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

Emmanouel (Manos) Varvarigos

Abstract— **This paper proposes two almost all-optical packet switch architectures, called the "packing switch" and the "scheduling switch" architecture, 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. The scheduling switch requires $2k \log T + k^2$ two-state elementary switches (or $2k \log T + 2k \log k$ elementary switches, if a different version is used) where k is the number of inputs and T is a parameter that measures the allowed burstiness of the traffic. The packing switch requires very little processing of the packet header, and uses $k^2 \log T + k \log k$ two-state switches. We also examine the suitability of the proposed architectures 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.**

Index Terms— **All-optical packet and circuit switches, flow control protocols, lossless communication.**

I. Objectives of the Design

THE rapid developments in optoelectronics technology have substantially increased system transmission rates in optical communication networks since the first systems were installed a decade ago. The first 8 Gb/s system and the first 16 Gb/s system were demonstrated in AT&T in the 1980s. In Japan, several companies (including NEC, Fujitsu, Hitachi, and Toshiba) have developed 9.8 Gb/s networks. A large effort also exists in Europe under the RACE support, where a number of companies have developed 10 Gb/s networks. In the United States, several gigabit network testbeds have or are currently being developed, including among others the AT&T Lucky Net, the Aurora gigabit tetbed, the PARIS network, the Zeus Project at Washington University at St. Louis, MO, the all-optical testbed at Lincoln Laboratories and MIT, the WEST project at Rockwell, and the 40 Gb/s Thunder and Lightning network at UCSB. Since the fiber bandwidth is practically infinite (20 THz), considerably higher bit rates are expected to be feasible in the near future.

Manuscript received April 29, 1998; revised March 1998. This work was supported by DARPA under the MOST Project.

The author is with the Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106 USA.

Publisher Item Identifier S 0733-8724(98)07397-6.

Traffic in high-speed networks can be switched either optically, or electronically. Even though optical switching has advantages for circuit switching, it is generally 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 (optical storage, using optical fiber loops with optical amplifiers and optical switches, is bulky and expensive compared to electronic storage).

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 [12], [5], and [7]); 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 describe.

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. For bursty sessions, additional delay lines, the number of which depends on the degree of burstiness, are required to provide lossless communication. 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

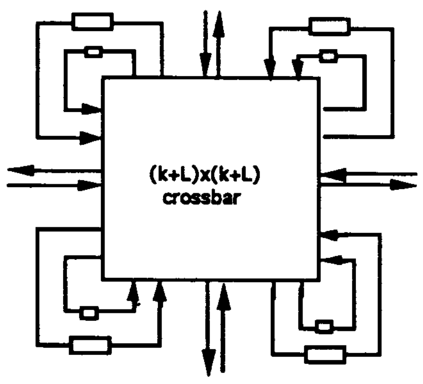


Fig. 1. A crossbar switch with k external inputs and outputs and L recirculating loops for storage implementation.

of packets that use a given input–output pair; this is important for multigigabit networks, where a session may involve the transfer of millions of packets, making packet resequencing at the destination a difficult task if order is not maintained within the network. Switch architectures that use recirculating loops for optical storage implementation (see Fig. 1) are not modular and tend not to preserve the order of packets. Also, switches that rely on (datagram) deflection routing to provide lossless communication do not guarantee packet arrival in the correct order.

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 withing 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 describe wait-for-reservation and tell-and-go type of flow and connection control protocols that can be combined with the proposed switch architectures to meet our objectives. 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 may 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. The admittance of 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 II, 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 III and IV, respectively. In Section V, we describe protocols that can be combined with the proposed switch architectures to meet our objectives. In Section VI, we comment on the processing requirements of the switch designs when used as packet switches. Finally, in Section VII we consider the suitability of the packing and the scheduling switch for building circuit-switched networks.

II. Traffic Model

We assume that all packets have the same length and require one slot for transmission. Following the discussion in [10], 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 $(n \in \{1, \cdots, T\})$ of the session arrive at that node during a frame. By using a leaky bucket scheme [8] to shape traffic at the source, and the stop-and-go queueing discipline [10] 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 $R_S = n_S C/T$, where C is the link capacity. Since capacity can be allocated to a session only at discrete levels that are multiples of $C/T, T$ 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 during a frame over an incoming link i and have to be transmitted on the same outgoing frame of link j. Assuming unicast communication, we always have

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

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

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

In other words, we assume that the protocols ensure that the number of packets requesting the same outgoing frame is

