A Modular, Scalable, Extensible, and Transparent Optical Packet Buffer

Benjamin A. Small, *Member, IEEE, Member, OSA*, Assaf Shacham, *Student Member, IEEE*, and Keren Bergman, *Member, IEEE, Fellow, OSA*

***Abstract*—We introduce a novel optical packet switching buffer architecture that is composed of multiple building-block modules, allowing for a large degree of scalability. The buffer supports independent and simultaneous read and write processes without packet rejection or misordering and can be considered a fully functional packet buffer. It can easily be programmed to support two prioritization schemes: first-in first-out (FIFO) and last-in first-out (LIFO). Because the system leverages semiconductor optical amplifiers as switching elements, wideband packets can be routed transparently. The operation of the system is discussed with illustrative packet sequences, which are then verified on an actual implementation composed of conventional fiber-optic componentry.**

***Index Terms*—Buffers, optical fiber communication, photonic switching systems.**

I. Introduction

BUFFERING in optical packet switching (OPS) networks is a challenging problem that has been studied for some time. For nearly all network topologies, packet buffering has the potential to significantly improve network acceptance rates and thereby increase overall throughput and efficiency [1]. However, the physical nature of optical signals prohibits the implementation of optical buffers in a manner similar to conventional electronic ones. Because, at present, buffer schemes that require slowing the speed of light in exotic materials have their own distinct challenges [2], schemes that instead use long loops of conventional optical fiber to delay the signals provide more opportunities for successful implementation in current systems.

Numerous architectures have been proposed and implemented. We detail a new buffer architecture that is modular, scalable, extensible, and transparent and therefore provides significantly improved performance over other designs. This architecture was first introduced briefly in [3], and we provide a complete description here.

Two important characteristics of a successful OPS buffer architecture are 1) physical-layer transparency, or conservation of optical power and signal quality without introducing significant distortions, and 2) the ability of the buffer structure to support independent and simultaneous read and write processes without requiring modifications to the surrounding subsystems. The implementation detailed here is the first to fully address both concerns.

Buffer architectures based on cascaded fiber delay line (FDL) modules and parallel FDL arrays have been proposed [4]–[10] and implemented [8], [11]–[13]. However, a careful examination of complexities associated with the actual implementation of these structures reveals that read and write processes cannot be executed independently under physical timing requirements. Parallel FDL structures [5]–[9] and other architectures that allow packets to be stored for a predetermined amount of time, such as in [12], require advanced knowledge of the packet's duration in the buffer and, hence, do not support truly independent read and write processes. Schemes based on cascaded FDLs [4], [8]–[11] are difficult to construct in a way that maintains physically realizable timing and signaling necessary to serve multiple packets simultaneously. The architecture proposed in [10] cleverly addresses these concerns but does not guarantee packet arrival; some packets are dropped or routed incorrectly.

Many of these schemes only support an independent treatment of the stored packets. That is, first-come–first-served (FCFS) or first-in–first-out (FIFO) prioritization is not easily supported. However, having well-defined behavior is essential for maintaining fairness and ordering in OPS systems [1], [14]. The architecture presented here meets all of these requirements, supporting both FIFO and last-in–first-out (LIFO) prioritization. Furthermore, with minor adjustments, the buffer can support a priority queue implementation and a number of other schemes.

Our architecture behaves like a complete buffer, in the sense that designers of electronic packet-switched networks and networking protocols expect: Packet read and write operations are executed independently, and the whole system manages itself. It is not meant to provide retiming functions and fractional delay offsets, but instead operates in an entirely time-slotted manner. The buffer architecture maintains scalability and extensibility because of its modular structure. It also guarantees complete and independent read and write functionality without packet rejection, misordering, or loss (except in the case of overflow) and can be dynamically extended to increase capacity.

In the subsequent sections, we first discuss the functionality and operation of the proposed architecture. Routing examples are also provided to illustrate this functionality. We then present a network performance analysis of the buffer architecture, given its unique internal routing behavior. Finally, the implementation of a prototype system is detailed, and we demonstrate independent read and write operations on that implementation;

Manuscript received November 1, 2006; revised December 14, 2006. This work was supported by the U.S. Department of Defense under Subcontract B-12-664.

The authors are with the Department of Electrical Engineering, Columbia University, New York, NY 10027 USA (e-mail: bas@ee.columbia.edu; assaf@ee.columbia.edu; bergman@ee.columbia.edu).

Digital Object Identifier 10.1109/JLT.2007.891176

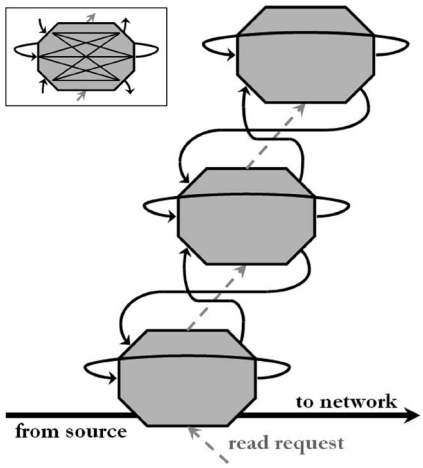

Fig. 1. Cascaded buffer architecture with intraconnecting optical fibers for packets and electronic cables for *read request* signals. (Inset) Full 3 × 3 cross-connect, of which a subset is used in each module.

power penalty measurements are also taken as a measure of the distortions incurred on buffered packets.

