with shorter delay than others, we need to ensure that all packets receive similar performance regardless of their inputs. Round robin policy has been widely applied in switch scheduling and shown to be effective in maintaining fairness. Secondly, as will be shown

later, round robin can be readily implemented in hardware using a shuffle network, which does not incur extra time cost.

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 $[pr, pr \mod N+1, (pr+1) \mod N+1, \cdots, (pr+N-2) \mod N+1]$, and tries to assign their output copies to the earliest eligible entries among D scheduling vectors. For example, consider a switch with size $N=4$ and current priority $pr=2$. Then, the arriving packet from input 2 is scheduled first. Next, packets from inputs $3, 4$ and 1 will be scheduled one by one. pr is updated to 3 in the next time slot, then $4, 1, 2 \ldots$ in the subsequent time slots. When an output copy cannot be assigned after searching all D scheduling vectors, it will 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. It is easy to see that the time complexity of LLMS is $O(N^2D)$ in the worst case, where N is the interconnect size and D is the number of scheduling vectors.

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_t is moved to $S_{t-1}, t = 2, 3, \ldots, D$. Then, the last scheduling vector and mask vector is emptied, because packets existing in M-FDLs buffers would have been transmitted within next $D-1$ time slots. Finally, the scheduler starts the scheduling process as described above 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. Clearly, the delay of any transmitted output copy is bounded by the number of scheduling vectors D in LLMS. The detailed description of LLMS is given in Table I.

D. A Scheduling Example

A scheduling example of LLMS algorithm for a 4×4 interconnect is shown in Fig. 4. 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. 4(a). Note that LLMS allows the packets arrived later to be

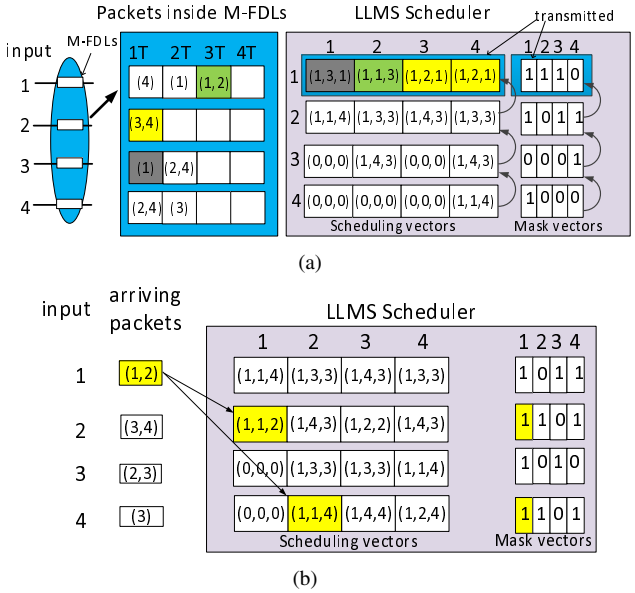


Fig. 4. A scheduling example for a 4×4 interconnect. (a) The output copies corresponding to S_1 are transmitted (marked by blocks of the same color), then the scheduler rotates the scheduling vectors and mask vectors. (b) At the beginning of the next time slot, the scheduler schedules all arriving packets. The schedule for packet $(1, 2)$ is marked by yellow blocks as an example.

transmitted before the packets arrived earlier, thus eliminates the Head-of-Line (HOL) blocking. For example, packet (1) from input 3 arrived one time slot later than packet $(2, 4)$, yet is scheduled to be transmitted earlier. Such a feature enables more efficient buffer management and reduces packet delay.

In the meanwhile, it is worth mentioning that in-order transmission of packets in the same flow (i.e., packets sharing the same input and fanout) is guaranteed in LLMS. For example, suppose packet A arrives at some input port with a previous packet B in the same flow still in the M-FDLs buffer. We can prove by contradiction that any output copy of A cannot be scheduled before B in LLMS, given that the output copy of B has not been dropped. Assume that an output copy of packet A is assigned an eligible entry that is earlier than the corresponding output copy of B. Since each eligible entry for packet A would also be eligible for B as they belong to the same flow, such a scheduling result contradicts with the scheduling procedure of LLMS, which always assigns the *earliest eligible* entries for each packet among the scheduling vectors. Therefore, we can see that in-order transmission is guaranteed in LLMS, as it is impossible to deliver a later packet prior to its predecessor in the same flow.

The scheduler then configures the interconnect and M-FDLs according to the first scheduling vector, and rotates the scheduling vector and mask vector forward by one position, as shown in Fig. 4(a). 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. Packet (1) from input 3 (grey blocks) and packet $(3, 4)$ from input 2 (yellow blocks) will be transmitted completely, while packet $(1, 2)$ from input 1 (green blocks) will send one copy to output 2, and stays in the M-FDLs for future transmission. Assume the current priority indicator pr is 1. The scheduling results of the arriving packets are shown in Fig. 4(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

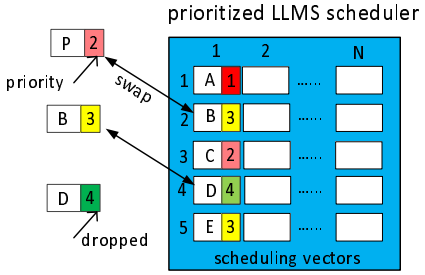


Fig. 5. An example for the prioritized LLMS ($D = 5$). The scheduler swaps packet P with packet B. Then, it schedules B from the rest of the scheduling vectors (begin from the $3^r d$ vector). B is swapped with packet D. Low priority packet D is dropped as no eligible entry can be found from the rest of the scheduling vector.

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

E. Prioritized LLMS

In practice, Internet traffic consists of flows with different Quality-of-Service (QoS) requirements. For example, some video-on-demand service providers offer good QoS guarantee for premium members, while trying to serve free viewers in a best-effort manner. Therefore, it is desirable for interconnects to be able to provide differentiated QoS for flows of varied priorities. In this subsection, we propose a simple yet efficient modification to LLMS, denoted as prioritized LLMS, that is able to schedule each packet according to the priority of the corresponding flows.

Similar to the original LLMS, the prioritized LLMS also tries to find the earliest *eligible* entry among the scheduling vectors for output copies of each packet. Recall that an eligible entry has to be empty in the original LLMS, as denoted in the first of the three conditions. In the prioritized LLMS, we modify the condition to be that an entry is eligible for an arriving packet if it is empty *or* has been assigned to some packet with lower priority. Correspondingly, if the eligible entry found for an arriving packet is empty, then the packet is assigned in the same way as the original LLMS. Otherwise, if the eligible entry has been assigned to some packet with lower priority, the scheduler will *swap* the two packets, that is, assigns the entry to the packet with higher priority, then continues to schedule the packet with lower priority among the rest of the scheduling vectors. Low priority packets will be dropped if no eligible entry can be found.

