Scheduling Algorithms for Optical Packet Fabrics

K. Kar[†] **T. V. Lakshman**[*] **D. Stiliadis**[*] **L. Tassiulas**[†]

† ECE Department, University of Maryland, College Park, MD 20742, USA

∗ Bell Labs, Lucent Technologies, 101 Crawfords Corner Road, Holmdel, NJ 07733, USA

Abstract

Utilizing optical technologies to build packet fabrics for high-capacity switches and routers has several advantages in terms of scalability, power consumption, and cost [1]. However, several technology related problems have to be overcome to be able to use such an approach. The reconfiguration times of optical crossbars are longer than those of electronic fabrics, and end-to-end clock recovery in such systems add to the reconfiguration overheads. Both these problems can limit the efficiency of optical packet fabrics. In addition, existing work on input-buffered switches mostly assumes fixed size packets (referred as *envelopes* in this paper). When fixed size switching is used for IP networks where packets are of variable size, the incoming packets need to be fragmented to fit the fixed size envelopes. This fragmentation can lead to, possibly large [2], loss of bandwidth and even instability. This paper addresses all of the above issues by presenting packetization and scheduling techniques that allow optical packet fabrics to be used within switches and routers. The proposed scheme aggregates multiple packets in a single envelope, and when used in combination with proper scheduling algorithms, it can provide system stability as well as bandwidth and delay guarantees. As a result of the aggregation method, the reconfiguration frequency required from the optics is reduced, facilitating the use of optical technologies in implementing packet switch fabrics.

I. Introduction

The deployment of Dense Wavelength Division Multiplexing (DWDM) technologies has expanded fiber transmission capacities to an extent that electronic routers and switches are the capacity bottlenecks. Consequently, there is need for building faster routers. Moreover, the need to shift from a best-effort network to one providing Service Level Assurances (SLAs) requires routers and switches that support traffic management and protection mechanisms. This implies that switch fabrics used in routers must be able to provide bandwidth and delay guarantees in addition to being faster and faster.

Consider a general model of a high-capacity router or switch. The system consists of a set of port cards interconnected through a central switching fabric. Since the requirements for bandwidth and number of ports is increasing at a tremendous rate, the number of port cards that a single switch/router must accommodate is increasing as well. Therefore it is most likely that port-cards will have to be distributed over several racks of equipment and their interconnection to the switch fabric will either be through serial high-speed electronic connections like Gigabit Ethernet or through optical fiber connections (Figure 1). When such a system architecture is deployed, packets that arrive from external

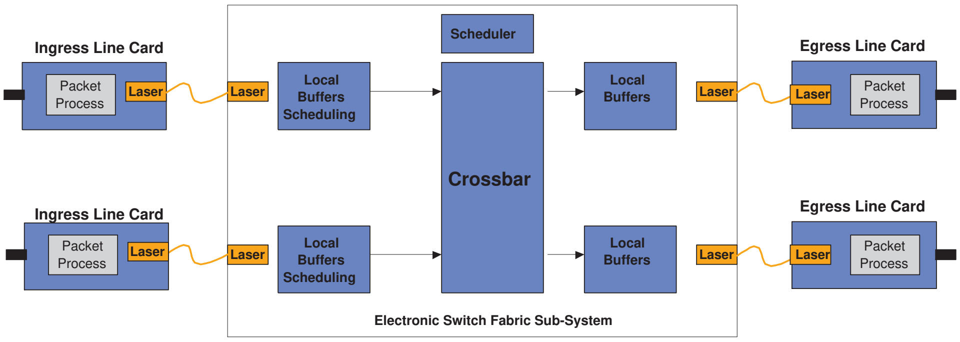

Fig. 1. Architecture of a router system with a large number of line cards distributed over several racks of equipment

interfaces are processed and buffered in the electronic domain. Data are then transmitted over a fiber link to the switch fabric, where they are again processed in electronics. Once they are switched, a different fiber connection is used to transmit the packets to the egress logic of the line cards, where they are again processed and buffered in electronics, before being finally transmitted in optics.

It is clear from the above description that the system has a series of opto-electronic conversions, that translate in significant cost and power requirements. An obvious evolution of this architecture is to try to replace the central opto-electronic fabric with a passive optical fabric and avoid at least two of the opto-electronic conversions (Figure 2). Technologies such as tunable lasers and Array Waveguide Routers can be used to implement the optical fabric [3], [1]. In this configuration the central fabric can be considered as a non-blocking crossbar, and the system is based on a input-output buffered architecture. However, optical technologies impose several design challenges because of relatively low reconfiguration speeds and loss of bandwidth due to clock recovery times.

Our main goal in this work is to address the main design challenges of such a system, and design scheduling algorithms that enable optical technologies to be used at the core of packet switches and routers. We first introduce the technology problems in designing optical fabrics and how these can affect the throughput of the system. We then show how we can use a simple packetization scheme to alleviate the throughput problems. A simple application of this scheme, however, can lead to unnecessarily large latencies at low loads. We extend our scheme by providing a set of scheduling algorithms that can reduce the latencies, and provide bandwidth and delay guarantees.