II. Functionality

A. Overview

The optical packet buffer is composed of independent and identical building-block modules that are cascaded to form a complete buffer implementation. When the entire system is assembled, it has one input port for optical packets and one output port, in addition to the input and output for a *read request* signal, which is transmitted from the switching network or router. The total capacity of the buffer equals the number of modules it contains since each contains one FDL (see Section II-B).

The individual building-block modules are all preprogrammed with a particular routing logic, which can allow the whole system to behave either as a FIFO (or queue) or as a LIFO (or stack) buffer. This behavior is complete and fully functional, and the buffer is transparent to the surrounding OPS system. Packets are simply stored in the structure and released upon receiving a *read request* signal; these processes are entirely independent.

The buffer implementation has no central arbitration of any kind and requires no central management; all routing and management is entirely distributed, with each module functioning individually. After the root module has been inserted on a pathway between a packet source and the OPS system, an arbitrary number of modules can be cascaded laterally from it (Fig. 1). Because these modules are all self-contained and independent, it is not necessary to configure or customize the assembled buffer structure further.

B. Module Structure

The individual building-block modules that make up the buffer structure have two input ports and two output ports, one each connected to adjacent modules. The modules also contain an FDL, so that each can hold a single packet when necessary.

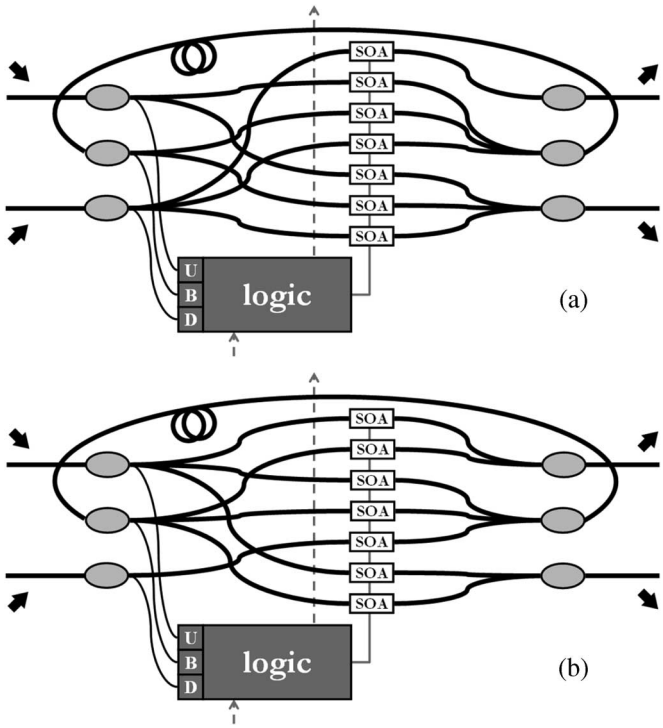

Fig. 2. Schematic of module for (a) FIFO implementation and (b) LIFO implementation (except root module) with ovals for couplers. L, B, and D for their respective low-speed receivers; dashed lines for *read request* signals.

In order to provide the optical pathways to allow for packets to propagate to and from either of the adjacent modules or the internal buffer (i.e., FDL), a subset of a 3 × 3 cross-connect is used (Fig. 1 inset; Fig. 2).

In addition, in order to maintain functional modularity, each module has *read request* signal inputs and outputs that are connected to the two adjacent modules. The individual modules execute routing logic based only on the incoming *read request* signal and the presence or absence of packets on each of the three inputs (including the internal buffer); the routing logic itself is memoryless (i.e., combinatorial logic). This kind of modular and distributed structure leverages the same paradigm as [15]–[18]. Table I details the truth table needed to implement the routing logic in each module as required for the queue (FIFO) functionality; Table II implements a stack (LIFO).

For each buffer configuration, only a subset of the complete 3 × 3 cross-connect is required. The modules are arranged so that, by the nomenclature used here, packets enter and exit the bottom module and are passed upward for storage and downward to exit in response to a *read request* signal. The lower modules in the structure remain occupied, whereas the upper ones are empty until more packets enter the buffer. Overflow occurs when each module contains a packet, and then, one more packet is injected into the buffer without a simultaneous *read request* signal; in this case, only a single packet is lost (the newest in the FIFO; the oldest in the LIFO), whereas the rest of the buffer remains intact and coherent.

Although each module can store only one packet at a time in its FLD, packets can pass packets between adjacent modules without limitation. This allows the buffer to maintain the ordering required by the FIFO or LIFO prioritization schemes.

TABLE I
TRUTH TABLE FOR FIFO (QUEUE) OPERATION

D	B	U	R	D2D	D2B	D2U	B2D	B2B	B2U	U2D	U2B	U2U	RO	
0	0	0	X											*empty queue*
1	0	0	0		1									*write*
0	1	0	0					1						*hold*
1	1	0	0			1		1						*subsequent write*
0	0	1	0								1			*after read*
1	0	1	0			1					1			*write after read*
X	1	1	X	*inaccessible state*										
1	0	0	1	1										*read and write (last module)*
0	1	0	1				1						1	*read*
1	1	0	1			1	1						1	*read and write*
0	0	1	1							1			1	*subsequent read*
1	0	1	1			1				1			1	*subsequent read and write (root module)*

TABLE II
TRUTH TABLE FOR LIFO (STACK) OPERATION

D	B	U	R	D2D	D2B	D2U	B2D	B2B	B2U	U2D	U2B	U2U	RO	
0	0	0	X											*empty stack*
1	0	0	0		1									*write*
0	1	0	0					1						*hold*
1	1	0	0		1				1					*subsequent write*
0	0	1	0								1			*after read*
1	0	1	0		1							1		*write after read*
X	1	1	X	*inaccessible state*										
1	0	0	1	1										*read and write (root module)*
0	1	0	1				1						1	*read*
1	1	0	1	1				1						*read and write (root module)*
0	0	1	1							1			1	*subsequent read*
1	0	1	1	1							1			*subsequent read and write (root module)*

The behavior of a module that implements these schemes is very similar; read and write operations are nearly identical between the two. Differences occur only when managing the order of the packets within the buffer; this depends on which prioritization scheme is used.