Next, we use a simple example to illustrate the operation of the prioritized LLMS in Fig. 5. As the prioritized LLMS operates similarly to the original LLMS, we only show their differences. We assume that each packet carries signaling bits representing the priority of the corresponding flow, with higher priority denoted by a smaller number, as shown in Fig. 5. At the beginning of a time slot, the prioritized LLMS scheduler tries to schedule an arriving packet P destined for output 1 with priority 2. The earliest eligible entry found for packet P has already been assigned to an earlier packet B with priority 3, thus the scheduler swaps packet P and B, then continues to schedule packet B among the rest of the scheduling vectors. Similarly, the scheduler swaps packet B with a packet with lower priority D, then tries to schedule D from the scheduler vectors left. As an eligible entry for packet D cannot be found, the packet is thus dropped.

It can be observed that packets from flows with high priority will generally be assigned to the front of scheduling vectors, whereas low priority packets occupy the back of the scheduling vectors. Correspondingly, the packets from high priority flow will be scheduled with lower latency and drop ratio. Therefore, with simple modification to the original LLMS algorithm, the prioritized LLMS can provide differentiated QoS according to the priorities of traffic flows efficiently.

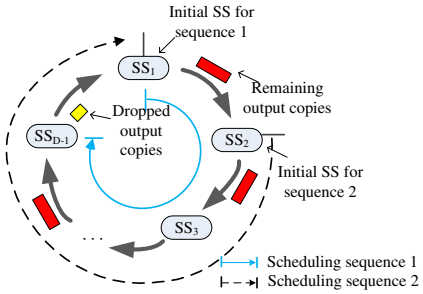


Fig. 6. Ring of cascaded schedulers. The solid line and dashed line indicate the sequence of sub-schedulers packets go through in different time slots.

IV. Pipeline and Parallel Architecture

In this section, we first present a pipelining technique that distributes scheduling tasks to a sequence of sub-schedulers, each of which can finish its scheduling task in $O(N^2)$ time. Then, we show the procedures within each sub-scheduler can be further distributed to N processing modules that operate in parallel, which can be built using simple combination gates. We also give the combination circuit implementation for each processing module. We show that with such an implementation, each processing module in the proposed pipeline and parallel architecture achieves $O(1)$ time complexity.

The most time consuming part in LLMS 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. 6.

In each time slot, the arriving packets are processed through a sequence of sub-schedulers along the ring in a pipelined fashion. Each $SS_t, t \in [1, 2, \ldots, D]$, has one built-in scheduling vector S_t and mask vector M_t, and takes as inputs the remaining output copies of the processed packets that have not been scheduled. Each SS is responsible for the scheduling of the processed packets according to the built-in scheduling vector and mask vector, then passes the remaining copies as outputs to the next SS. The output copies that cannot be scheduled after going through all the D SS will be dropped. We denote the first sub-scheduler in the sequence as the *initial* SS. Starting from the initial SS, the t^{th} SS in the sequence is responsible for the scheduling for the time slot which is t time slots after the current time slot.

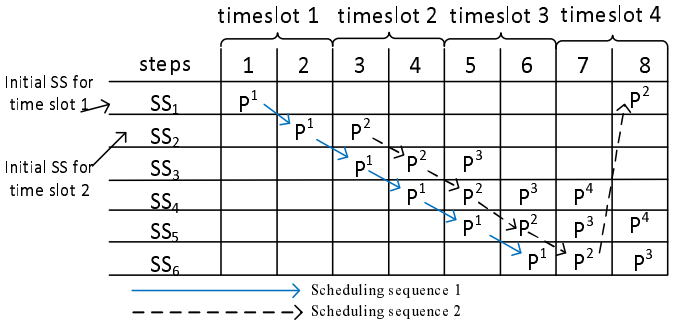


Fig. 7. Pipeline operation for the first three time slots ($D = 6$).

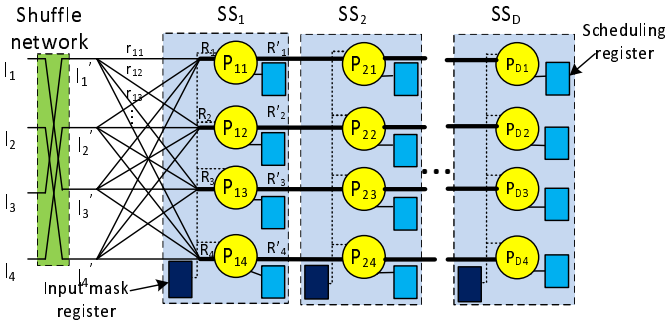


Fig. 8. Pipeline and parallel architecture ($N = 4$).

To relax the time constraints of LLMS, all sub-schedulers operate in a pipelined fashion. Fig. 7 illustrates the pipeline example for an optical multicast interconnect with $D = 6$ scheduling vectors, in which the time each SS takes to schedule the packet arrivals in one time slot is denoted as a *step*. The packet arrivals in the k^{th} time slot are denoted as P^k. Assume SS_1 is chosen as the initial sub-scheduler in the first time slot. SS_1 is responsible for the assignment of packet arrivals P^1 to its scheduling vector. When it finishes, the remaining output copies that cannot be assigned by SS_1 are passed to SS_2 as inputs. As shown in Fig. 7, the scheduling of P^1 is completed after all the sub-schedulers have been visited.

According to LLMS, the scheduling process for the current time slot cannot begin till all the packet arrivals from the previous time slot have been scheduled and the scheduling vectors have been shifted. As all the sub-schedulers must be visited during the scheduling process for one time slot, each step has to be completed within $1/D$ time slot. However, a key observation is that only the results of the first scheduling vector S_1 need to be ready for interconnect configuration in each time slot. After interconnect configuration, the information stored in S_1 will no longer be needed. This observation makes it possible to pipeline the scheduling for consecutive time slots to further reduce the time constraints of LLMS, as explained next.

In LLMS, 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 simplify the operation, we clear the initial SS then simply "shift" the scheduling sequence 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. 6, 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. 6. 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.