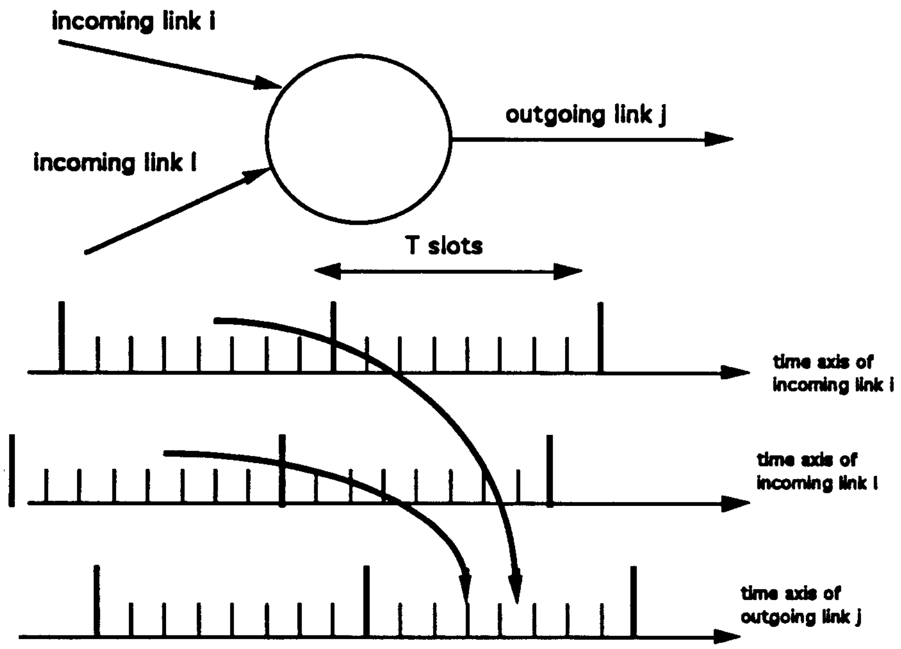

incoming link i
incoming link l
outgoing link j
T slots
time axis of incoming link i
time axis of incoming link l
time axis of outgoing link j

Fig. 2. Illustrates the incoming and outgoing frames at a node. Packets arriving in a particular frame of an incoming link that want to use the same outgoing link are sent over the same frame of the outgoing link.

always less than or equal to the duration of the frame. If (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 to delay 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 $R_{ij} = n_{ij}C/T$, (2) can be restated as

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

which simply requires that the sum of the average rates of the sessions using a link should be less than the link capacity. In Section V, we briefly describe wait-for-reservation and tell-and-go type of connection and flow control protocols that guarantee that (2) [or equivalently, (3)] holds at all network links.

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 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 pth subsequent frame, where p is a constant), as shown in Fig. 2.

Fig. 3 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 VI. The *Packer* in Fig. 3 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 II, we assume that the flow control protocol guarantees that the number $n_{ij}, i = 1, 2, \cdots, k, n_{ij} \in \{0, 1, \cdots, T\}$, of packets that arrive during frame $F(i)$ and are transmitted during frame $F(j)$ of link j satisfy (2) [or, equivalently, (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 link capacities of the order of 50 Gb/s and ATM-sized cells, the slot duration is approximately 10 ns, and each km of fiber can store about 500 cells. For $T = 1024$ (that is, 10 μs frames), the total length of the fiber per link needed for storage is $T(2T-1)/500 = 10^5$ km, which is clearly impractical. For a design using delay lines to be feasible, the number of delay elements has to be reduced. In what follows, we describe a construction that uses only $\log T$ (as opposed to $2T-1$) delay elements per link, with a total fiber length proportional to T (as opposed to $2T^2$).

The delay lines that implement the buffering system for a particular outgoing link j are depicted in Fig. 4 for the case $T = 4$. 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