For the FIFO prioritization (Table I), when a packet enters a module (via the down or "D" port), it is passed up to the next empty module (via the up or "U" port), ensuring that older packets are the first to exit the queue. When a packet receives a cascaded *read request* signal, it sends its packet down to the previous module and propagates the *read request* signal up the chain to the next module (RO), which results in another packet being sent down to it. This scheme ensures that all packets are placed in adjacent modules, so that the buffer can be read out as quickly and efficiently as possible. When a *read request* signal is sent to an empty module, the *read request* signal does not need to propagate further since that module is necessarily the last used module in the chain. When a packet enters an unoccupied module at the same time that a *read request* signal is received, it is immediately sent down to the previous module. When no *read request* signals are present, packets are circulated within the FLDs of each module to hold the state of the buffer.

The LIFO prioritization (Table II) is very similar to the one described. The biggest difference is that new packets are not sent up to the end of the buffer; they are instead stored in the first module, pushing all other packets further up the stack. Thus, the first packet to exit the stack on a *read request* signal is actually the newest one.

For both schemes, it is impossible to have packets in both the buffer and coming from the next module simultaneously since packets are requested from the next module (via a propagated *read request* signal) during a read process, which first requires the buffer to be emptied.

The functionality of the complete buffer structure based on modules that execute the truth tables found in Tables I and II has been verified with a specially designed model in Verilog (a hardware description language typically used for digital systems). Illustrations of the behavior of the buffer under both prioritization schemes, which discuss the specific operations in more detail, are provided later in the text.

Finally, it is important to note that neither scheme requires all nine gates from the 3×3 cross-connect structure. For the queue, the buffer-to-up (B2U) and up-to-up gates are not needed for any module [Fig. 2(a)]. For the stack, the down-to-up (D2U) gate is not needed by any module; only the root module uses the down-to-down (D2D) one [Fig. 2(b)]. These gates are unnecessary since the specified routing logic does not utilize certain transitions for each prioritization scheme (empty columns in Table I or II).

C. Illustrations of Buffer Operation

In order to provide an instructive illustration of the buffer architecture, consider a buffer that contains two modules. Fig. 3 traces the paths taken by the packets for a particular sequence of injected packets and *read request* signals. Fig. 3(a) depicts the

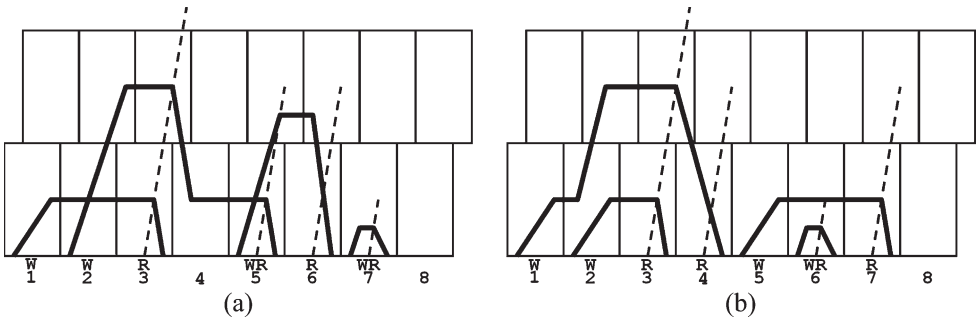

Fig. 3. Timing diagrams for (a) queue (FIFO) and (b) stack (LIFO) implementations that process sequences of read (R) and write (W) operations according to the rules in Tables I and II. With packets (sold lines) and *read request* signals (dashed lines) entering from the bottom edge, the lower rectangles represent the root module, and the upper ones represent the second module. Time advances from left to right, with the second module lagging slightly behind the root one, as mentioned in Section IV.

queue (FIFO) functionality, whereas Fig. 3(b) is for the stack (LIFO) functionality. The reader can verify that the overall behavior of the buffer results from the correct execution of the appropriate truth table at each module. Clearly, this modular behavior could be extended to larger buffer systems as well.

1) Queue (FIFO): During the first timeslot, a packet is offered to the root module (i.e., write operation), and since the buffer is empty and no *read request* signal is received, the packet is simply directed into the internal buffer with the D2B gate.

Another packet is injected at the second timeslot, and now, because the root module's buffer is already occupied, the root module sends the packet up to the next module with D2U. That module is empty and therefore behaves in exactly the same way as the root module did during the first timeslot: The packet is sent to the buffer with D2B. Meanwhile, the first packet is held in the root module's FDL by B2B.

Next is a simple read operation. To maintain FIFO ordering, the packet in the root module's FDL, which is necessarily the oldest, egresses from the buffer structure by the B2D gate. The root module also propagates the *read request* signal to the next module, which behaves in an identical way: It sends its packet down via B2D. This packet will reach the root module (now into the up port) at the beginning of the next timeslot.

For the fourth timeslot, no read or write operation occurs; thus, the packet that is just now entering the root module through the up port is directed to the FDL with U2B. (It is important to note that read and write operations could be correctly performed during this timeslot as well, independent of other activity within the modules. In those cases, the logic on the "subsequent read" or "write after read" lines from Table I are executed.)

