The "Packing" and the "Scheduling" Packet Switch Architectures for Almost-All Optical Lossless Networks[1]

Emmanouel A. Varvarigos

Department of Electrical and Computer Engineering
University of California
Santa Barbara, CA 93106-9560

Abstract— We propose two almost-all optical packet switch architectures, called the Packing Switch and the Scheduling Switch, which when combined with appropriate wait-for-reservation or tell-and-go connection and flow control protocols provide lossless communication for traffic that satisfies certain smoothness properties. Both switch architectures preserve the order of packets that use a given input-output pair, and are consistent with virtual circuit switching. We find a lower bound on the number of elementary (2-state) switches required by any switch architecture to meet the objectives we have set. The number of 2-state switches used in the Scheduling Switch is of the optimal order jointly with respect to the number of inputs and with respect to the burstiness of the traffic streams (as measured by an appropriate parameter). The Packing Switch requires very little processing of the packet header, and uses a number of 2-state switches that is of the optimal order in terms of the burstiness parameter. We also examine the suitability of the proposed switches for the design of circuit switched networks. We find that the Scheduling Switch combines low hardware cost with little processing requirements at the nodes, and is an attractive architecture for both packet-switched and circuit-switched high-speed networks.

I. Objectives of the design

Traffic in high-speed networks can be switched either optically, or electronically. Even though optical switching has advantages for circuit switching, it is considered difficult to combine with packet switching. This is because efficient packet switching requires substantial packet storage, which is difficult to implement with current optical technology.

Networks using optical switching offer the potential of larger transmission speeds than networks using electronic switching by eliminating the need for optical to electronic (O/E) and electronic to optical (E/O) conversion of the data signal at intermediate switches, the so-called *electronic bottleneck*. For packet switching, however, O/E conversion is still required in order to process the packet header (see [HaG91], [Haa92], [ChF93], [BFS94], [CrT96]); switches in which the data remains in the optical domain while the packet header is processed electronically will be referred to as *almost-all optical switches*. In this paper we describe two switch architectures, called the Packing Switch and the Scheduling Switch architectures, to efficiently perform packet switching in almost-all optical networks. We also examine the suitability of these architectures for circuit-switched networks.

The objectives that we set for the Packing Switch and the Scheduling Switch architectures when used as packet switches are: 1) lossless communication, 2) efficient utilization of the capacity, 3) suitability for (almost) all-optical implementation, 4) consistency with virtual circuit switching, 5) simple or no resequencing requirements at the destination, and 6) modularity of the design. To meet these objectives both switch architectures have to be combined with appropriate connection and flow control protocols, which we also discuss.

The Packing Switch and the Scheduling Switch can provide lossless communication for sessions that have certain smoothness properties, or sessions that can tolerate the delay induced when transforming them into smooth sessions through the use of input flow control. When the sessions are bursty, additional delay lines, the number of which depends on the degree of burstiness, are required to provide lossless communication. We are interested in switches whose cost, measured by the number of elementary 2-state switches they require, increases in an optimal way as a function of the burstiness allowed. The Packing and the Scheduling switch architectures are modular, so that they can be easily expanded to accommodate more burstiness in the traffic, should this become desirable, in the same way that adding buffer space at an electronic switch can be done relatively easily. Both switch architectures preserve the order of packets that use a given input-output pair; this is important for multigigabit networks, where packet resequencing at the destination may be very difficult if order is not maintained within the network.

The Packing Switch requires $k^2 \log T + k \log k$ elementary 2-state switches to build, where k is the number of input ports, and T is some parameter

that determines the burstiness allowed for the sources and the flexibility we have in assigning rates to sessions. It uses a simple scheme to assign output slots to the incoming packets so as to minimize the processing requirements, while preventing internal packet collisions within the switch. The Scheduling Switch consists of $2k \log T + k^2$ two-state elementary switches (or $2k \log T + 2k \log k$ elementary switches, if a different version is used). We show that $\Omega(k \log(k+T))$ is a lower bound on the number of switching elements required

[1]Research supported by DARPA under the MOST project.

0-7803-4198-8/97/$10.00 © 1997 IEEE

by any switch architecture that meets the objectives we have set. Therefore, the number of 2-state switches used by the Scheduling Switch is of the optimal order, both with respect to the number of inputs and with respect top the burstiness of the traffic streams that are switched.

We also consider implementations of the Packing and the Scheduling Switch architectures as *circuit switches*. When circuit switching is employed, the frequency with which the switch state has to be reconfigured is of the order of the arrival rate of new connections rather than of the order of the packet arrival rate. When used as circuit switches, both the Packing and the Scheduling Switch are nonblocking, so that a new circuit connection can always be routed through the switch as long as there is adequate available capacity on the desired incoming and outgoing links. We distinguish between two types for the reconfigurations that have to take place at a switch to admit a new circuit connection: reconfigurations of the *local* type, where accommodating a new connection at the switch involves changes only in the state of that switch, and reconfigurations of the *nonlocal* type, where accepting a new connection at a switch requires changing the state of other switches. We show that the admittance of a new connection at a Scheduling Switch involves in most cases only a local reconfiguration of the switch, with nonlocal reconfiguration required rather infrequently. Also, the Scheduling Switch requires little processing of the setup packet, making the design particularly suitable for circuit switching. Admitting a new connection at a Packing Switch, however, often requires nonlocal reconfigurations, making the Packing Switch unattractive for circuit switching.