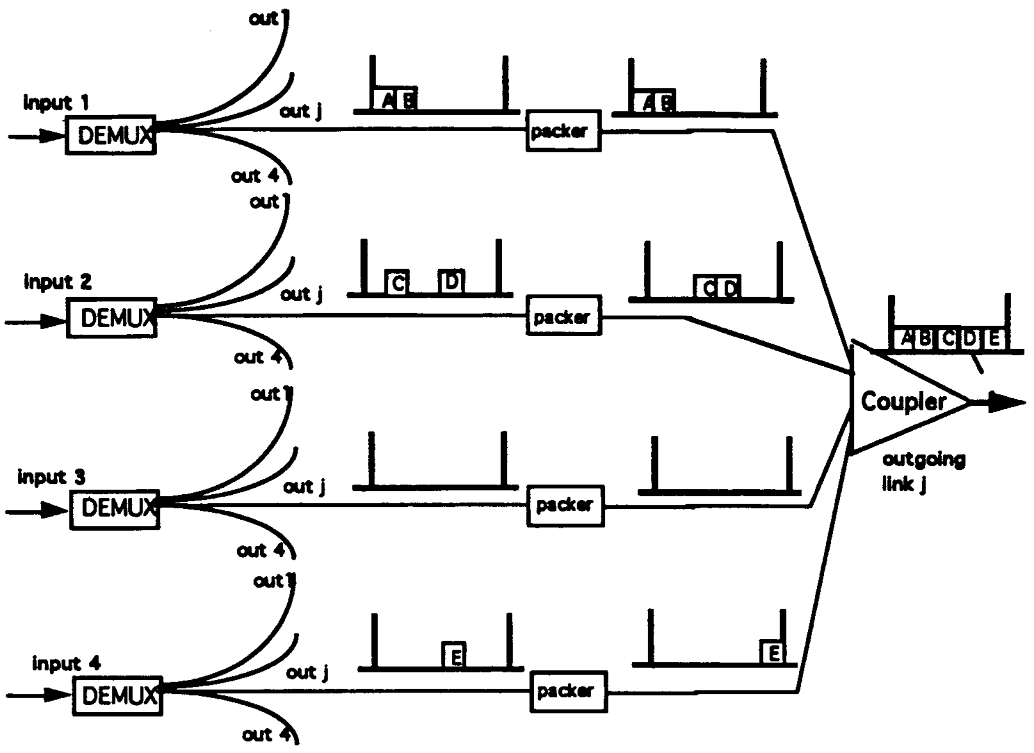


Fig. 3. Illustrates the packing switch. Only the details corresponding to output link j are shown.

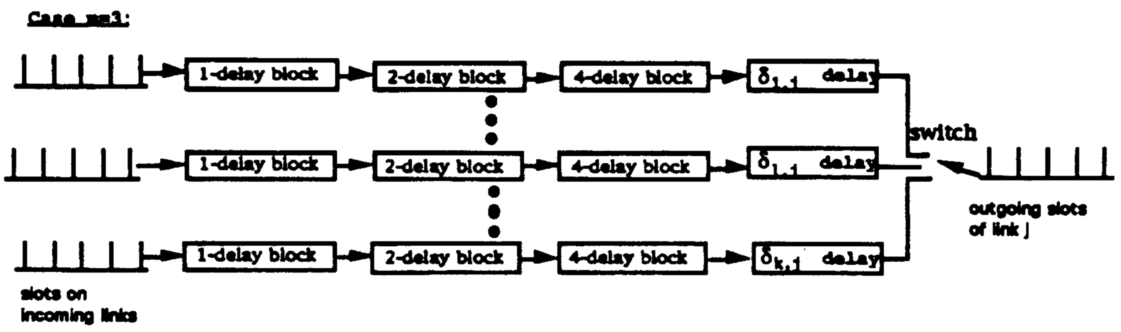


Fig. 4. We illustrate the output system for a particular outgoing link j. Each delay block can be implemented by a switch and an optical fiber of appropriate length, as shown in Fig. 5. Depending on the distance between the arriving and the departing slot of a packet, the state of each delay block is 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. Clearly, even though the frames on the outgoing links of a node can be synchronized, if desired, this is unrealistic to assume for the incoming links of the node, since it would require global synchronization and exact knowledge of the lengths of the links connecting different nodes.

$2T-1 = 2^m - 1$ 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. 4 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. 5 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. 4. We focus on a particular frame $F(j)$ of an outgoing link j. Consider a packet A that arrives in slot $x_A \in \{0, 1, \cdots, T-1\}$ 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 $r_A, r_A \in \{1, \cdots, n_i\}$, will be referred to as the *rank* of packet A). Then, according to the packing rule, packet A is assigned to slot $y_A = \Sigma_{l=1}^{i-1} n_{lj} + r_A - 1, y_A \in \{0, 1, \cdots, T-1\}$, of the outgoing frame $F(j)$ (see Fig. 6). As shown in the following theorem, when packets are assigned to outgoing slots according to the packing rule, no collisions occur at the outputs of the delay blocks.

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

Proof: Clearly, packets arriving on different incoming links will never collide, since they are routed through different delay lines, and they are assigned to different outgoing slots.

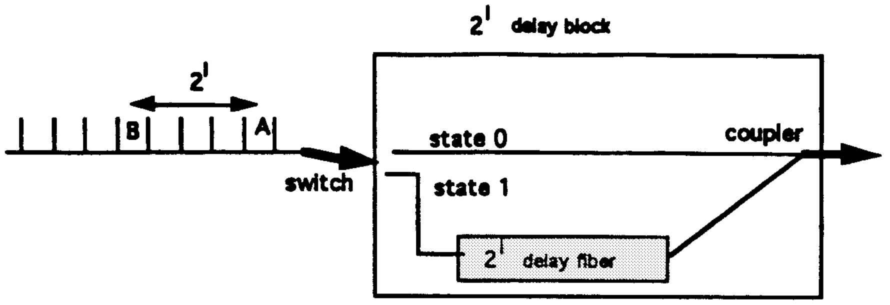


Fig. 5. We illustrate the design of a 2^l-delay block. A packet collision may occur in the case where packet B lags packet A by 2^l slots, and packet A passes through the upper branch (state 0), while packet B passes through the lower branch (state 1) of the 2^k-delay block. We prove in Theorem 1 that 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.