During the fifth timeslot, a simultaneous read–write operation occurs. To maintain FIFO prioritization, the packet in the root module's FDL, which has been in the buffer system for three timeslots now, is allowed to egress via B2D. Simultaneously and independently, the new packet is sent up to the next module with D2U at the same time that the *read request* signal propagates upward. The second module encounters both the incoming packet and this signal; thus, the packet is sent back to the root module with the second module's D2D gate. Although this functionality seems wasteful, the effect is to hold this packet between the two modules for a single timeslot, buffering it in a manner similar to an FDL, since the packet enters the root module (through the up port) only at the beginning of the next timeslot.

The sixth timeslot contains only a read operation. Because the aforementioned packet comes from the second module, the root module enables the U2D gate, allowing the packet to egress from the queue. The *read request* signal is also propagated upward since the root module has no information about whether the next module is empty or not.

The seventh timeslot contains another simultaneous read–write operation, but because the buffer structure is empty, the execution differs from the fifth timeslot. The root module instead behaves in the same way as the second module did. Its D2D gate routes the packet back out of the queue. Because the root module's buffer was empty, none of the other modules could contain packets either, and propagation of the *read request* signal is unnecessary.

Finally, on the eighth timeslot, the buffer rests.

2) Stack (LIFO): The root module receives a packet during the first time slot. Because the stack is empty, this packet is to be stored in the root module's FDL and is directed there by the D2B gate.

Another packet ingresses during the second timeslot. Now, because the root module already contains a packet, that first packet is sent up the stack to the second module with B2U. The root module and the second module both receive their new packets and store them in their respective FDLs with the D2B gates, just as the root module did in the first timeslot (cf. first two timeslots of FIFO illustration).

During the third timeslot, a *read request* is received. The root module allows its packet to egress the buffer with B2D. The *read request* signal is propagated to the second module, which also enables the B2D gate.

Another read operation occurs during the fourth timeslot. In this case, the root module allows the packet that was sent from the second module to egress with the U2D gate. The *read request* signal is propagated to the second module, but because that module is empty, the signal is curtailed.

The fifth timeslot is identical to the first.

In the sixth timeslot, however, a simultaneous read–write operation is executed. While the root module has a packet contained in its FDL by B2B, a second packet is written and immediately read, passing only through the D2D gate.

The packets are handled independently in the root module for this operation, and the root module is the only one in the stack implementation that must execute this particular functionality. Moreover, because no changes to the buffer structure occur outside of the root module, propagating the *read request* signal further is unnecessary. (Note that U2B can also be used simultaneously with D2D.)

The seventh timeslot requires that the root module read a packet from its FDL, just as it did during the third timeslot.

Finally, on the eighth timeslot, the buffer rests.

III. Analysis

It has been demonstrated that the architecture described here allows for a complete and fully functional implementation of a FIFO or LIFO buffer. This functionality is consistent with the propositions of conventional queuing theory. However, the physical execution of the buffer structure is unique and must be analyzed in the context of OPS. For many OPS systems, the number of switching gates or amplifiers traversed by a particular optical packet is an important metric for estimating signal quality. This is a result of the unfortunate fact that optical packets experience signal degradation very easily when they encounter these devices. This section provides equations that count the number of switching elements traversed by packets within the buffer architecture.

As mentioned earlier, the implemented optical packet buffer architecture is able to execute read and write processes independently and simultaneously. Packets are stored in the structure, and service order is well defined; thus, the architecture meets Erlang's formal definition of a queue, as used in the discipline of queuing theory [19], [20]. In most analyses, traffic is assumed to arrive and to be serviced with uniformly distributed processes; that is, there is Bernoulli traffic and similar statistics for the *read request* signals. This scenario is classified as an M/M/1 queue, for which significant theoretical background exists [19]–[21]. Naturally, application-specific traffic patterns should also be used to analyze the buffer capacity more precisely, but we provide an approximate analysis here based on these common assumptions.

First, load factor ρ is defined by convention as

$$\rho = \lambda/\mu \tag{1}$$

where λ is the mean arrival time (i.e., arrival rate $1/\lambda$), and μ is the mean service time (service rate $1/\mu$). Markov chain analysis reveals that the mean number of packets in the buffer at a given time (the buffer occupancy) is then

$$\overline{N} = \frac{\rho}{1-\rho}. \tag{2}$$

Applying Little's law [21], the mean number of timeslots that a packet spends in the buffer is

$$\overline{T} = \frac{\overline{N}}{\lambda} = \frac{1}{\lambda}\frac{\rho}{1-\rho} = \frac{1}{\mu-\lambda} \tag{3}$$

which is the universal result for an M/M/1 queue. (Technically, when packets arrive on discrete timeslots, the queue is of type Geom/Geom/1 with a discrete uniform distribution instead of a continuous uniform one; however, the important results are identical to the M/M/1 case [20].)

For the queue (FIFO) implementation, packets must first ascend the cascade of modules to the first unoccupied one. This process requires $N+1$ semiconductor optical amplifier (SOA) hops if the buffer already contains N packets. (Even if the buffer is empty, a single SOA hop is required to enter the root module.) Then, the packet is held for a total of T timeslots within the buffer, possibly in different modules; each timeslot requires a single SOA hop. Because the packet descends the cascade of modules throughout its duration in the buffer, the last module in which it is held for a timeslot is necessarily the root module. Thus, for the queue (FIFO) implementation, the required number of SOA hops for a particular packet is

$$S = N + T + 1. \tag{4}$$

This result can be confirmed by reexamining the illustration in Section II-C. The first packet traverses three SOAs; the second, five; the third, three; and the fourth, one.