The organization of the remainder of the paper is as follows. In Section 2 we describe our objectives for the switch architectures and our assumptions on the traffic. The Packing and the Scheduling Switch architectures are described in Sections 3 and 4, respectively. In Section 5 we find a lower bound on the number of 2-state elementary switches required by any switch architecture that meets the objectives we have set, and compare it to the complexity of the switch architectures that we propose. In Section 6 we comment on the processing requirements of the switch designs when used as packet switches. Finally, in Section 7 we consider the suitability of the Packing and the Scheduling Switch for building circuit-switched networks.

II. Assumptions on the traffic

We assume that all packets have the same length and require one slot for transmission. Following the discussion in [Gol91], we view the time axis on a link as being divided into frames of duration equal to T packet slots. A session is said to have the (n, T)-*smoothness property* at a node if at most n packets ($n \in \{1, \ldots, T\}$) of the session arrive at that node during a frame. By using a leaky bucket scheme [ELL90] to shape traffic at the source, and the stop-and-go queueing discipline [Gol91] to forward traffic at intermediate nodes, a session can be made to have the (n, T) - smoothness property throughout the network. The idea behind stop and go queueing is to transmit all packets arriving over the same incoming frame of a link and requesting the same outgoing link during the same outgoing frame, preserving in this way frame integrity and the (n, T) - smoothness property at subsequent nodes. The parameter T can be viewed as a measure of the burstiness we allow: the larger T is, the more bursty the session is allowed to be. A session S having the (n_S, T)-smoothness property has average rate at most equal to $R_S = n_S C/T$. Since link capacity can be allocated to a session only at discrete levels that are multiples of C/T, where C is the link capacity, T can also be viewed as a measure of the flexibility we have in assigning rates to sessions. We let n_{ij} be the number of packets that arrive over an incoming link i and have to be transmitted on the same outgoing frame of link j. Assuming unicast communication, we always have

$$\sum_{j=1}^{k} n_{ij} \leq T, \quad \text{for all } i \in \{1, 2, \ldots, k\}, \tag{1}$$

where k is the number of incoming (or outgoing) links. We assume that the connection and flow control protocols used guarantee that

$$\sum_{i=1}^{k} n_{ij} \leq T, \quad \text{for all } j \in \{1, 2, \ldots, k\}. \tag{2}$$

In other words, we assume that the protocols ensure that the number of packets requesting the same outgoing frame is always less than or equal to the duration of the frame. If Eq. (2) holds, a different outgoing slot can be assigned to each incoming packet, so that no packets will have to be dropped, provided that the switch is able of delaying the packets until their assigned slots arrive (and assuming no transmission errors). Since the total average rate of sessions using incoming link i and outgoing link j is $R_{ij} = n_{ij} C/T$, Eq. (2) can be restated as

$$\sum_{i=1}^{k} R_{ij} \leq C, \quad \text{for all } j, \tag{3}$$

which simply requires that the sum of the average rates of the sessions using a link should be less than the link capacity all network links. The condition of Eq. (2) [or the equivalent condition of Eq. (3)], which ensures the lossless character of the design, can be enforced for all network links by using either a *wait-for-reservation* protocol (see, e.g., [CGS93], [VaS95]) or a *tell-and-go* type of protocol (such as the Virtual Circuit Deflection protocol [VaL96], where virtual circuits, as opposed to packets, may be deflected).

III. The Packing Switch

The frames on the incoming and the outgoing links of a node will not, in general, be synchronized. We let k be the number of incoming (or outgoing) links of a node, and let $\delta_{i,j}$ be the phase difference between the beginning of the

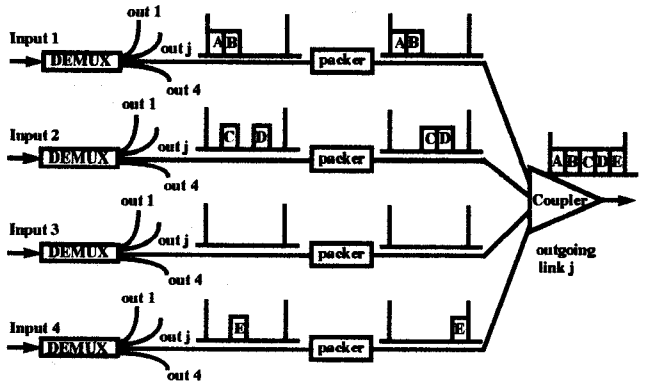

Fig. 1. Illustrates the Packing Switch. Only the details corresponding to output link j are shown.