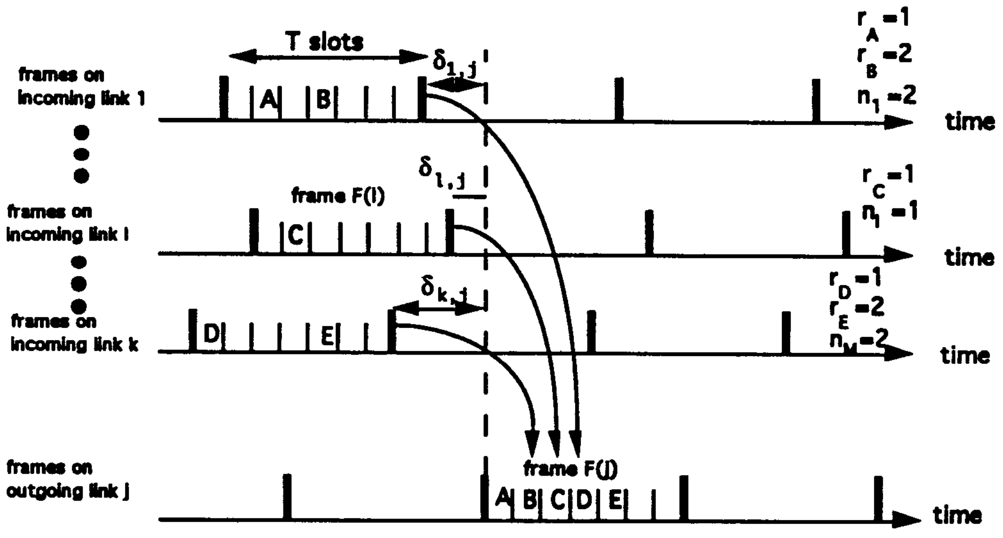


Fig. 6. We illustrate the incoming and outgoing frames at a node. Packets not intended for outgoing link j are not shown. Packets intended for link j are assigned to outgoing slots according to the packing rule described in the text.

Consider two packets A and B that arrive on incoming link i during slots x_A and x_B of the same frame $F(i)$, and they both have to be transmitted in frame $F(j)$ of outgoing link j. We let r_A and r_B be their ranks, and we assume (without loss of generality) that $r_A < r_B$. According to the packing rule, packets A and B are assigned to outgoing slots

$$y_A = \sum_{l=1}^{i-1} n_{lj} + r_A - 1 \tag{4}$$

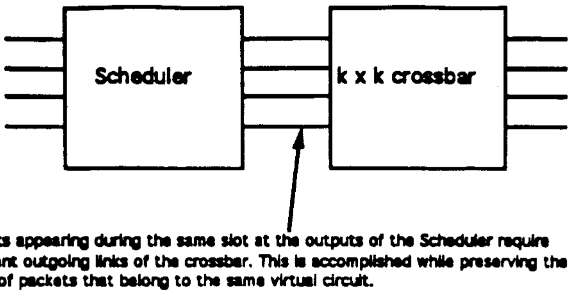
and

$$y_B = \sum_{l=1}^{i-1} n_{lj} + r_B - 1 \tag{5}$$

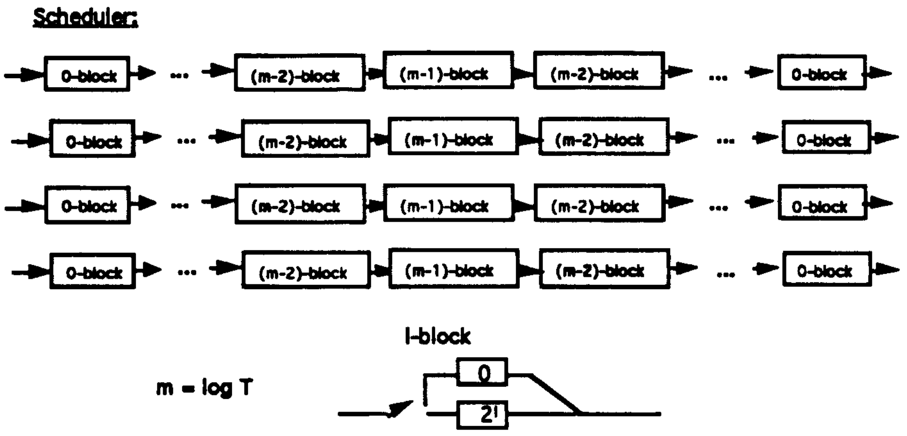
respectively. For packets A and B to collide at the output of stage 0, they should arrive at successive input slots (that is, we should have $x_B = x_A + 1$, which implies $r_B = r_A + 1$ and $y_B = y_A + 1$), and A should be delayed by 1 slot, while B should not be delayed at all at stage 0. This cannot happen because in that case A and B are assigned to successive slots in outgoing frame $F(j)$, which implies that they have to be delayed by the same amount, and their delay at stage 0 must be the same. Thus, packets A and B cannot collide at the output of stage 0.