Finally, using the universal definitions (1)–(3) in (4), and in accordance with the law of large numbers, the mean number of SOAs through which packets propagate in the queue is

$$\overline{S} = \overline{N} + \overline{T} + 1 = \frac{\rho}{1-\rho}\left(1+\frac{1}{\lambda}\right) + 1 = \frac{\mu+1}{\mu-\lambda}. \tag{5}$$

For the stack (LIFO) implementation, because packets are pushed up and down the buffer contiguously, the number of SOA hops S required for a particular packet is just

$$S = T + 1 \tag{6}$$

independent of the number of packets that preceded it. Referring again to the illustrations in Section II-C, but this time for the LIFO implementation, the first packet encounters four SOAs; the second, two; the third, three; and the fourth, one. By the same argument as is used to obtain (5), the mean number of SOAs through which packets propagate in the stack is

$$\overline{S} = \overline{T} + 1 = \frac{\mu-\lambda+1}{\mu-\lambda}. \tag{7}$$

IV. Implementation

Buffer modules that implement the desired functionality can be assembled from conventional photonic and fiber-optic components in a relatively straightforward manner (Fig. 4). Electrically controlled SOA switching gates are used to implement the appropriate subset of the 3×3 structure. Standard fiber-optic couplers combine and divide the necessary pathways; no optical filters are necessary. A small amount of power is tapped from each of the three inputs and directed to low-speed optical receivers in order to determine the presence or absence of a packet (Fig. 2). These three signals, in addition to an electronic *read request* signal transmitted over microwave cables, are the only inputs used to execute the routing decision by a standard

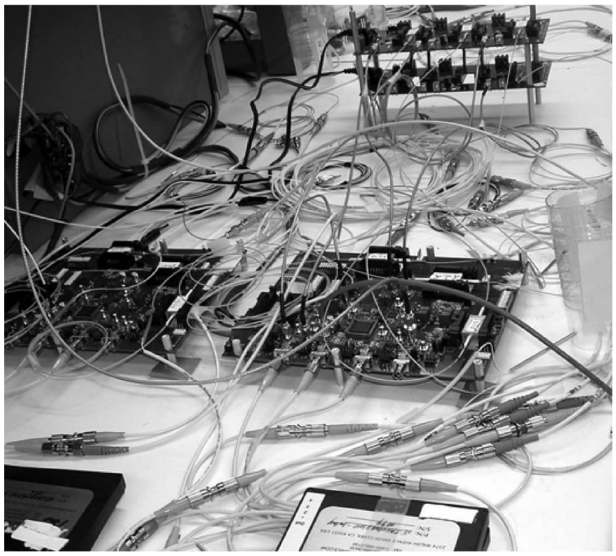

Fig. 4. Photograph of implemented buffer system; the two circuit boards (midground) contain SOAs and the CPLDs for the two modules, with additional SOAs (further background) and passive fiber-optic components (foreground).

commercial complex programmable logic device (CPLD), as specified by Tables I and II, or by any similar table that could implement other priority schemes.

While the routing decision is computed by the CPLD, the packets are held in a short length of fiber. Then, the appropriate SOAs are enabled, so that the packet or packets are routed to the correct module output. An electronic *read request* signal can also propagate to the next module in the buffer. This part of the node latency is approximately 22 ns. Packets that then ascend the buffer propagate through a fiber delay of approximately 15 ns before reaching the next module, those that descend the buffer face a 65 ns delay, and those that remain buffered within the module are held in an FDL with a latency of approximately 80 ns. With this timing arrangement, the implemented system functions correctly with 102-ns packet timeslots. The packets themselves are approximately 90 ns long, allowing for 12 ns of deadtime between the packets, which is more than enough to accommodate the SOA gate switching time of less than 2 ns. This implementation can be adjusted to support other packet slot times by increasing or decreasing the length of fiber in the modules' FDLs.

All packets remain in the optical domain throughout their lifetime in the buffer, and because only wideband components (i.e., SOAs and couplers; no optical filters) are used in the optical paths, the buffer can transparently route packets that contain wavelengths over almost the entire C-band. These packets can also have a multiple-wavelength (wavelength-striped) format, as is found in [15], [16], and [22].

The gain of the SOAs is set to exactly compensate for the coupler losses found in their particular branch of the 3×3 cross-connect. Because these losses depend on which branches are required for a particular module implementation, the SOAs are set to deliver between 6 and 11 dB of gain, which requires between 45 and 75 mA of drive current. The net gain or loss incurred on each packet payload can be kept to less than about 0.5 dB, as in [15], [16], and [24].

A fully operational two-module buffer is constructed to experimentally verify functionality and to perform physical-layer investigations. The two modules are completely independent, as is specified in Section II, and therefore, this setup can easily be extended to include more modules while maintaining consistent operation.

V. Experimental Results

In order to verify the behavior of the architecture, a routing experiment is performed on both the FIFO and LIFO implementations of the buffer. These results confirm for the implemented buffer that read and write operations occur independently and that the desired ordering is maintained. Furthermore, power penalty measurements are taken on packets that traverse the whole buffer structure, propagating through varying numbers of SOAs. These data show the minor signal degradation induced by the buffer, which is in support of its overall physical-layer transparency and scalability.