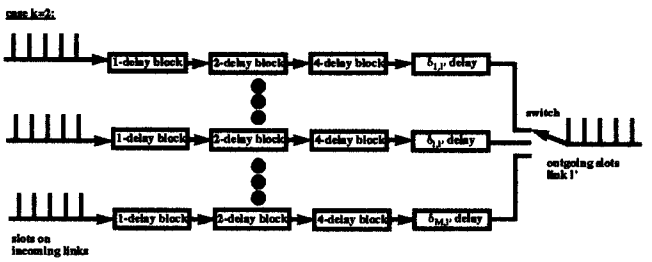

Fig. 2. We illustrate the output system for a particular outgoing link j. Each delay block can be implemented by a switch and an optical fiber of appropriate length, as shown in Fig. 3. Depending on the distance between the arriving and the departing slot of a packet, the state of each delay block is set so that a packet is delayed until its assigned outgoing slot comes. The $\delta_{i,j}$ delays are implemented using fibers of appropriate length, and account for the misalignment between incoming and outgoing frames.

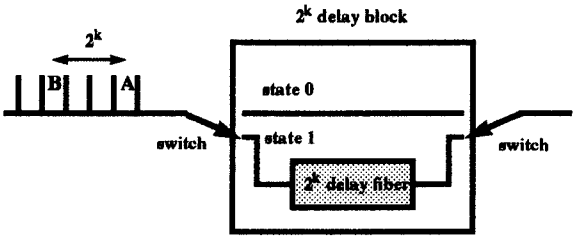

Fig. 3. We illustrate the design of a 2^l-delay block. A packet collision may occur in the case where packet B lags packet A by 2^l slots, and packet A passes through the upper branch (state 0), while packet B passes through the lower branch (state 1) of the 2^k-delay block. We prove in Theorem 1 that if the packing rule is used to assign packets to outgoing slots, two packets will never appear at the output of a stage during the same slot.

frames on links i and j. To preserve frame integrity, we request that packets arriving in frame $F(i)$ of incoming link i and destined for outgoing link j, are transmitted in the first frame $F(j)$ of link j that starts after the end of $F(i)$ (or, more generally, they are transmitted in the p-th subsequent frame, where p is a constant); see also Fig. 4.

Figure 1 shows a block diagram of the Packing Switch architecture. The output of the demultiplexer at which a packet is forwarded is determined based on its virtual path identifier (VPI) as described in Section 7. The *Packer* in Fig. 1 is a restricted type of time slot interchanger that rearranges the packets requesting the same output j so that they appear at different time slots, in a way to be described shortly. The multiplexer can then be implemented as a passive coupler that combines the streams of packets arriving over different inputs, and transmits them over link j.

As mentioned in Section 2, we assume that the flow control protocol guarantees that the number n_{ij}, $i = 1, 2, \ldots, k$, $n_{ij} \in \{0, 1, \ldots, T\}$, of packets that arrive during frame $F(i)$ and are transmitted during frame $F(j)$ of link j satisfy Eq. (2) (or, equivalently, Eq. (3)), so that there are always enough slots in outgoing frame $F(j)$ to serve all packets that have to be transmitted in it. For this to happen, however, it is necessary to delay a packet arriving in incoming frame $F(i)$ until the time of its transmission on outgoing frame $F(j)$ comes. The required delay can take any value between 1 and $2T-1$ slots, and it can be implemented using $2T-1$ optical delay lines of variable lengths between 1 and $2T-1$ slots, for a total fiber length per incoming link equivalent to $T(2T-1)$ slots. For a design using delay lines to be practical, the number of delay elements has to be reduced. In what follows, we describe a construction that uses only $\log T$ delay elements per link, with a total fiber length proportional to T.

The delay lines that implement the buffering system for a particular outgoing link j are depicted in Fig. 2 for the case $2T = 2^3$. A 2^l-delay block at stage l can be in state 0 or 1. If the block is in state 0, it does not introduce any delay, while if it is in state 1 it introduces delay equal to 2^l slots. In our protocols, a packet may have to be delayed by anywhere between 1 and $2T-1 = 2^m - 1$ slots. Clearly, all delays in this range can be implemented by appropriately choosing the states of the delay blocks. Since different packets have to be delayed by different amounts, the state of a block will in general change at the end of a slot. However, as long as the arrival pattern on the incoming links remains the same (for example, if the packets of each session arrive periodically in the incoming frames and as long as no new sessions are added), the sequence of states used will be the same for successive frames. For the design given in Fig. 2 to work, it is necessary that two different packets never appear during the same slot at the output of a stage. To prevent collisions (see Fig. 3 for an example of such a collision), the assignment of incoming slots (packets) to outgoing slots cannot be arbitrary. In what follows, we present an assignment method, called the *packing rule*, which guarantees that no collisions arise in the system of Fig. 2. We focus on a particular frame $F(j)$ of an outgoing link j. Consider a packet A that arrives in slot $x_A \in \{0, 1, \ldots, T-1\}$ of frame $F(i)$, and assume that it is the r_A^{th} packet destined for outgoing link j to arrive in $F(i)$ (the integer r_A, $r_A \in \{1, \ldots, n_i\}$, will be referred to as the *rank* of packet A). Then, according to the packing rule, packet A is assigned to slot $y_A = \sum_{l=1}^{i-1} n_l + r_A - 1$, $y_A \in \{0, 1, \ldots, T-1\}$, of the outgoing frame $F(j)$ (see Fig. 4). As stated in the following theorem, when packets are