We now generalize the previous argument to show that packets A and B cannot collide at the output of any stage $s > 0$. To show this, we let $(x_0^A, x_1^A, \cdots, x_{m-1}^A)$ and $(y_0^A, y_1^A, \cdots, y_{m-1}^A)$ be the binary representations of x_A and y_A, and $(x_0^B, x_1^B, \cdots, x_{m-1}^B)$ and $(y_0^B, y_1^B, \cdots, y_{m-1}^B)$ be the binary representations of x_B and y_B, respectively. We also let T_A^s [or T_B^s] be the slot, counting from the beginning of frame $F(i)$, at which A [or B] is transmitted at the output of stage s, and we let $(t_0^{A,s}, t_1^{A,s}, \cdots, t_m^{A,s})$ [or $(t_0^{B,s}, t_1^{B,s}, \cdots, t_m^{B,s})$, respectively] be its binary representation. Since the delay introduced at any subsequent stage $l, l > s$, is either 0 or 2^l, we have that $(t_0^{A,l}, t_1^{A,l}, \cdots, t_s^{A,l}) = (y_0^A, y_1^A, \cdots, y_s^A)$ for all $l > s$. In other words, at any stage after stage s, the s least significant bits of the slot in which a packet appears are identical to the s least significant bits of the outgoing slot to which the packet has been assigned and will finally appear. [For example, for $s = 0$ we have $t_0^{A,l} = y_0^A$, for all $l > 0$, which means that after stage 0, packet A will always appear at the output of a stage in an even slot, if its assigned outgoing slot y_A is even, and in an odd slot, if y_A is odd.] For packets A and B to appear during the same output slot of stage s, we should have that

$$\left(t_0^{A,s}, t_1^{A,s}, \cdots, t_m^{A,s}\right) = \left(t_0^{B,s}, t_1^{B,s}, \cdots, t_m^{B,s}\right)$$


Scheduler
k x k crossbar
Packets appearing during the same slot at the outputs of the Scheduler require different outgoing links of the crossbar. This is accomplished while preserving the order of packets that belong to the same virtual circuit.

(a)



(b)

Fig. 7. (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.

which can happen only if

$$\begin{aligned}\left(y_0^A, y_1^A, \cdots, y_s^A\right) &= \left(t_0^{A,s}, t_1^{A,s}, \cdots, t_s^{A,i}\right)\\ &= \left(t_0^{B,s}, t_1^{B,s}, \cdots, t_s^{B,s}\right)\\ &= \left(y_0^B, y_1^B, \cdots, y_s^B\right).\end{aligned}$$

Therefore, for packets A and B to collide at the output of stage s, the s least significant bits of y_A and y_B should be identical. This implies (since $y_A < y_B$) that $y_B \geq y_A + 2^{s+1}$, and [in view of (2)–(3)], $r_B \geq r_A + 2^{s+1}$. By the definition of the rank, we then have that $x_B \geq x_A + 2^{s+1}$, which means that there should be a delay of at least 2^{s+1} slots between the arrivals of A and B in the incoming frame $F(i)$. Since the first $s-1$ stages reduce this delay by at most $1+2+\cdots+2^{s-1} = 2^s - 1$ slots, the slots at which packets A and B appear at the output of stage $s-1$ will be separated by a distance of at least $2^{s+1} - 2^s + 1 = 2^s + 1$ slots. Since stage s cannot introduce delay larger than 2^s slots, it follows that packets A and B will not collide at the output of stage s. Q.E.D.

The loss introduced by the passive coupler at the right end of a delay block in Fig. 5 can be avoided by replacing the passive coupler by a two-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 (2) [or the equivalent condition of (3)] is satisfied. In Section V, we describe flow and control protocols which guarantee that this condition holds at all nodes of the network.

IV. The Scheduling Switch

In this section we describe the Scheduling Switch architecture, a block diagram of which is shown in Fig. 7. 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)th is equal to the number $n_{ij}, i = 1,2,\cdots,k, j = 1,2,\cdots,k,$ 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.*