Moreover, 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 the scheduling for the previous time slot to complete before starting the scheduling for the next time slot. In other words, the scheduling in consecutive time slots can also be pipelined. At the beginning of the 2^{nd} time slot, the scheduler configures the interconnect according to the scheduling vector of SS_1, then clears the register storing the scheduling vector and mask vector in SS_1. Next, it shifts the initial SS of the processing sequence for the 2^{nd} time slot to SS_2. Instead of waiting for the completion of the scheduling for the 1^{st} time slot, the scheduling for the 2^{nd} time slot can start as soon as SS_2 is available, as shown by the dashed line in Fig. 7. Similarly, the scheduler configures the interconnect according to SS_2 and initiates the processing sequence from stage 3 at the beginning of the 3^{rd} time slot. To avoid conflict (i.e., the packet arrivals from two time slots enter the same SS), each SS needs to complete a step within a half time slot, which is a constant factor independent of the interconnect size N and the number of scheduling vectors D.

In each step, each SS has to schedule up to N arrival packets among the N entries in its scheduling vector, which takes $O(N^2)$ time. In high-speed switching, this requirement may be too stringent when N is large. Therefore, we also propose a processing architecture, which distributes the processing task of each SS to N processing modules operating in parallel. We show that combined with the pipelining technique and combination circuit implementation, the proposed processing architecture can reduce the time complexity of LLMS to $O(1)$.

The architecture for a 4×4 interconnect is shown in Fig. 8, which consists of D cascaded sub-schedulers, with N processing modules in each SS. Each processing module $P_{kn}, (n = 1, 2, \ldots, N)$ in SS_k has a scheduling register V_{kn} storing the n^{th} entry of the corresponding scheduling vector, and all the processing modules in SS_k share one register recording the input mask vector.

At the beginning of each time slot, input port i sends a binary request $r_{ij}(j = 1, 2, \ldots, N)$ to the j^{th} processing module in the initial SS according to the fanout information of the incoming packet, that is, $r_{ij} = 1$ if the incoming packet is destined for output j, otherwise $r_{ij} = 0$. The set of requests towards output j, i.e., $r_{ij}(i = 1, 2, \ldots, N)$, are denoted as R_j. For processing module P_{kj}, a request r_{ij} is said to be *eligible* if scheduling register V_{kj} is empty and no previous packet from input i has been assigned to stage k. The basic operation for each processing module is to assign the eligible request with the *smallest* input index to its scheduling register.

In each time slot, processing module P_{kj} in the initial stage k takes request set R_j as the input, and outputs the set of remaining requests R'_j after processing, which is then fed to the next stage in the sequence as input. This process repeats until all the D stages have been visited. If there are still requests left unassigned, the corresponding output copies will be dropped. Since inputs with a smaller index are always scheduled at higher priority, we also adopt a simple shuffle network as in [22] to dynamically change the priority of each input port. In this way,

all input ports have an equal chance to get its requests assigned first.

For each processing module, it takes $O(N)$ time to search among N requests, which is a non-trivial task. Next, we show this operation can be implemented by the following simple combination circuit with $O(1)$ time complexity. For processing module P_{kj}, we have following variables.

1) Request vector, denoted as $r_{ij}, i = 1, 2, \ldots, N$, is the input to the processing module; the remaining request vector after processing, denoted as r'_{ij}, is the output to the processing module;
2) $full$ bit of the entry in the scheduling register, denoted as F, where $F = 1$ if the entry of the scheduling vector is occupied, otherwise it is 0; updated $full$ bit after processing is denoted as F';
3) Input mask vector, denoted as M_i, where $M_i = 1$ if a packet from input i has been assigned to stage k previously, otherwise it is 0; updated input mask vector after processing is denoted as M'_i;
4) Scheduling results denoted as Q_i, $i = 1, 2, \ldots, N$. $Q_i = 1$ if request r_{ij} is assigned to the scheduling register in the processing module, otherwise it is 0;

For a processing module P_{kj}, Q_i is set to 1 only when all the following three conditions are met:

1) The scheduling register must be empty, i.e., $F = 0$;
2) None of the requests from inputs with an index smaller than i can be assigned to the scheduling register, that is, $r_{xj} \cap \overline{M_k(x)} = 0$ for $x \in \{1, 2, \cdots, i-1\}$;
3) The request from input i must be able to get assigned to the scheduling register, $r_{ij} \cap \overline{M_k(i)} = 1$.

Thus we can give the logic expression for Q_i as follows, from which r'_{ij}, F' and M'_i can be easily derived.

$$Q_i = \overline{F} \cap \overline{(r_{1j} \cap \overline{M_1})} \cap \cdots \overline{(r_{i-1,j} \cap \overline{M_{i-1}})} \cap (r_{ij} \cap \overline{M_i})$$

With pipeline and parallel processing, the time complexity of each processing module to implement LLMS becomes $O(1)$, and the hardware cost (i.e., the number of logic gates) of the proposed architecture is $O(N^2 D)$. It is worth mentioning that the proposed architecture can implement the prioritized LLMS algorithm in Section III-E with a small modification: each processing module can use a comparator to compare the priority of incoming requests with that of currently assigned to its scheduling register, and simply swap the assigned one with the incoming request if the incoming one has higher priority.

Note that, the parallel pipelined architecture for output buffering scheme in [22] also achieves $O(1)$ time complexity. However, each packet has to go through multiple processing stages before it can be scheduled in the processing architecture, which introduces large pipelining overhead. In comparison, the scheduling results are immediately available at the initial stage of the processing sequence in the proposed scheduling architecture, therefore no scheduling overhead is introduced. Also, it should be emphasized that the processing module in the proposed scheduler can be built using simple combination gates, thus is much less costly than a general-purpose processor.

As mentioned before, the size of multicast capable optical switching fabrics is limited (no larger than 64×64 currently [29]). Therefore, one or a small number of high end FPGAs are sufficient to implement the proposed scheduler in practice. To see the hardware feasibility of the proposed architecture under high-speed switching, let us look at a 64×64 optical packet interconnect with 40 Gb/s port speed. Assume the packet length is 128 bytes and there are $D = 16$ scheduling vectors. Then the processing architecture requires approximately 65,000 logic gates and 80 MHZ clock frequency, which can be easily accommodated by the state-of-art FPGAs. To build a scheduler for larger switches in the future, we may need to use multiple FPGAs. One possible solution is to use FPGA prototyping platforms [33], which interconnect a number of FPGAs to provide significantly more powerful processing capacity.

V. Performance Evaluations

We have conducted extensive simulations to evaluate the performance of LLMS, which 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 arriving during the simulation period, and the average delay, which is calculated by the average interval between the arrival and departure of all successfully transmitted output copies.