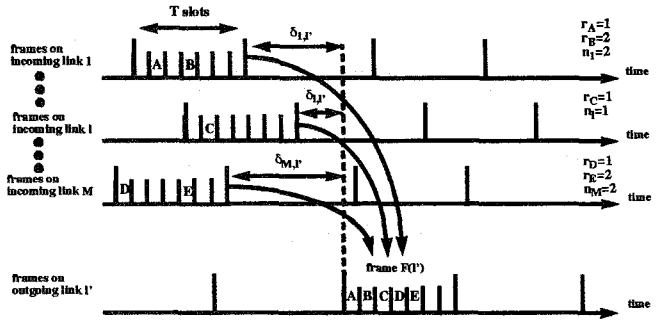

Fig. 4. We illustrate the incoming and outgoing frames at a node. Packets not intended for outgoing link j are not shown. Packets intended for link j are assigned to outgoing slots according to the packing rule described in the text.

assigned to outgoing slots according to the packing rule, no collisions occur at the outputs of the delay blocks. We omit the proof.

Theorem 1: When the packing rule is followed, two packets will never appear at the output of a stage during the same slot.

The loss introduced by the passive coupler at the right end of a delay block in Fig. 3 can be avoided by replacing the passive coupler by a 2-state switch. In this case, the states of the switches at the left and the right side of a delay block will have to be jointly set.

An important advantage of the packing rule is that it requires very little processing at the switch. Indeed, computing the rank of a packet is very simple and can be done in hardware. The packing rule ensures lossless communication when the condition of Eq. (2) [or the equivalent condition of Eq. (3)] is satisfied.

IV. The Scheduling Switch

In this section we describe the Scheduling Switch architecture, a block diagram of which is shown in Fig. 5. The purpose of the Scheduler is to rearrange the incoming packets so that packets appearing during the same slot at the outputs of the Scheduler require different outgoing links of the crossbar switch. If this property is satisfied by the Scheduler, then the crossbar switch will be able to route each packet to its desired outgoing link without any collisions. An important data structure that will be useful in describing the operation of the Scheduler is that of the *frame matrix*, defined as the $k \times k$ matrix $N = \{n_{ij}\}$ whose $(i,j)^{\text{th}}$ is equal to the number n_{ij}, $i = 1, 2, \ldots, k$, $j = 1, 2, \ldots, k$, of packets that arrive during a given frame $F(i)$ of incoming link i and require the same frame $F(j)$ of outgoing link j.

Definition 1: The *critical sum* h of a matrix is equal to $\max_{i,j}(r_i, c_j)$, where r_i is the sum of the entries of row i, c_j is the sum of the entries of column j, and the maximization is performed over all rows i and columns j. A row or column with sum of entries equal to h is called a *critical line*.

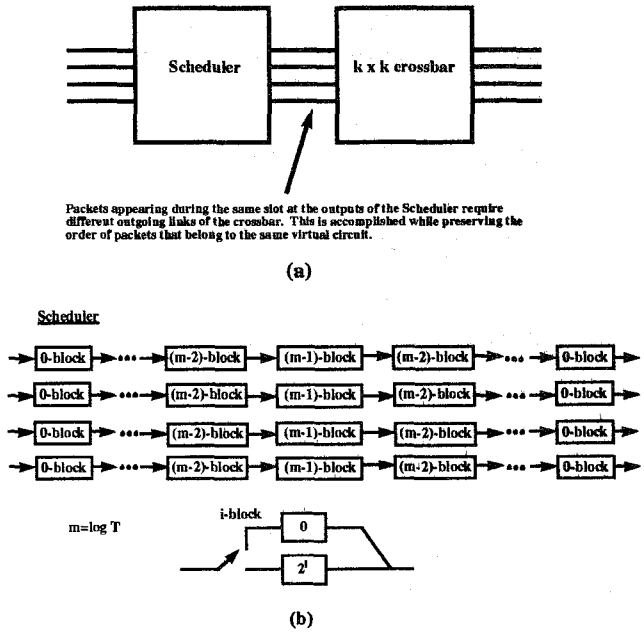

Fig. 5. (a) The Scheduling Switch architecture. (b) The Scheduler is implemented by k parallel branches, one for each input. A branch in turn consists of $2\log T - 1$ delay blocks, for a total of $2k\log T - k$ 2-state switches and couplers.

From Eqs. (2)-(3) we have

$$h \leq T \tag{4}$$

for the critical sum h of a frame matrix. This is because the number of packets r_i arriving over link i during a frame is always less than or equal to T, and the number of packets c_j that request a given frame of output link j is guaranteed by the connection and flow control protocol to be at most equal to T.

For any matrix, we use the term *line* to refer to a row or column of the matrix.

Definition 8: A *perfect* matrix is a square matrix with nonnegative integer entries and with the property that the sum of the entries of each line is the same for all lines.

Definition 9: A *permutation matrix* is any matrix with entries equal to 0 or 1 with the property that each line of the matrix has at most one nonzero entry.

We now give a well known result which is due to Hall (see [Rys65], p. 57).

Theorem 3 (Hall's Theorem): A perfect matrix can be written as a sum of h permutation matrices, where h is the sum of the entries of its lines.