From (2) to (3), we have

$$h \leq T \tag{6}$$

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

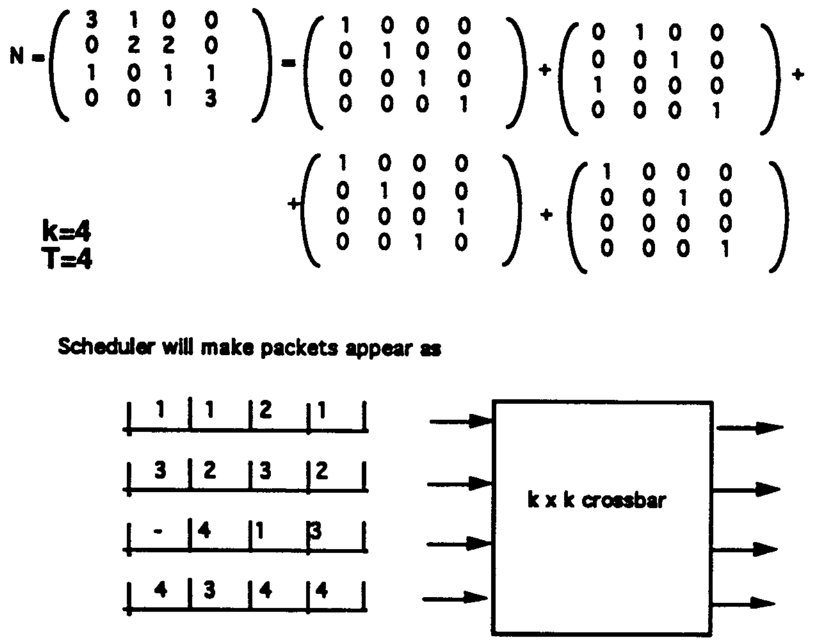


Fig. 8. A decomposition of a frame matrix as a sum of permutation matrices, and the corresponding assignment of packets to outgoing slots of the scheduler.

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 2: 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 3: 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 [20, p. 57]).

Theorem 2 (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 an algorithm for decomposing a perfect matrix into a sum of permutation matrices; the basic idea is to view 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.

Proof: The decomposition algorithm consists of h phases, such that during phase $s, s = 1, 2, \cdots, h$, the sth permutation matrix P_s is found. In particular, in phase s an optimal matching is found for bipartite graph (U, V, E_s) where $U = \{u_1, u_2, \cdots, u_k\}$ is the set of left nodes, $V = \{v_1, v_2, \cdots, v_k\}$ is the set of right nodes, and E_s is the set of edges of the bipartite graph. An edge (u_i, v_j) is present in the set E_s if and only if the (i, j)th entry of matrix $M - \Sigma_{q=1}^{s-1} P_q$ is nonzero, and row i or column j (or both) is a critical line of matrix $M - \Sigma_{q=1}^{s-1} P_q$. An optimal matching for the bipartite graph (U, V, E_s) can be found in time $O(k^{5/2})$ [18]. The permutation matrix P_s is then defined as the matrix whose (i, j)th entry is equal to one, if (u_i, v_j) is used in the matching, and zero otherwise. Q.E.D.

The following theorem is found in [2].

Theorem 3: 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 2 and 3 combined with (6) yield the following lemma.

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

$$N = \sum_{s=1}^{T} P_s \tag{7}$$

of at most T permutation matrices.

Fig. 8 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 sth 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 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. 7(b), 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, satisfying 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. 7(b) and described in detail in [15].

V. Network Protocols

The condition of (2) [or the equivalent condition of (3)], which ensures the lossless character of the design, can be enforced by using either a *wait-for-reservation virtual circuit* (WRVC) protocol or a *tell-and-go virtual circuit deflection* (TGVCD) protocol. In both protocols, each session S is assumed to arrive at a source with a specified bandwidth requirement R_S. A path with adequate residual capacity is then computed at the source based on (possibly outdated) topology and link utilization information that is available at the source at that time. After determining a route through the network, a setup packet is transmitted over the path to set the routing tables and reserve capacity at intermediate nodes. In a WRVC protocol, a reservation is successful only when (3) can be satisfied on all links on the session's path, and data transmission starts only after a positive acknowledgment is received at the source. Therefore, WRVC protocols can easily ensure that (2)–(3) hold, but this happens at the expense of a roundtrip pretransmission delay required for reservations. In the TGVCD protocol, proposed in [23], the source starts transmission shortly after transmitting a setup packet to the destination, achieving in this way a pipelining between the setup phase and the data transmission phase, and reducing the pretransmission delay to the minimum possible. If upon the arrival of the setup packet at the intermediate node the capacity available on the preferred outgoing link is not sufficient to accommodate the session, the setup packet and the data packets that follow it may have to be routed over a different, longer path; we then say that the session is *deflected*. Assuming that the total outgoing capacity is equal to the total incoming capacity of a node (or equivalently, that the total number of incoming slots is equal to the total number of outgoing slots in a frame), adequate capacity can always be made available on the outgoing links of an intermediate node to accommodate a new incoming session [23]. This, however, may happen at the expense of interrupting (preempting) existing sessions that originate at that node. Also, since the outgoing capacity that is available or may become available through the preemption of existing sessions originating at a node may not all belong to the same outgoing link, the session may have to be split into two or more subsessions of smaller rates, each of which is routed over a different path to the destination.