Due to the similarity between the adopted switching architecture and the input queued (IQ) electronic interconnect, we compare the performance of LLMS with several well-known multicast scheduling algorithms for IQ electronic interconnects, including FIFOMS [17], MCMS [19] and MLRRMS [20] algorithms. In our simulation, we also include a simple FIFO scheduling algorithm on the output queued interconnect (OQFIFO) as a performance benchmark. Scheduling algorithms for electronic interconnects usually assume infinite buffer size because of the available large size electronic buffer, which means that no packets will be dropped in the scheduling process, therefore only average delay performance will be evaluated when comparing LLMS with these algorithms. To evaluate the performance of LLMS in the terms of both average delay and packet drop ratio, we also compare LLMS with the output buffering scheme for optical packet interconnects [22]. 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 timestamp 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 latency.

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 latency performance of MCMS outperforms most of previous scheduling algorithms such as WSPLIT, revision scheme and windows-based algorithms.

MLRRMS schedules the HOL packets in each input queue first. If there are any output and input left idle, the scheduler then tries to send the packets behind the HOL packet in each idle input to idle outputs. This process continues until either the maximum look-ahead depth d is reached or all packets in the queues are examined, or no idle outputs are found in the schedule. MLRRMS can effectively alleviate HOL blocking, thus achieves high switch throughput. However, one major

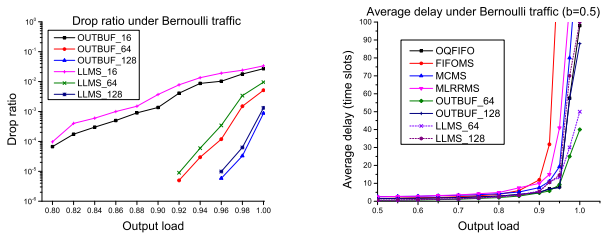
constraint of MLRRMS is that the internal memory access rate of input buffers must run d times faster than switch port bit rate. Such a constraint makes MLRRMS impractical to implement in high speed switches. Nevertheless, we include its simulation results to evaluate LLMS. The maximum search depth d is set to 1 in the simulation as in [20], which means that the scheduler examines the first two packets in each queue.

The output queued interconnect is known to have optimal delay performance when no packets are dropped, but requires N times faster memory speed. Despite its much stronger hardware requirement, in our simulation, we include a simple FIFO scheduling algorithm on the output queued interconnect (**OQFIFO**) as a performance benchmark to demonstrate the good performance of LLMS.

The output buffering (**OUTBUF**) scheme places an output buffer consisting of an $N \times D$ high speed optical interconnect and D FDLs with incremental delay in front of each output port. When there are multiple packets destined for the same output, the scheduler determines the proper FDL segment that each packet should go to, such that they would exit the FDL buffer without contention. When operating in time-slotted manner, OUTBUF can emulate the OQFIFO scheduling with the constraints that each output queue is of size D, thus can achieve optimal performance in both average delay and packet drop ratio. However, as shown earlier, OUTBUF requires a prohibitive amount of FDL segments. We use it as a performance benchmark to test how close our algorithm can be to the optimal results in the terms of packet drop ratio and average delay.

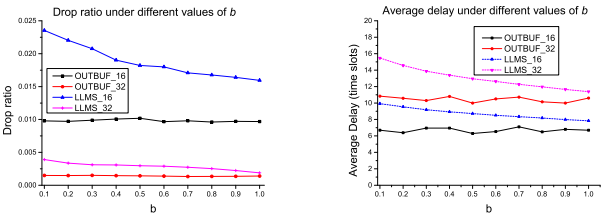
In the second part of the simulation, we study the performance of the prioritized LLMS algorithm. Similar to [34], we consider two priority classes in the traffic, with the arrival ratio of high priority traffic to low priority traffic $\rho_1 : \rho_2$ set to $1 : 3$, i.e., 25% of packets are high priority packets and the remaining are low priority packets. The effectiveness of prioritized LLMS is demonstrated by comparing the packet drop ratio and average delay of traffic belonging to different priority classes.

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 interconnect reaches a stage where it cannot sustain the offered load). For cases in which the packet drop ratio is in the order of 10^{-4}, we extend the simulation period to 10^7 time slots for more reliable results. In order to compare the performance of the algorithms in various networking environments, simulation are conducted for different interconnect sizes (8×8, 16×16 and 32×32) under several different types of traffics, including Bernoulli traffic, gathered traffic, unicast traffic, and real Internet traffic. For statistical traffic patterns (i.e., Bernoulli traffic, gathered traffic and unicast traffic) used in this paper, all inputs are assumed to have identical packet arrival process, and for Internet traffic simulation, a different trace file is fed into each input to simulate the packet arrival process. Since we observe similar results for all interconnect sizes, only the results for the 16×16 interconnect are shown in the paper.



(a) (b)

Fig. 9. Performance for a 16×16 interconnect under Bernoulli traffic with $b = 0.5$. (1) packet drop ratio; (b) average delay.



(a) (b)

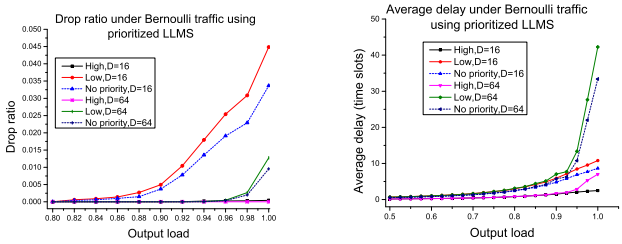
Fig. 10. Performance for a 16×16 interconnect under the same output load (0.95) with different values of b. (1) packet drop ratio; (b) average delay.

A. *Performance under Bernoulli Traffic*

Bernoulli traffic is one of the most commonly used traffic models in the simulation of scheduling algorithms. Bernoulli traffic can be described using two parameters, λ and b. λ is the probability that an input port is busy in a time slot, i.e., the rate packets arrive at some input port at the beginning of a time slot. Given an arriving multicast packet, determining whether the packet is destined to a specific output can be considered as a *Bernoulli trial*. The trial is called a "success" if the output is a destination of the packet. As there are N outputs, finding the set of destination outputs of a packet requires N independent and identical Bernoulli trials, with b being the success probability in each trial. The number of destinations outputs of the packet follows a binomial distribution with expected value bN. b is set to 0.5 in the simulation. For an $N \times N$ interconnect, the average fanout of a multicast packet is bN and the output load μ is λbN.