The following lemma gives the complexity for decomposing a perfect matrix into a sum of permutation matrices; it can be proved by viewing the decomposition problem as a sequence of bipartite matching problems.

Lemma 1: The decomposition of a perfect matrix M that has critical sum h as the sum of h permutation matrices can be found in $O(k^{5/2}h)$ time.

The following theorem is found in [BCW81].

Theorem 4: Given any nonnegative integer square matrix N with critical sum h, there exists a nonnegative integer matrix E such that $N + E$ is a perfect matrix with critical sum h.

Theorems 3 and 4 combined with Eq. (4) yield the following lemma.

Lemma 2: A frame matrix N can be written as the sum

$$N = \sum_{s=1}^{T} P_s$$

of at most T permutation matrices.

Figure 4 shows an example of the decomposition of the frame matrix N as a sum of permutation matrices.

Matrix P_s is used to determine the packet (if any) that will appear during slot s at each of the outputs of the Scheduler. In particular, if the (i, j)-th entry of matrix P_s is equal to one, then a packet arriving over link i and departing over link j is assigned to the s^{th} outgoing slot of the Scheduler. Since P_s is a permutation matrix, this assignment guarantees that no packets arriving over the same incoming link or requesting over the same outgoing link of the switch appear during the same outgoing slot s of the scheduler. The first property ensures that there will be no collisions at the output of the Scheduler when the Scheduler is implemented as a set of parallel delay lines, as indicated in Fig. 5b, while the second property ensures that there will be no collisions at the outputs of the crossbar switch. Since there are n_{ij} packets arriving over link i and requesting link j, we have freedom in choosing the order in which these packets will be assigned to the outgoing slots of the Scheduler. The assignment can therefore be chosen so as to preserve the order of packets arriving over the same input and requesting the same output, meeting in this way one of the main requirements of virtual circuit switching.

The Scheduler consists of k parallel branches, each of which has the purpose of delaying the packets arriving over a particular incoming link until their assigned outgoing slot arrives. Since different packets arriving over a link are assigned to different outgoing slots of the Scheduler, the functionality of each branch is identical to that of a Time Slot Interchanger. Therefore, each branch can be implemented by $2 \log T - 1$ elementary switches, as shown in Fig. 5b and described in detail in [LiG93].

V. Can we do better?

The Packing and the Scheduling Switch architectures provide lossless communication for (n, T)-smooth traffic, while preserving the order of packets that arrive over the same input and request the same output of the switch. The total number of 2-state switching elements required by the Packing Switch and the Scheduling Switch to meet these objectives is equal to $k^2 \log T + k \log k$ and $2k \log T + k^2$, respectively, where k is the number of inputs or outputs. The

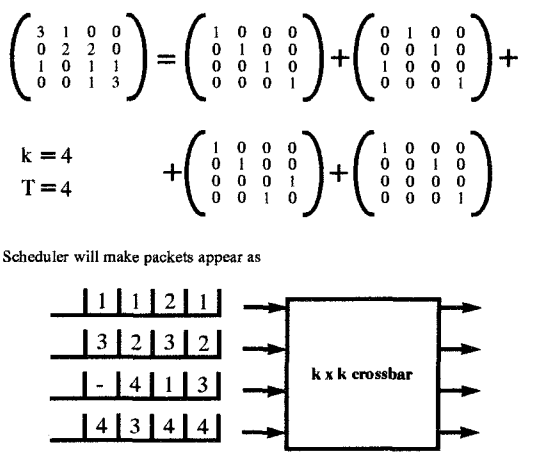

Fig. 6. A decomposition of a frame matrix as a sum of permutation matrices, and the corresponding assignment of packets to outgoing slots of the Scheduler.

$k \log k$ and k^2 terms in the previous expressions correspond to the number of 2-state switches required to build the k multiplexers (each implemented as a tree) in the Packing Switch (Fig. 1), or the $k \times k$ crossbar switch that follows the Scheduler in the Scheduling Switch (Fig. 5a), respectively. In what follows we find lower bounds on the minimum number of 2-state switching elements required by any switch architecture that meets our objectives, and compare it with the number of elementary switches required by our designs.

We let S be the number of 2-state switching elements in a switch. Clearly, different instances of the frame matrix N require different configurations of the S switching elements during a frame, if N is a perfect matrix and has critical sum equal to T. The number of different such frame matrices is equal to $\binom{k+T}{k}^k$, as can be found by noting that the row sum $r_i = T$ of row i, $i = 1, 2, \ldots, k$, can be split into numbers $n_{i1}, n_{i2}, \ldots, n_{ik}$ in $\binom{k+T}{k}$ distinct ways, and there is a total of k rows in the frame matrix. Furthermore, each of the $T!$ permutations of the T packets arriving during a frame of input i (equivalently, each permutation of the T elements of row i) has to be switched using a different switch configuration. This is because different permutations of the packets arriving over the same input and requesting different outputs require different switch configurations (otherwise packets would end up at the wrong output), and different permutations of the packets arriving over the same input and requesting the same output also require different switch configurations (because the switch has to preserve the order of such packets). Therefore, we have $(T!)^k$ different switch configurations corresponding to a given perfect matrix, and $\binom{K+T}{k}^k$ different choices for a perfect matrix of critical sum T, for a total of $\binom{k+T}{k}^k (T!)^k$ different switch configurations during a frame. Since the number of different states in which S 2-state switching elements can be set during T slots is 2^{ST}, we have