WRVC protocols tend to be inefficient in terms of link utilization, because they reserve capacity for more time than the minimum required. This is because capacity is reserved at a node starting at the time the setup packet arrives at the node, which is at least one round-trip delay earlier than the time at which the first data packet will arrive. Furthermore, the pretransmission delay required for connection setup is often significant compared to the delay requirements of the session. Since in tell-and-go protocols link capacity is reserved for duration only slightly larger than the holding time of a session, and is available for the remaining time, the TGVCD protocol has an efficiency advantage over WRVC protocols. This is particularly important for high-speed networks where propagation times are often comparable to the typical holding time of sessions. An important advantage of the TGVCD protocol over *datagram* deflection schemes is that it significantly reduces the need for packet resequencing at the destination. This is because deflections in the former occur on a per session basis (or a per subsession basis if session splitting is required to find adequate capacity on the outgoing links), while in the latter they occur on a per packet basis. Consequently, message reassembly at the destination, which is one of the main problems of datagram deflection schemes [17], is easier to accomplish with the tell-and-go virtual circuit deflection protocol: when a session is split, blocks of data (each of which is in the correct order) have to be resequenced, instead of individual packets. Moreover, in the TGVCD protocol, data packets are routed through a switch based on the virtual path identifier (VPI) they carry and the routing tables established by the setup packet, maintaining in this way one of the main advantages of virtual circuit switching; in contrast, in datagram deflection schemes, routing decisions are made individually for each data packet making the switch processor a potential bottleneck of the design.

For the TGVCD protocol to work, the time gap between the transmission of the setup packet and the transmission of the first data packet from a source has to be larger than the maximum number of hops allowed for the session times the processing time of a setup packet at a node. In other words, the time gap must be at least as large as the minimum time by which the connection setup phase and the data transmission phase should be separated to ensure that data packets do not overpass the setup packet. Since we are interested in almost-all optical switching, this delay should be large enough to permit the electronic processing of the setup packet, without it being overpassed by the data packets, which will mostly remain in the optical domain (except for their VPI, which will have to be processed electronically).

VI. Reading the Packet Header

For packet switching, the 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 [12] 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 III and IV by the control unit of the

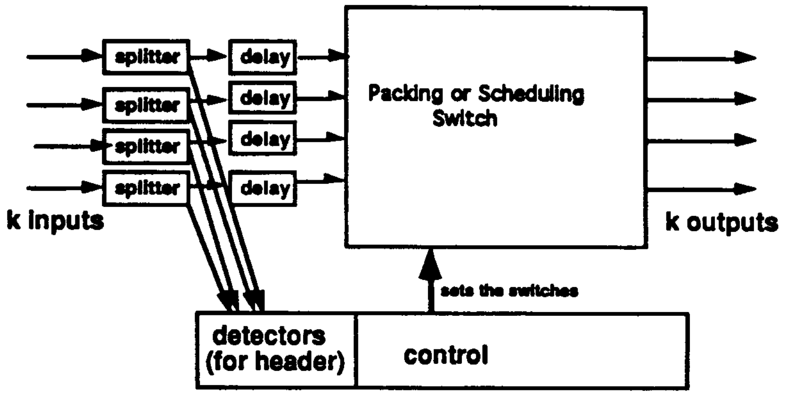


Fig. 9. Block diagram of the switch.

switch to determine the state at which each of the two-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 (see Fig. 9).

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 per slot, 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 that the time axis is divided into frames of T slots each, as described in Section II, 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 (e.g., a given slot in each frame) 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 III, 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 3, 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

$$M = N + E = \sum_{s=1}^{T} P_s. \tag{8}$$

The permutation matrices $P_s, s = 1, 2, \cdots, T$, can be used (in the way described in Section IV) to set the state of the elementary two-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 the (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 $M = \Sigma_{s=1}^{T} P_s$ 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 ith row sum and the jth column sum of N are both strictly less than T, and there exist $p, q \in \{1, 2, \cdots, k\}$, such that $e_{pj} > 0$ and $e_{iq} > 0$. By defining the new target matrix

$$M^{\text{new}} = M + 1_{ij} - 1_{iq} - 1_{pj} + 1_{pq} \tag{9}$$

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 $N = \Sigma_{s=1}^{T} P_s$ 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 $P_m + P_n + 1_{ij} - 1_{iq} - 1_{pj} + 1_{pq}$ is perfect with critical sum equal to 2, it can be decomposed in time $O(k^{5/2})$ as the sum of two permutation matrices, so that

$$P_m + P_m + 1_{ij} - 1_{iq} - 1_{pj} + 1_{pq} = \tilde{P}_m + \tilde{P}_n. \tag{10}$$

Equations (8)–(10) 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. 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 has smaller hardware requirements than the packing switch, and it appears to be an attractive architecture for building both packet-switched and circuit-switched almost all-optical high-speed networks.

References

[1] B. Awerbuch, I. Cidon, I. Gopal, M. Kaplan, and S. Kutten, "Distributed control for PARIS," in *Proc. 9th Annu. ACM Symp. Principles Distributed Comp.*, 1990, pp. 145–160.