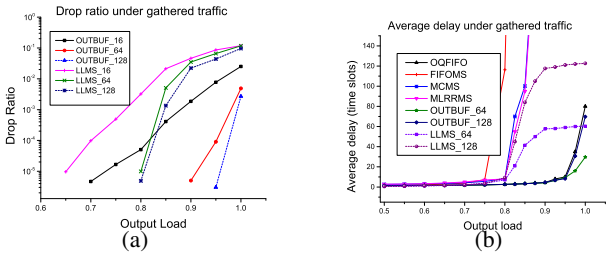
We first compare the packet drop ratio of both LLMS and OUTBUF under Bernoulli traffic in Fig. 9(a) for different values of D. For LLMS, D means the number of scheduling vectors, whereas D is the longest delay of each output buffer in OUTBUF. As can be seen from the figure, packet drop only occurs at very high traffic loads (over 0.8) under Bernoulli traffic, which can be explained as follows. When the traffic load is light, the scheduling vectors are relatively empty, and the expected number of arriving packets competing for the scheduling vectors in each time slot is small. Therefore, most output copies can be assigned in the vectors and few packets will be dropped. On the other hand, when the traffic load is heavy, the scheduling vectors are mostly occupied, and the expected number of arriving packets in each time slot is larger, leading to increased packet drop ratio.

The drop ratio reduces drastically when we increase D from 16 to 64, but only a slight improvement is observed when we further increase D to 128, indicating that most packets can be scheduled for transmission within 64 time slots. LLMS closely matches OUTBUF in the terms of packet drop ratio when D



(a) (b)

Fig. 11. Performance comparison of traffic of different priorities for a 16×16 interconnect under Bernoulli traffic using prioritized LLMS. (1) packet drop ratio; (b) average delay.



(a) (b)

Fig. 12. Performance for a 16×16 interconnect under gathered traffic with probability destined for each output $b = 0.5$. (1) packet drop ratio; (b) average delay.

is sufficiently large (e.g., $D = 64$). When D is small (e.g., $D = 16$), the packet drop ratio of LLMS is slightly higher than that of OUTBUF, which indicates that LLMS can achieve near-optimal packet drop ratio under Bernoulli traffic. The reason why LLMS has higher packet drop ratio than OUTBUF is that a packet cannot be assigned to an entry in a scheduling vector even when it is empty, if a previous packet from the same input is scheduled to be transmitted in the same vector, which leads to more dropped packets under LLMS compared with OUTBUF.

Fig. 9(b) compares the average packet latency of LLMS under Bernoulli traffic with other algorithms. It can be seen that, as the traffic load increases, MLRRMS, MCMS and FIFOMS become saturated due to HOL blocking, which coincides with the theory that the IQ interconnect cannot maintain sustainability under all admissible multicast traffic conditions [35]. At the same time, we can see that the proposed LLMS algorithm closely matches the performance of OUTBUF, and significantly outperforms all other algorithms when the traffic load is heavy.

When the traffic load approximates 1, LLMS achieves better delay performance than OQFIFO because LLMS can detect and promptly drop output copies with overlong latency instead of keeping them in the buffer, thus significantly reduces the average packet latency.

We also observe that LLMS_64 performs better than LLMS_128 in terms of average packet latency. The reason is that with more scheduling vectors, the scheduler is less prone to drop packets and more tolerant to packet latency. For example, an output copy of some packet that expects to have a delay of 70 time slots would be dropped by LLMS_64, while it would be scheduled for transmission by LLMS_128. Such trade-off between the packet drop ratio and the average packet latency can be easily adjusted by changing the number of scheduling vectors, making LLMS highly adaptive to various transmission requirements.

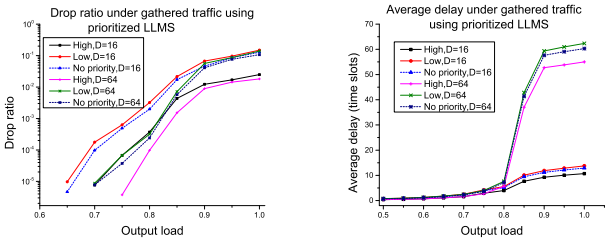
Given the same output load (0.95), Fig. 10 compares the performance of LLMS and OUTBUF under different values of b. We observe that b has negligible impact on the performance of OUTBUF, which is expected because the output load stays the same with different b. Meanwhile, packet drop ratio and average delay under LLMS slightly increase when b is smaller, which can be explained as follows. Given that at most one packet can exit an M-FDLs buffer in each time slot, a scheduled packet would block other packets arriving at the same input from using the scheduling vectors it occupies, which we refer to as input blocking. Under the same output load, the input blocking level increases when b is smaller, because of the larger packet arrival rate at each input, which causes increased packet drop ratio and average delay.

Fig. 11 compares the performance of prioritized LLMS algorithm for traffic belonging to different priority classes. The performance of non-prioritized LLMS is also presented for comparison purpose (dash lines). We consider prioritized LLMS with the number of scheduling vectors D set to 16 and 64, respectively. We can see from Fig. 11(b) that all high priority packets are delivered with negligible drop ratio under all tested output loads. The difference in average delay performance between high priority traffic and low priority traffic is also very significant, as shown in Fig. 11(b). Meanwhile, low priority traffic only suffers a marginal increase of packet drop ratio and average delay, compared to the non-prioritized LLMS. These results confirm that prioritized LLMS is very effective in providing service differentiation to traffic of different priorities.

B. Performance under Gathered Traffic

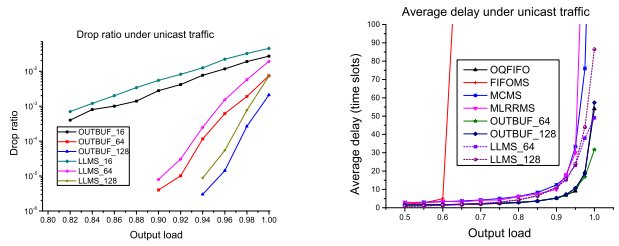
As multicast applications, such as IPTV and Video on Demand (VoD), often generate sustained and long-lasting flows that may gather among fewer active input ports and engage more output ports at a given switch, we also examine the performance of the LLMS scheduler under gathered traffic. The gathered traffic scenario is known to be difficult to schedule for input-queued switches, and has been widely adopted in the simulation studies on multicast scheduling [36]. We adopt a similar setting as [36] in our simulation, where the number of active input ports M is set to 5 and the number of active output ports N is fixed at 16. We assume the fanout of a multicast packet is chosen uniformly at random among all possible fanout sets with the average fanout $h_m = 8$. Let λ be the rate of packet arrival at each input, then output load $\mu = \lambda M h_m / N$. Next, we explore the sensibility of the scheduling performance to output load μ.