A. Routing Verification

In order to verify the overall routing behavior of the buffer, numerous trials were performed. Examples, including one each for the FIFO or queue [Fig. 5(a)] and LIFO or stack [Fig. 5(b)] implementations, are provided here. Packets are offered to the buffer, along with *read request* signals, in a manner identical to the illustrations provided in Fig. 3 and detailed in Section II-C. The 90-ns packets, which are spaced in 102-ns timeslots, contain a 10-Gb/s payload in addition to a unique 7-bit packet label, so that individual packets can easily be identified during the experiment. Incoming packets have a total average power of approximately -10 dBm; this keeps the SOAs well within the linear operating regime [24]. The implemented two-module buffer also includes internal packet and *read request* signal monitoring for troubleshooting and verification. A careful examination of the label waveforms in Fig. 5 reveals that packets are kept in FCFS order for the FIFO implementation and last-come–first-served (LCFS) order for the LIFO implementation, exactly as intended. The second packet in the FIFO sequence and the first packet in the LIFO are both delayed for 330 ns (three full timeslots plus processing), which is the longest delay experienced for the examples provided.

B. Power Penalty

In order to quantify the physical-layer transparency of the SOA-based buffer, power penalty measurements are taken on packets that encounter different numbers of SOAs within the structure. Using the packets from the routing verification, which contain 10-Gb/s payloads, bit-error-rate (BER) values are measured (Fig. 6) with a tester that is synchronized to the packet generator. With the FIFO or queue implementation, the first and third packets encounter two SOAs: the second, five; and the fourth, one, as discussed previously. From this selection of routing options, it can be deduced that each SOA hop introduces approximately 0.4 dB of power penalty on average. This result is similar to those for other simplistic SOA-based OPS nodes and modules [11], [15], [17], [23], [24].

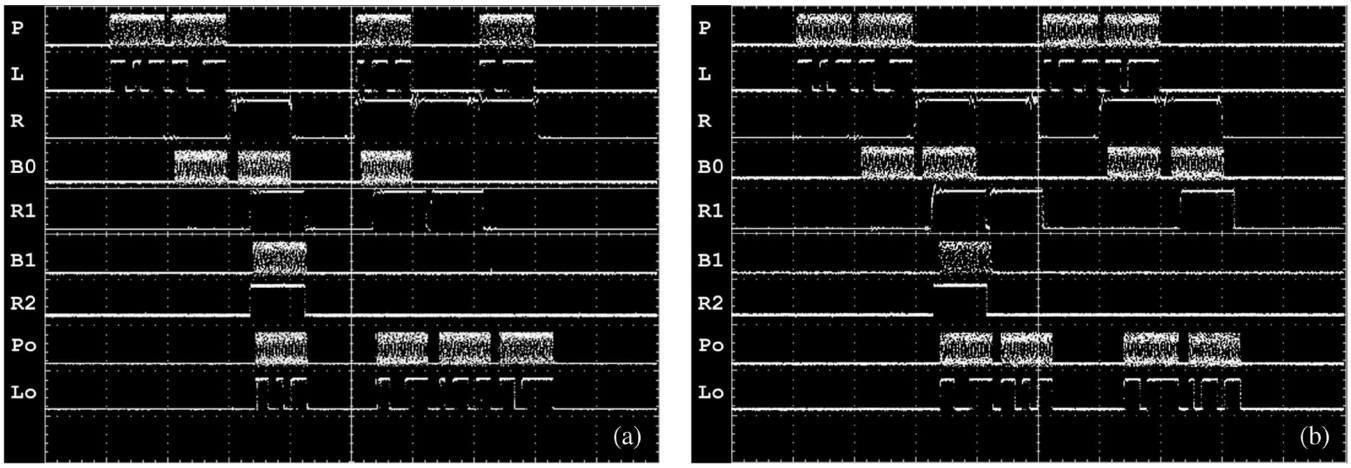

Fig. 5. Waveforms for routing verification. (a) FIFO implementation. (b) LIFO implementation. Packets are introduced to the root module of the buffer, containing both payloads (P) and labels (L), along with *read request* signals (R). The correct sequence of packets can be seen exiting the module (Po and Lo) in response to the *read request* signals. Within the buffer, each module's FDL is monitored (B0 and B1), as are the propagated *read request* signals (R1 and R2). Note the order of the labels on the inputs and outputs, which are (a) FCFS or (b) LCFS based on the *read request* signals (cf. Fig. 3).

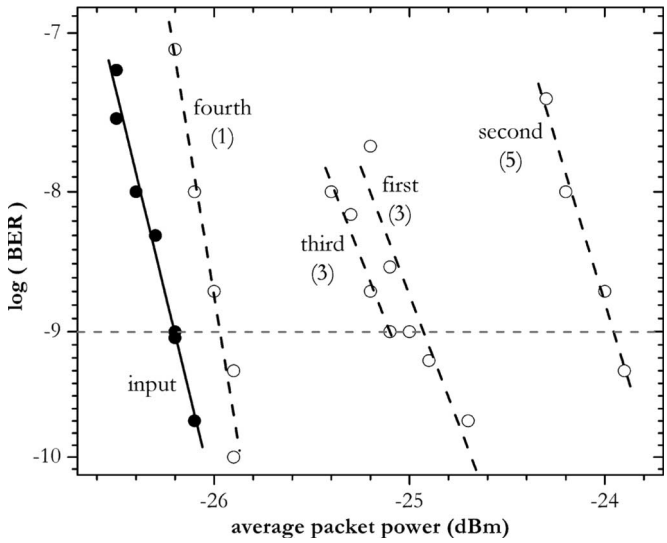

Fig. 6. Power penalty curves for each packet in the illustrative sequence for the FIFO, which are labeled in order with the number of SOA hops parenthetically.