[2] G. Bongiovanni, D. Coppersmith, and C. W. Wong, "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.

[3] F. Borgonovo, L. Fratta, and J. Bannister, "On the design of deflection-routing networks," *IEEE INFOCOM'94*, Toronto, Ont., Canada, June 1994, pp. 120–129.

[4] I. Cidon and I. Gopal, "PARIS: An approach to integrated high-speed private networks," *Int. J. Digital and Analog Cabled Syst.*, vol. 1, no. 2, pp. 77–86, Apr.–June 1988.

[5] I. Chlamtac and A. Fumagalli, "An optical switch architecture for Manhattan Street networks," *J. Select. Areas Commun.*, vol. 11, pp. 550–559.

[6] I. Cidon, I. S. Gopal, and A. Segall, "A connection establishment in high-speed networks," *IEEE/ACM Trans. Networking*, vol. 1, pp. 469–481, Aug. 1993.

[7] R. L. Cruz and J.-T. Tsai, "COD: Alternative architectures for high speed packet switching," *IEEE/ACM Trans. Networking*, vol. 4,Feb. 1996.

[8] A. Eckeberg, D. Luan, and M. Lucantoni, "An approach to controlling congestion in ATM networks," *Int. J. Digital and Analog Commun. Syst.*, vol. 3, no. 2, pp. 199–209, 1990.

[9] S. J. Golestani, "Congestion-free communication in high-speed packet networks," *IEEE Trans. Commun.*, vol. 39, no. 12, Dec. 1991.

[10] Z. Haas, "The "Staggering switch: An almost-all optical packet switch," pp. 1593–1599.

[11] Z. Haas and D. R. Cheriton, "Blazenet: A packet-switched wide-area network with photonic data path," *IEEE Trans. Commun.*, vol. 38, pp. 818–829, June 1990.

[12] Z. Haas and R. D. Gitlin, "Field coding: A high-peed 'almost-all' optical interconnect," in *Proc. Twenty Fifth Annu. Conf. Inform. Sci. Syst.*, Baltimore, MD, Mar. 20–22, 1991.

[13] D. K. Hunter and D. G. Smith, "New architectures for optical TDM switching," *J. Lightwave Technol.*, vol. 11, Mar. 1993.

[14] ——, "An architecture for frame integrity optical TDM switching," *J. Lightwave Technol.*, vol. 11, no. 5/6, 1993.

[15] P. J. Lin and R. G. Gallager, "Time slot interchange using fiber delay lines," preprint.

[16] N. F. Maxemchuk, "Comparison of deflection and store-and-forward techniques in the Manhattan Street and shuffle-exchange networks," in *INFOCOM '89*, vol. 3, pp. 800–809, Apr. 1989.

[17] ——, "Problems arising from deflection routing: Live-lock, Lock-out, congestion and message reassembly," in *Proc. NATO Workshop Architecture and High Performance Issues of High Capacity Local and Metropolitan Area Networks*, France, June 1990.

[18] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization, Algorithms, and Complexity*. Englewood Cliffs, NJ: Prentice Hall, 1982.

[19] P. O'Reilly, "The case for circuit switching in future wide bandwidth networks," in *Proc. IEEE Int. Conf. Commun.*, vol. 2, pp. 899–904, June 1988.

[20] H. J. Ryser, *Combinatorial Mathematics*. Rahway, NJ: The Mathematical Association of America, 1965.

[21] E. A. Varvarigos and V. Sharma, "An efficient reservation virtual circuit protocol," *Computer Networks and ISDN Systems*, to be published; also presented in part in Proc. Int. Symp. on Information Theory, Sept. 1995.

[22] ——, "The ready-to-go virtual circuit protocol: A loss-free connection control protocol for the thunder and lightning network," in *Proc. IEEE Global Telecommun. Conf. (GLOBECOM'95)*, pp. 450–456.

[23] E. A. Varvarigos and J. Lang, "A novel virtual circuit deflection protocol for multigigabit networks and its performance for the MS topology," in *Proc. IEEE Global Telecommun. Conf. (GLOBECOM'96)*, 1996, pp. 1544–1548.

Emmanouel (Manos) Varvarigos was born in Athens, Greece, in 1965. He received the B.S. degree in electrical and computer engineering from the National Technical University of Athens, in 1988 and the M.S. and Ph.D. degrees in electrical engineering and computer science from the Massachussetts Institute of Technology, Cambridge, in 1990 and 1992, respectively.

In 1990, he worked for Bell Communications Research, Morristown, NJ, on CDMA systems. In 1992, he joined the Department of Electrical and Computer Engineering at the University of California, Santa Barbara, where he is currently an Associate Professor. His research interests are in the areas of high-speed networks, performance analysis, parallel and distributed computation, and mobile networks.

Dr. Varvarigos has received an NSF Research Initiation Award, the first panhellenic prize in mathematics, and five Technical Chamber of Greece Awards.