Fig. 12 shows the performance for a 16×16 interconnect under gathered traffic with probability destined for each output $b = 0.5$. From the figure, we can see that both output-queued scheduling algorithms (OQFIFO and OUTBUF) have comparable performance under the gathered traffic to the Bernoulli traffic scenario in terms of packet drop ratio and average delay, while other algorithms relying on input queue perform notably worse. The underlying reason is that input-queued scheduling algorithms, like FIFOMS and MCMS, can only send one packet for transmission from each input port within a time slot. Since traffic is concentrated among a few active input ports in gathered traffic, it is likely that the output copy of a buffered packet cannot be transmitted even though the corresponding input port and output port is idle, because it is blocked by the HOL packet in the same queue that fails to be scheduled for transmission.



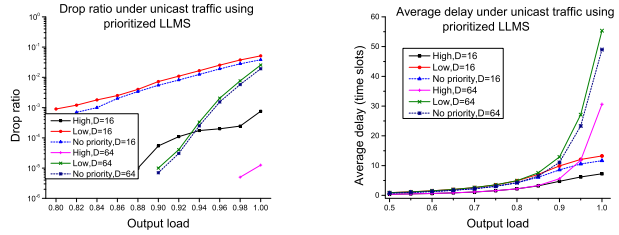
(a) (b)

Fig. 13. Performance comparison of traffic of different priorities for a 16×16 interconnect under gathered traffic using prioritized LLMS. (1) packet drop ratio; (b) average delay.



(a) (b)

Fig. 14. Performance for a 16×16 interconnect under unicast traffic. (1) packet drop ratio; (b) average delay.



(a) (b)

Fig. 15. Performance comparison of traffic of different priorities for a 16×16 interconnect under unicast traffic using prioritized LLMS. (1) packet drop ratio; (b) average delay.

Among all the input-queued scheduling algorithms, MLRRMS has the best delay performance under gathered traffic, which is because it will try to send packets behind the HOL packet of each queue to fill as many idle output as possible, which reduces the time each packet has to stay in buffer.

Meanwhile, we observe that LLMS is able to achieve significantly better performance than other input-queued algorithms in terms of average delay, and achieve comparable performance to OUTBUF when output load is below 0.9. Due to the limitation imposed by the input-queued structure, MLRRMS, FIFOMS and MCMS saturate before the output load reaches 0.85 under gathered traffic, which means that the number of queued packets will keep increasing until buffer overflow. In such cases, it could take quite a long time for the upper layer protocol to detect congestion at the switch and drop packets if the buffer size is large. In contrast, the proposed $LLMS$ enables swift congestion detection and promptly drops packets with overlong delay shortly after arrival, which allows the upper layer protocols to adjust to network condition in time.

Fig. 13 shows that prioritized LLMS is quite effective at performance differentiation in gathered traffic too: packet drop in high priority traffic occurs only under high output load ($\mu = 0.85$), and is $1/10$ that of low priority traffic when output load $\mu = 1$. The average delay of high priority traffic is also much lower than that of low priority traffic. On the other hand, we also observe that there are about 2% high priority packets being dropped under gathered traffic when output load is 1, which is higher than Bernoulli traffic scenario, where there is nearly no packet drop in high priority traffic. Such difference is also caused by the fact that arriving packets are concentrated in a few input ports, which could more easily overwhelm the capacity of each M-FDLs buffer, leading to higher packet drop ratio and longer average delay.

C. Performance under Unicast Traffic

Since a substantial portion of traffic in the Internet is unicast, where each packet is forwarded to a single destination, in this subsection, we show that LLMS is also capable of dealing with unicast traffic efficiently. For a unicast packet, it has an equal probability $(1/N)$ being destined for each output port. It is easy to see that when the traffic only consists of unicast packets, the output load μ is equal to arrival rate λ.

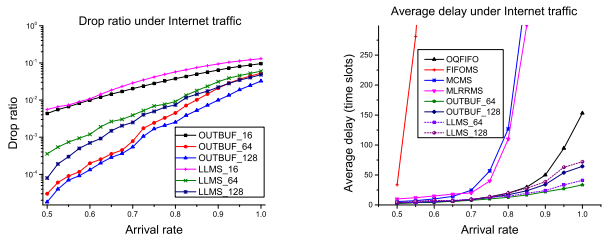
The effect of D on the packet drop ratio and average delay of both LLMS and OUTBUF under unicast traffic for a 16×16 interconnect is illustrated in Fig. 14. We can see that, with $D = 64$, packet drop only occurs at very high traffic load (over 0.95) and can be kept at very low level. We also show that the average packet latency of LLMS under unicast traffic in Fig. 14(b). We can observe that MLRRMS, MCMS and FIFOMS become saturated before the traffic load reaches 1. In comparison, LLMS consistently achieves low latency under all traffic loads, and closely matches OUTBUF in terms of both average packet delay and packet drop ratio, indicating that it can also achieve near-optimal performance under unicast traffic. Fig. 15 demonstrates that the prioritized LLMS achieves similar effectiveness to Bernoulli traffic in providing QoS differentiation under unicast traffic.

D. Performance under 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 LLMS algorithm under real Internet traffic traces obtained from the backbone network link monitors.

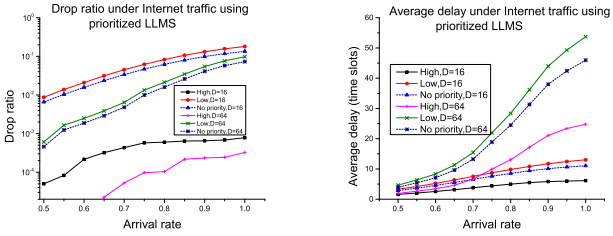
The anonymized traffic traces used here were obtained from the CAIDA's passive $OC192$ network link monitors [37]. All trace files consist of one line per IP packet arrival in the form of *<packet_ index, time_stamp, protocol, source_IP_address, destination_IP_address>*, where *time_stamp* of each packet records the exact time when the packet is intercepted by the link monitor.

In the simulation, we feed each input with a separate trace file, in which all packets are assumed to have a fixed size of 64 bytes and fit into one time slot. Note that due to lack of regulation in real Internet traffic, it is likely that certain outputs are heavily oversubscribed, that is, they receive more packets than that can be transmitted over the simulation period. Also, it is impossible to regulate the traffic load in each output. 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 as in [38].



(a) (b)

Fig. 16. Performance for a 16×16 interconnect under Internet traffic. (a) packet drop ratio; (b) average delay.



(a) (b)

Fig. 17. Performance comparison of traffic of different priorities for a 16×16 interconnect under Internet traffic using prioritized LLMS. (1) packet drop ratio; (b) average delay.