$$2^{ST} \geq \binom{K+T}{k}^k (T!)^k = \left(\frac{(k+T)!}{k!}\right)^k,$$

or, equivalently,

$$S \geq \frac{k}{T} \log \left[\frac{(k+T)!}{k!} \right]$$

Using Stirling's approximation (see [Gal68], p. 530) we get the following lower bound

$$S \geq \frac{k^2}{T} \log(k+T) - \frac{k^2}{T} \log(k) + k \log(k+T) > k \log(k+T)$$

on the number of elementary (2-state) switches required by a $k \times k$ node to provide lossless communication for (n, T)-smooth traffic, while preserving the order of packets that use a given input-output pair. Comparing the lower bound with the number of 2-state switching elements required by the Scheduling Switch we see that they are of the same order of magnitude with respect to the parameter T that determines the burstiness of the sessions (or the flexibility of assigning rates to sessions). Note that the crossbar switch of cost k^2 that follows the Scheduler could be replaced by a rearrangeably nonblocking multistage switch of cost $O(k \log k)$ elementary switches. In that case, the dependence of the number of elementary switches on the number of inputs k of the Scheduling Switch would also be of the optimal order; the additional complexity, however, required for the control of the switch will probably more than offset the decrease in hardware complexity, when k is small.

VI. Reading the Packet Header

For packet switching, the virtual path identifier (VPI) of a packet is required to determine the desired outgoing link of the packet. The VPI can be obtained by using a splitter at each input of the switch to direct a small fraction of the received energy to a photodetector. The splitting ratio should be chosen so that the energy that arrives at the photodetector is sufficient to decode the header. This scheme can also be combined with the *field coding* technique [HaG91], where a smaller rate is used to transmit the packet header, allowing the electronic part of the switch to operate at a smaller rate than the data transmission rate. The VPI's of the packets arriving during a frame are processed as described in Sections 3 and 4 by the control unit of the switch to determine the state at which each of the 2-state switching elements is set during a slot. The splitters have to be followed by delay lines of sufficient length to allow for the latency incurred by the electronic processing of the VPI of a packet (Figure 4).

In order to avoid the need for modifying the header of a packet at each node (which would require an E/O conversion for the header in addition to the O/E conversion mentioned above, considerably complicating the switch design), we request that a session uses the same VPI for its entire path. This is easy to do if we assign to each source a set of VPI's for its exclusive use. The owner of an unused VPI can lend it to another node, and may request it back when it wants to use it. Other ways of distributing the available VPI's are also possible, and the only requirement is that a VPI should not be used by more than one sessions at any given time (for example, we could assign a distinct set of VPI's to each destination; this solution, however, looks inferior to the previous one, because it considerably complicates the redistribution of unused VPI's).

VII. Using the Packing and the Scheduling Switch as Circuit Switches

In the previous sections we have introduced the Packing and the Scheduling Switch as almost-all optical switch architectures for building packet-switched multigigabit-per-second networks. Packet switching has a number of well-known advantages, but it imposes severe processing requirements at the nodes. In our designs, for example, the sequence of states at which the elementary switches are set during each slot of a frame has to be calculated for each frame. For the Packing Switch this involves computing the rank of the incoming packets (which requires 1 or 2 additions per packet, and can be performed using counters), and then finding the binary representation of the delay between the incoming and the assigned outgoing slot of a packet. For the Scheduling Switch, $O(k^{5/2}T)$ operations are required to decompose the frame matrix as a sum of permutation matrices, and an additional $O(kT \log T)$ operations are needed for setting the switches (the latter is equivalent to only $O(1)$ operations per switching element, and can be done in parallel for each of the k inputs). Even though the above processing requirements are close to the minimum possible for packet switching (because at least $\Omega(1)$ operations are needed to set a switching element during a slot, and there are $O(k \log T)$ of them), they may still become a bottleneck of the design if the switch processor(s) is not fast enough.

In this section we examine the suitability of the Packing and the Scheduling Switch for building high-speed networks that use *circuit switching*. We assume again that the time axis is divided into frames of T slots each, but now a session (circuit) of rate nC/T is allocated n particular slots in each frame for its exclusive use throughout the duration of the session. With circuit switching reading the packet headers at a switch is no longer necessary, since packet arrivals are periodic, with packets of a given session always arriving over the same incoming slot(s) and leaving over the same outgoing slot(s) of a frame. As a result, the state of the switch has to be reconfigured only when the setup packet of a new session arrives, and the time scale at which computations have to be performed is of the order of the session holding times, rather than of the order of the packet transmission times. Circuit switching, however, does not handle bursty traffic efficiently, and it requires additional overhead for tearing down a circuit when completed. Also, a separate control channel is required to setup connections.