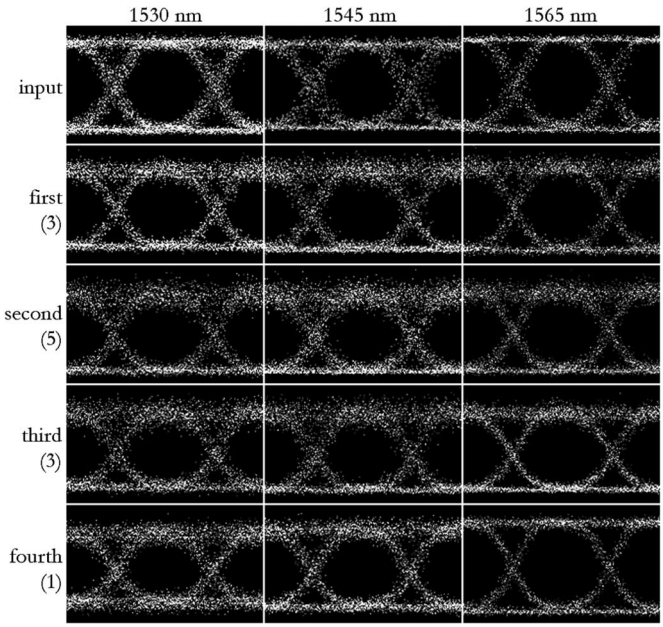

Fig. 7. Eye diagrams of payloads with BER thresholds of 10^{-12} for payloads at three wavelengths (columns) at the buffer input and for the four packets in the illustrative sequence for the FIFO implementation with the number of SOA hops parenthetically (rows).

Also, in order to demonstrate the wideband transparency of the buffer implementation, eye diagrams for different payload wavelengths are presented as well (Fig. 7). All payloads can easily attain 10^{-12} BERs, and no error floor is observed.

VI. Conclusion

A novel OPS buffer architecture is introduced and discussed. This architecture supports independent read and write processes and manages itself in a way that leaves the rest of the network unaffected. It also maintains physical-layer transparency by introducing minimal signal degradation on wideband optical payloads. The buffer is composed of multiple identical building-block modules, each of which contains a single FDL for holding packets. This design is scalable and extensible, having the ability to implement a buffer of any size and support at least two distinct prioritization schemes (i.e., FIFO and LIFO). An algebraic analysis of the buffer architecture is also included. The actual implementation of a two-module buffer illustrates the feasibility of this architecture and attests to its elegance.

References

[1] W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. San Francisco, CA: Morgan Kaufmann, 2004.

[2] R. S. Tucker, P. C. Ku, and C. J. Chang-Hasnain, "Slow-light optical buffers: Capabilities and fundamental limitations," *J. Lightw. Technol.*, vol. 23, no. 12, pp. 4046–4066, Dec. 2005.

[3] A. Shacham, B. A. Small, and K. Bergman, "A novel optical buffer architecture for optical packet switching routers," presented at the 32nd Eur. Conf. Optical Commun., Cannes, France, Sep. 2006, Paper We1.4.4.

[4] D. K. Hunter and I. Andonovic, "Optical contention resolution and buffering module for ATM networks," *Electron. Lett.*, vol. 29, no. 3, pp. 280–281, Feb. 1993.