To adjust the arrival rate, we process each trace file using a similar method as in [38], elaborated as follows. First, we calculate the average throughput of a trace by dividing the total traffic volume by the time span of the trace. Then, we set the length of each time slot according to the throughput of the trace to achieve a desired arrival rate. For example, if the throughput of a trace is 500 Mb/s and we want to achieve the arrival rate of 0.8, then the time slot length is set as $((64 \times 8)/500M) \times 0.8 = 0.8192\mu$s. Finally, we find a mapping from packets to different time slots according to the time stamp of packets. If the time stamp of multiple packets fall in the same time slot, they are placed in consecutive time slots. After the above procedures, each processed traffic trace is used to simulate the packet arrival process in an input port.

With the absence of the forwarding table, determining how to map the destination IP address of packets to output ports of the interconnect 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. We use the destination IP address of a packet to determine whether it is a unicast packet or a multicast packet. If the destination IP of a packet is a class D address, then it is a multicast packet. Otherwise, it is a unicast packet. For unicast 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 interconnect 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 a multicast packet, we assume its fanout f (i.e., the number of its destination outputs) is uniformly distributed between 1 and N, then we randomly choose f output ports as its destination outputs.

From Fig. 16(a), we can see that there is a noticeable increase in the packet drop ratio compared to previous traffic patterns. 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, for a switch with $N = 16$, which is typical size for optical interconnects [5], a M-FDLs buffer with approximately $D = 128$ FDL segments for each input port is sufficient to keep the packet drop ratio at a reasonably low level under the most congested Internet traffic condition.

As for average packet latency shown in Fig. 16(b), FIFOMS, MCMS and MLRRMS, perform considerably worse under real Internet traffic than under other traffic models. FIFOMS saturates before the packet arrival rate reaches 0.55, while both MCMS and MLRRMS saturate before the arrival rate reaches 0.9. Meanwhile, we can see that LLMS manages to achieve low average packet latency for the same reason as that under statistical traffic models. Again, LLMS closely matches OUTBUF in both packet drop ratio and average delay, indicating that it is able to achieve near optimal performance under Internet traffic.

From Fig. 17, we can see that prioritized LLMS can deliver high priority traffic with negligible drop ratio and much lower delay under Internet traffic, which has practical implications. For example, suppose a typical network congested with a large volume of IPTV or VoD traffic, prioritized LLMS is able to deliver high priority packets of applications sensitive to latency and packet drop, e.g., an on-line conference, across the network with little interference, which is highly desirable.

VI. Conclusions

In this paper, we have studied multicast scheduling problem for input-buffered optical multicast interconnects. 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 Low Latency Multicast Scheduling (LLMS) Algorithm, the main features of which can be summarized as follows: (1) guarantee a fixed delay upper bound for all transmitted packets; (2) achieve close to optimal average delay and packet drop ratio even under the most congested traffic conditions; (3) enable fast congestion detection and promptly drop output copies with long delay; (4) require much shorter FDL length compared to existing multicast scheduling schemes for all-optical packet interconnects; (5) easily extendable to provide QoS differentiation. We also presented a pipeline and parallel processing architecture and the combination circuit design for LLMS, where it takes each processing module $O(1)$ complexity to finish scheduling in each time slot. The proposed processing architecture does not incur any latency overhead and enables packet scheduling at the line rate, which is essential to optical packet interconnects with ever-increasing port speed. Finally, we evaluated the performance of LLMS through extensive simulation using both statistical traffic models and real Internet traffic traces.

Acknowledgments

This research work was supported in part by the U.S. National Science Foundation under grant number CCF-0915823.

REFERENCES

[1] J. Choi, M. Yoo, and B. Mukherjee, "Efficient video-on-demand streaming for broadband access networks," *IEEE/OSA J. Optical Commun. and Networking,* vol. 2, no. 1, pp. 38–50, Jan. 2010.

[2] Z. Guo, X. Luo, Y. Jin, *et al.*, "Improving resource utilization in hybrid packet/circuit multicasting for IPTV delivery," *2008 OFC.*

[3] Q. Huang and W. Zhong, "A wavelength-routed multicast packet switch with a shared-FDL buffer," *J. Lightwave Technol.,* vol. 28, pp. 2822–2829, Oct. 2010.

[4] Q. Huang and W. D. Zhong, "An optical wavelength-routed multicast packet switch based on multitimeslot multiwavelength conversion," *IEEE Photonic Technol. Lett.*, vol. 20, no. 18, pp. 1518–1520, 2008.

[5] P. Gambini, *et al.*, "Transparent optical packet switching: network architecture and demonstrators in the KEOPS project," *IEEE J. Sel. Areas Commun.*, vol. 16, no. 7, pp. 1245–1259, Sep. 1998.

[6] C. Guillemot, *et al.*, "Transparent optical packet switching: the European ACTS KEOPS project approach," *IEEE J. Lightwave Technol.,*, vol. 16, no. 12, pp. 2117–2134, Dec. 1998.

[7] X. Qin and Y. Yang, "Multicast connection capacity of WDM switching networks with limited wavelength conversion," *IEEE/ACM Trans. Networking*, vol. 12, no. 3, pp. 526–538, June 2004.

[8] C. Zhou and Y. Yang, "Wide-sense nonblocking multicast in a class of regular optical WDM networks," *IEEE Trans. Commun.*, vol. 50, no. 1, pp. 126–134, Jan. 2002.

[9] Y. Wang and Y. Yang, "Multicasting in a class of multicast-capable WDM networks," *J. Lightwave Technol.*, vol. 20, no. 3, pp. 350–359, Mar. 2002.

[10] A. G. P. Rahbar and O. W. W. Yang, "Contention avoidance and resolution schemes in bufferless all-optical packet-switched networks: a survey," *IEEE Commun. Surveys & Tutorials*, vol. 10, no. 4, pp. 94–107, 2008.

[11] Z. Zhang, Z. Guo, and Y. Yang, "Bufferless routing in optical Gaussian macrochip interconnect," to appear in *IEEE Trans. Computers*, 2014.

[12] Z. Zhang and Y. Yang, "Performance analysis of optical packet switches enhanced with electronic buffering," in *Proc. 2009 IEEE International Parallel and Distributed Processing Symposium.*

[13] Z. Guo, Z. Zhang, and Y. Yang, "Performance modeling of hybrid optical packet switches with shared buffer," in *Proc. 2011 IEEE INFOCOM*, pp. 1692–1700.

[14] Z. Guo and Y. Yang, "High speed multicast scheduling in hybrid optical packet switches with guaranteed latency," *IEEE Trans. Computers,* vol. 62, no. 10, pp. 1972–1987, Oct. 2013.