An important issue in evaluating the suitability of a switch architecture for circuit switching is related to whether or not the acceptance of a new circuit connection at a switch requires the reconfiguration of the state of that switch only, or it requires the reconfiguration of the state of other

switches as well. Clearly, the Packing Switch architecture is not well suited for circuit-switched networks, because it requires the frequent reconfiguration of other nodes in order to admit new connections at a node. To see that, consider the case where a new setup packet arrives at a node requesting an outgoing link that has adequate available capacity. In order to serve the new connection, it may be necessary to change the outgoing slots used by the existing connections (because the rank of the existing connections, defined in Section 3, may change). This in turn requires reconfiguring the state of the downstream switches used by these connections, resulting in a possibly large number of changes that have to be performed to accept the new connection at a particular switch (not to mention serving the new connection at subsequent switches).

In what follows we show that accepting new connections at a Scheduling Switch requires in most cases only local changes in the state of the switch. To see that, let N be the frame matrix prior to the arrival of a new connection (or prior to the departure of an existing one), and let h ($h \leq T$) be its critical sum. As shown in Theorem 4, we can find a (nonnegative) matrix E, to be referred as the *slack matrix*, such that $N + E$ is equal to a perfect matrix M of critical sum T. Furthermore, M can be written as a sum of T permutation matrices, yielding

$$M = N + E = \sum_{s=1}^{T} P_s. \tag{5}$$

The permutation matrices P_s, $s = 1, 2, \ldots, T$, can be used (in the way described in Section 4) to set the state of the elementary 2-state switches so as to serve all the connections recorded in frame matrix N. Therefore, matrix M, which will be referred to as the *target perfect matrix*, determines through its decomposition into permutation matrices the current state of the switch. Note also that in addition to the existing connections in N, more connections (those corresponding to the slack matrix E) could also be served without requiring any reconfiguration of the switch.

When an ongoing connection S using input i and output j is terminated, no reconfiguration of the switch is required, and the only computation that has to take place is to update the (i, j)-th entries of matrices N and E according to $n_{ij} := n_{ij} - 1$ and $e_{ij} := e_{ij} - 1$, respectively.

We now consider the case where a new connection S from input i to output j is requested, and there is enough available capacity on outgoing link j to accommodate it. For the switch to be nonblocking, such a request should be served. Since there exist a slot on input i and a slot on output j that were not previously occupied by a connection, the new frame matrix

$$N^{\text{new}} = N + 1_{ij}$$

will also have critical sum less than or equal to T, where 1_{ij} denotes the $k \times k$ matrix that has all entries equal to zero, except for thenew(i, j)-th entry, which is equal to one. If $e_{ij} > 0$, it is clear that the new connection can be served without reconfiguring the switch. This is because we will then have $N^{\text{new}} + E^{\text{new}} = M$, where $E^{\text{new}} := E - 1_{ij} \geq 0$, and the same target perfect matrix M and decomposition $M = \sum_{s=1}^{T} P_s$ can be used to determine the outgoing slot at which the new connection is assigned. In other words, if the entries of the slack matrix E are viewed as corresponding to dummy connections, the new connection can just replace one of the dummy connections, without changing the switch configuration. If $e_{ij} = 0$, however, establishing a new connection from input i to output j requires changing the state of the switch (equivalently, changing the target matrix M and its decomposition into permutation matrices). Since a new connection is requested from input i to output j, the i-throw sum and the j-th column sum of N are both strictly less than T, and there exist $p, q \in \{1, 2, \ldots, k\}$, such that $e_{pj} > 0$ and $e_{iq} > 0$. By defining the new target matrix

$$M^{\text{new}} = M + 1_{ij} - 1_{iq} - 1_{pj} + 1_{pq}, \tag{6}$$

and the new slack matrix

$$E^{\text{new}} = E - 1_{iq} - 1_{pj} + 1_{pq},$$

we have

$$M^{\text{new}} = E^{\text{new}} + N^{\text{new}},$$

with $E^{\text{new}} \geq 0$. The number of operations required to update M^{new} and E^{new} when a new connection is accepted is $O(k)$.

We next show how to decompose M^{new} into permutation matrices efficiently, while minimizing the number of existing connections that have to be reassigned to different outgoing slots. Since, $e_{iq} > 0$ and $e_{pj} > 0$, there exist permutation matrices P_m and P_n (not necessarily different) in the decomposition $N = \sum_{s=1}^{T} P_s$ whose (i, q) and (p, j) entries are nonzero and correspond to dummy connections. If such entries can be found on the same matrix P_m, then replacing matrix P_m with matrix

$$\tilde{P}_m = P_m + 1_{ij} - 1_{iq} - 1_{pj} + 1_{pq}$$

yields a decomposition of M^{new} into the sum of T permutation matrices, which define the new switch configuration. Note that in this case, existing connections are assigned to the same outgoing slot they were using before, so that only a local reconfiguration is required at the switch. If such a matrix cannot be found, then there exist permutation matrices P_m and P_n, such that the (i, q)-th entry of P_m and the (p, j)-th entry of P_n are nonzero and correspond to dummy connections. Since the matrix $P_m + P_n + 1_{ij} - 1_{iq} - 1_{pj} + 1_{pq}$ has critical sum equal to 2, it can be decomposed in time $O(k^{5/2})$ as the sum of two permutation matrices, so that

$$P_m + P_m + 1_{ij} - 1_{iq} - 1_{pj} + 1_{pq} = \tilde{P}_m + \tilde{P}_n. \tag{7}$$