[5] R. Langenhorst, M. Eiselt, W. Pieper, G. Großkopf, R. Ludwig, L. Küller, E. Dietrich, and H. G. Weber, "Fiber loop optical buffer," *J. Lightw. Technol.*, vol. 14, no. 3, pp. 324–335, Mar. 1996.
[6] R. S. Tucker and W. D. Zhong, "A new wavelength-routed photonic packet buffer combining traveling delay lines with delay-line loops," *J. Lightw. Technol.*, vol. 19, no. 8, pp. 1085–1092, Aug. 2001.
[7] W. A. Vanderbauwhede and H. Novella, "A multiexit recirculating optical packet buffer," *IEEE Photon. Technol. Lett.*, vol. 17, no. 8, pp. 1749–1751, Aug. 2005.
[8] I. Chlamtac, A. Fumagalli, L. G. Kazovsky, P. Melman, W. H. Nelson, P. Poggiolini, M. Cerisola, A. N. M. M. Choudhury, T. K. Fong, R. T. Hofmeister, C.-L. Lu, A. Mekkittikul, D. J. M. Sabido, IX, C.-J. Suh, and E. W. M. Wong, "CORD: Contention resolution by delay lines," *IEEE J. Sel. Areas Commun.*, vol. 14, no. 5, pp. 1014–1029, Jun. 1996.
[9] C.-S. Chang, Y.-T. Chen, and D. S. Lee, "Constructions of optical FIFO queues," *IEEE Trans. Inf. Theory*, vol. 52, no. 6, pp. 2838–2843, Jun. 2006.
[10] D. K. Hunter, D. Cotter, R. B. Ahmad, W. D. Cornwell, T. H. Gilfedder, P. J. Legg, and I. Andonovic, "2 × 2 buffered switch fabrics for traffic routing, merging, and shaping in photonic cell networks," *J. Lightw. Technol.*, vol. 15, no. 1, pp. 86–101, Jan. 1997.
[11] E. F. Burmeister and J. E. Bowers, "Integrated gate matrix switch for optical packet buffering," *IEEE Photon. Technol. Lett.*, vol. 18, no. 1, pp. 103–105, Jan. 2006.
[12] Y.-K. Yeo, J. Yu, and G.-K. Chang, "A dynamically reconfigurable folded-path time delay buffer for optical packet switching," *IEEE Photon. Technol. Lett.*, vol. 16, no. 11, pp. 2559–2561, Nov. 2004.
[13] J. Spring and R. S. Tucker, "Photonic 2 × 2 packet switch with input buffers," *Electron. Lett.*, vol. 29, no. 3, pp. 284–285, Feb. 1993.
[14] M. Andrews, B. Awerbuch, A. Fernández, T. Leighton, Z. Liu, and J. Kleinberg, "Universal-stability results and performance bounds for greedy contention-resolution protocols," *J. ACM*, vol. 48, no. 1, pp. 39–69, Jan. 2001.
[15] B. A. Small, A. Shacham, and K. Bergman, "Ultra-low latency optical packet switching node," *IEEE Photon. Technol. Lett.*, vol. 17, no. 7, pp. 1564–1566, Jul. 2005.
[16] A. Shacham, B. A. Small, O. Liboiron-Ladouceur, and K. Bergman, "A fully implemented 12 ×12 data vortex optical packet switching interconnection network," *J. Lightw. Technol.*, vol. 23, no. 10, pp. 3066–3075, Oct. 2005.
[17] A. Shacham, B. G. Lee, and K. Bergman, "A wideband, non-blocking, 2 × 2 switching node for a SPINet network," *IEEE Photon. Technol. Lett.*, vol. 17, no. 12, pp. 2742–2744, Dec. 2005.
[18] A. Shacham, B. A. Small, and K. Bergman, "A wideband photonic packet injection control module for optical packet switching routers," *IEEE Photon. Technol. Lett.*, vol. 17, no. 12, pp. 2778–2780, Dec. 2005.
[19] A. K. Erlang, "Solution of some problems in the theory of probabilities of significance in automatic telephone exchanges," *Post Off. Electr. Eng. J.*, vol. 10, pp. 189–197, 1917.
[20] D. G. Kendall, "Stochastic processes occurring in the theory of queues and their analysis by the method of the imbedded Markov chain," *Ann. Math. Stat.*, vol. 24, no. 23, pp. 338–354, Sep. 1953.
[21] J. D. C. Little, "A proof of the queueing formula L = λW," *Oper. Res.*, vol. 9, no. 3, pp. 383–387, May/Jun. 1961.
[22] T. Lin, K. A. Williams, R. V. Penty, I. H. White, M. Glick, and D. McAuley, "Performance and scalability of a single-stage SOA switch for 10 × 10 Gb/s wavelength striped packet routing," *IEEE Photon. Technol. Lett.*, vol. 18, no. 5, pp. 691–693, Mar. 2006.
[23] B. A. Small, T. Kato, and K. Bergman, "Dynamic power considerations in a complete 12 × 12 optical packet switching fabric," *IEEE Photon. Technol. Lett.*, vol. 17, no. 11, pp. 2472–2474, Nov. 2005.
[24] O. Liboiron-Ladouceur, B. A. Small, and K. Bergman, "Physical layer scalability of WDM optical packet interconnection networks," *J. Lightw. Technol.*, vol. 24, no. 1, pp. 262–270, Jan. 2006.

Benjamin A. Small (S'98–M'06) received the B.S. (with honors) and M.S. degrees in electrical and computer engineering from Georgia Institute of Technology, Atlanta, in 2001 and 2002, respectively, and the M.Phil. and Ph.D. (with distinction) degrees in electrical engineering from Columbia University, New York, NY, in 2005.

He is currently a Postdoctoral Research Scientist with the Department of Electrical Engineering, Columbia University. His interests include optoelectronic device physics and modeling, as well as optical packet switching interconnection network traffic analysis and system-level behavior.

Assaf Shacham (S'03) received the B.Sc. (*cum laude*) degree in computer engineering from the Technion, Israel Institute of Technology, Haifa, Israel, in 2002 and the M.S. degree in electrical engineering from Columbia University, New York, NY, in 2004. He is currently working toward the Ph.D. degree in electrical engineering at the Department of Electrical Engineering, Columbia University.

From 1999 to 2001, he was a Circuit Designer with the Mobile Products Group, Intel Inc., Haifa. He then joined Charlotte's Web Networks in 2001 and spent two years working as a Logic Design Engineer in the core router hardware group. His research at Columbia University is focused on architectures of photonic chip-to-chip and intrachip interconnection networks in high-performance computing systems.

Keren Bergman (S'87–M'93) received the B.S. degree from Bucknell University, Lewisburg, PA, in 1988 and the M.S. and Ph.D. degrees from Massachusetts Institute of Technology, Cambridge, in 1991 and 1994, respectively, all in electrical engineering.

From 1994 to 2000, she was an Assistant Professor with the Department of Electrical Engineering, Princeton University, Princeton, NJ. She then joined the Optical Networking Group, Tellium, where she headed the optical design of large-scale MEMS-based cross-connects. Since 2001, she has been with the Department of Electrical Engineering, Columbia University, New York, NY, where she is a Professor of electrical engineering and directs the Lightwave Research Laboratory. She also leads multiple research projects in optical packet-switched networks, distributed grid computing over optical networks, photonic interconnection networks, nanophotonic networks-on-chip, and applications of optical networking in high-performance computing systems.

Dr. Bergman is a Fellow of the Optical Society of America (OSA). She is currently an Associate Editor of the IEEE PHOTONICS TECHNOLOGY LETTERS and the OSA *Journal of Optical Networking*.