[15] L. Liu and Y. Yang, "Packet scheduling in a low-latency optical switch with wavelength division multiplexing and electronic buffer," in *Proc. 2011 IEEE INFOCOM*, pp. 2759–2767.

[16] H. Yang and S. J. B. Yoo, "All-optical variable buffering strategies and switch fabric architectures for future all-optical data routers," *IEEE J. Lightwave Technol.,* vol. 23, pp. 3321–3330, Oct. 2005.

[17] D. Pan and Y. Yang, "FIFO-based multicast scheduling algorithm for virtual output queued packet switches," *IEEE Trans. Computers,* vol. 54, no. 10, pp. 1283–1297, Oct. 2005.

[18] B. Prabhakar, N. McKeown, and R. Ahuja, "Multicast scheduling for input-queued switches," *IEEE J. Sel. Areas Commun.* vol. 15, no. 5, pp. 855–866, June 1997.

[19] 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 Trans. Networking,* vol. 8, no. 4, pp. 517–525, 2000.

[20] Y. Hao, S. Ruepp, M. S. Berger, and L. Dittmann, "Integration of look-ahead multicast and unicast scheduling for input-queued cell switches," *IEEE HPSR 2012* pp. 24–27.

[21] Z. Guo and Y. Yang, "Pipelining multicast scheduling in all-optical packet switches with delay guarantee," *2011 International Teletraffic Congress.*

[22] H. Harai and M. Murata, "High-speed buffer management for 40 Gb/s-based photonic packet switches," *IEEE/ACM Trans. Networking,* vol. 14, pp. 191–204, Feb. 2006.

[23] R. Wang, J. Zhang, F. Guo, and K. Long, "An effective buffering architecture for optical packet switching networks," *IEEE/OSA J. Photonic Network Commun.,* vol. 16, pp. 239–243, Jun. 2008.

[24] N. McKeown, "The iSLIP scheduling algorithm for input-queued switches," *IEEE/ACM Trans. Networking*, vol. 7, no. 4, pp. 188–201, Apr. 1999.

[25] A. Prakash, S. Sharif, and A. Aziz, "An $O(log^2 N)$ parallel algorithm for output queueing," in *Proc. 2002 IEEE INFOCOM*, pp. 1623–1629.

[26] H.Furukawa, H. Harai, M. Ohta, and N. Wada, "Implementation of high-speed buffer management for asynchronous variable-length optical packet switch," *2010 OFC.*

[27] H. Furukawa, N. Wada, H. Harai, and T. Miyazaki, "Development of a 640-Gbit"ms"mport optical packet switch prototype based on wide-colored optical packet technology," *IEEE/OSA J. Optical Commun. Networking,* vol. 1, no. 3, pp. B30–B39, Aug. 2009.

[28] 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," *IEEE J. Lightwave Technol.,* vol. 23, pp. 211–218, Jan. 2005.

[29] A. Wonfor, H. Wang, R. V. Penty, and I. H. White, "Large port count high-speed optical switch fabric for use within datacenters [invited]," *IEEE/OSA J. Optical Commun. and Networking,* vol. 3, no. 8, pp. A32–A39, Aug. 2011.

[30] S. Yu, S.-C. Lee, O. Ansell, and R. Varrazza, "Lossless optical packet multicast using active vertical coupler based optical crosspoint switch matrix," *IEEE J. Lightwave Technol.,* vol. 23, no. 10, pp. 2984–2992, Oct. 2005.

[31] H. Furukawa, *et al.*, "A 31-FDL buffer based on trees of 1x8 PLZT optical switches," *European Conference on Optical Commun.,* vol. 4, no. 1, pp. 1–2, Sep. 2006.

[32] L. Liu and Y. Yang, "Achieving 100% throughput in input-buffered WDM optical packet interconnects," *IEEE Trans. Parallel Distrib. Syst.,* vol. 22, no. 2, pp. 273–286, Feb. 2011.

[33] "BEEcube's BEE4 FPGA prototyping platform." Available: http://www.beecube.com/products/BEE4.asp.

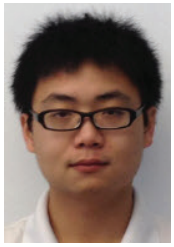
[34] H. Harai and M. Murata, "Optical fiber-delay-line buffer management in output-buffered photonic packet switch to support service differentiation," *IEEE J. Sel. Areas Commun.,* vol. 24, no. 8, pp. 108–116, Aug. 2006.

[35] 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 Trans. Networking*, vol. 11, no. 3, pp. 465–477, June 2003.

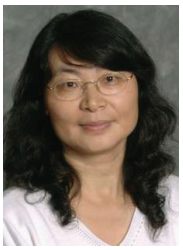
[36] A. Bianco, *et al.*, "On the number of input queues to efficiently support multicast traffic in input queued switches," *2003 IEEE High Performance Switching and Routing.*

[37] K. Claffy, D. Andersen, and P. Hick, "The CAIDA anonymized 2010 Internet traces."
Available: http://www.caida.org/data/passive/passive_2010_dataset.xml

[38] A. Kos, P. Homan, and J. Bester, "Performance evaluation of a synchronous bulk packet switch under real traffic conditions," *IEICE Trans. Commun.*, vol. E86-B, May 2003.



Zhiyang Guo received the B.Eng degree from Shanghai Jiaotong University, Shanghai, China, in 2008. Since then he has been studying towards the PhD. degree in the Department of Electrical and Computer Engineering, Stony Brook University, New York. His research interests include optical switching, QoS guaranteed scheduling and data center networks. He is a student member of the IEEE.



Yuanyuan Yang received the BEng and MS degrees in computer science and engineering from Tsinghua University, Beijing, China, and the MSE and PhD degrees in computer science from Johns Hopkins University, Baltimore, Maryland. She is a professor of computer engineering and computer science at Stony Brook University, New York, and the director of Communications and Devices Division at New York State Center of Excellence in Wireless and Information Technology (CEWIT). Her research interests include wireless networks, data center networks, optical networks and high-speed networks. She has published over 280 papers in major journals and refereed conference proceedings and holds seven US patents in these areas. She is currently the Associate Editor-in-Chief for the IEEE TRANSACTIONS ON COMPUTERS and an Associate Editor for the *Journal of Parallel and Distributed Computing*. She has served as an Associate Editor for the IEEE TRANSACTIONS ON COMPUTERS and IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS. She has served as a general chair, program chair, or vice chair for several major conferences and a program committee member for numerous conferences. She is an IEEE Fellow.