Equations (6)-(8) then give

$$M^{\text{new}} = \tilde{P}_m + \tilde{P}_n + \sum_{s \neq m,n} P_s.$$

This decomposition provides an assignment of connections to output slots that serves both the new and the existing

connections. Note that the only existing connections that may have to be reassigned to new outgoing slots are those previously assigned to outgoing slots m and n. It can be seen that at most $k-1$ (out of a total of up to kT) of the existing connections may have to be reassigned to a new slot, in the worst case. The number of arithmetic operations required for computing the new assignments is $O(k^{5/2})$ in the worst case, and these computations have to be performed only when a new setup packet is accepted. When the sum of the number of slots used on incoming link i and on outgoing link j is less than T, it can be shown that no existing connections will have to be reassigned. Finally, one can also opt to reject a new connection when its service would require the reassignment of existing connections to different outgoing slot (of course, in that case the functionality of the switch will not correspond to that of a nonblocking switch).

VIII. Conclusions

We have proposed two almost-all optical packet switch architectures, which when combined with appropriate flow and connection control protocols provide lossless communication and packet arrival in the correct order. We compared the cost of the switches (as measured by the number of elementary 2-state switches for a given number of inputs and a given burstiness of the traffic) to a lower bound on the cost required by any switch architecture that meets our objectives. The cost of the Scheduling Switch is of the optimal order both with respect to the number of inputs and with respect to the burstiness of the traffic streams, while the cost of the Packing Switch is of the optimal order with respect to the burstiness parameter but not with respect to the number of inputs. The Packing Switch architecture uses a very simple rule to assign incoming packets to outgoing slots, and is appropriate for building almost-all optical packet switches. The Scheduling Switch combines hardware simplicity with small processing requirement, and appears to be an attractive alternative for building both packet-switched and circuit-switched almost-all optical high-speed networks.

IX. Reference

[BCW81] Bongiovanni, G., Coppersmith, D., and Wong, C. W., "An Optimum Time Slot Assignment Algorithm for an SS/TDMA system with Variable Number of Transponders", IEEE Trans. Commun., Vol. COM-29, pp. 721-726, 1981.

[ChF93] Chlamtac, I., and Fumagalli, A., "An Optical Switch Architecture for Manhattan Street Networks," J. on Select. Areas in Commun., Vol. 11, pp. 550-559, 1993.

[CGS93] Cidon, I., Gopal, I. S., and Segall, A., "A Connection Establishment in High-Speed Networks," IEEE/ACM Trans. on Networking, Vol. 1, pp. 469-481, 1993.

[CrT96] Cruz, R. L., and Tsai, J.-T., "COD: Alternative Architectures for High Speed Packet Switching," IEEE/ACM Trans. on Networking, Vol. 4, No. 1, February 1996.

[ELL90] Eckeberg, A., Luan D., and Lucantoni, M., "An Approach to Controlling Congestion in ATM Networks," International Journal of Digital and Analog Communication Systems, Vol. 3, No. 2, pp. 199-209, 1990.

[Gal68] Gallager, R., *Information Theory*, Wiley, 1968.

[Gol91] Golestani, S. J., "Congestion-Free Communication in High-Speed Packet Networks," IEEE Trans. on Communications, Vol. 39, No. 12, December 1991.

[Haa92] Haas, Z., "The Staggering Switch: An Almost-All Optical Packet Switch," pp. 1593-1599.

[HaG91] Haas, Z., Gitlin, R. D., "Field Coding: A High-peed 'Almost-all' Optical Interconnect," Proc. of the Twenty Fifth Annual Conference on Info. Sciences and Systems, Baltimore, Maryland, March 20-22, 1991.

[HuS93a] Hunter, D. K., and Smith, D. G., "New Architectures for Optical TDM Switching," J. of Ligthwave Technology, Vol. 11, No. 3, 1993.

[HuS93b] Hunter, D. K., and Smith, D. G., "An Architecture for Frame Integrity Optical TDM Switching," J. of Ligthwave Technology, Vol. 11, No. 5/6, 1993.

[LiG95] Lin, P. J., ang Gallager, R. G., "Time Slot Interchange Using Fiber Delay Lines," preprint.

[PaS82] Papadimitriou, C. H., and Steiglitz, K., *Combinatorial Optimization, Algorithms and Complexity*, Prentice Hall, 1982.

[Rys65] Ryser, H. J., *Combinatorial Mathematics*, The Mathematical Association of America, Rahway, N.J., 1965.

[VaS95] Varvarigos, E. A., and Sharma, V., "An Efficient Reservation Virtual Circuit Protocol," under review (Version appeared in Proc. Int'l Symp. on Info. Theo., 1995.)

[VaL96] Varvarigos, E. A., and Lang, J., "A Novel Virtual Circuit Deflection Protocol for Multigigabit Networks and its Performance for the MS Topology," Proc. GLOBECOM'96, pp. 1544-1548.