II. Background and Motivation

An input-output buffered switch is based on a non-blocking crossbar and a fabric scheduler or *arbiter* that controls the operation of the crossbar, by matching input and output ports for transmitting packets. There has been a lot of research on applying this architecture to switches that handle fixed size packets [4], [5], [6], [7]. In order to avoid head-of-line blocking, most of these architectures maintain separate virtual queues for each pair of input/output ports, and the arbiter requires a complete knowledge of the state of the system. In other words, the arbiter must know whether a packet is waiting to be transmitted between any pair of input/output ports. Once the scheduler has collected this state information, it will i) match input and output ports, ii) program the crossbar switch and, iii) notify all input ports which

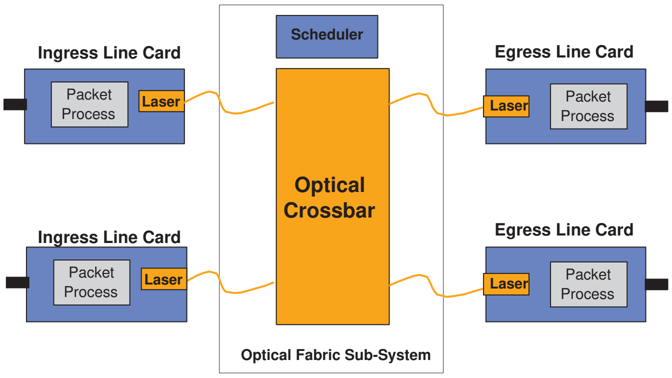

Fig. 2. Architecture of a system based on an optical fabric

output port they should send packets to.

A. *Technology Issues*

There are several issues that arise when one tries to implement such an architecture in the context of optical packet fabrics.

1. The rate of switching of the crossbar (or reconfiguration frequency) depends on the size of packets that are switched (referred as *envelopes* in this paper). Previous works have assumed that the size of these switched packets is set equal to the minimum packet size of the network. In the case of IP networks this translates to 40 bytes. One can easily calculate that a switch fabric with 10Gbps ports must switch every 40ns and a fabric with 40Gbps ports must switch every 10ns. However, most optical technologies have reconfiguration overheads much higher than these. For example the tuning overhead of lasers can be in the order of 20-30ns [1].
2. If the crossbar is a passive device and does not process the signals, as is the case in optical packet fabrics, the transmitter and receiver devices must be re-synchronized every time the crossbar is switching. In other words, every time an input port transmits data to a new output port, the receiving port must synchronize its clock to the transmitting clock. Even if a common reference clock is distributed across the whole system, the receiving port must adjust the phase of the clock to accommodate slight variations in fiber lengths or signal propagation times. This burst clock recovery operation usually takes several bit times and for most common transceivers, this time could be in the range of 10-20 ns, or even higher.
3. The central scheduler must be able to synchronize all the ports as well as the crossbar, so that they all switch at the same time. Because of possibly long and variable distances between the arbiter and the line cards, especially when the system is distributed across several racks of equipment, one has to account for clock skews (i.e., the signals do not arrive at all line cards at the same time). Thus, extra margins must be provided to the switching times to avoid data loss.

Even though one can devise several schemes to optimize the above overheads, the reconfiguration time is still much larger than the transmission time of a small packet. The total overhead of switching, clock recovery, and skew adjust-

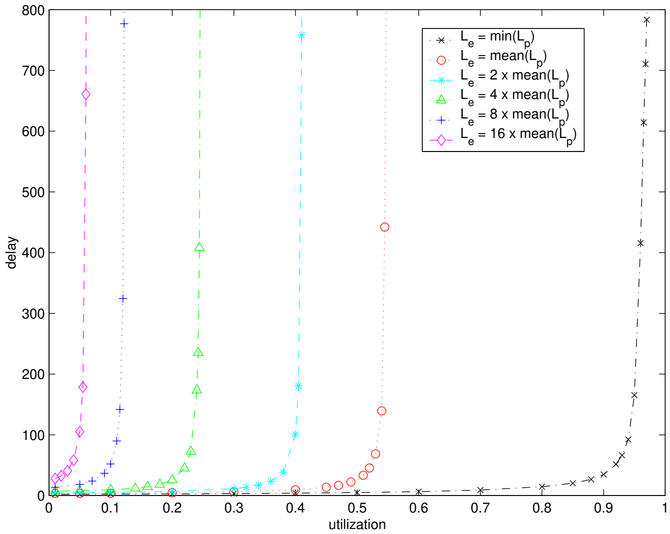

Fig. 3. Delay vs. Utilization for iSLIP for various envelope sizes

ments can be in the range of 50-100ns. An envelope transmission time for a 40Gbps fabric port is less than 10ns. This means that only a very small percentage of the bandwidth is used (10ns out of the 110ns), and the fabric can sustain high throughput only with a very large speed-up. Therefore, optical based fabrics can only become useful if one can find some method to reduce the reconfiguration frequency.

In addition to the technology problems, we also have to consider some performance issues.

B. Variable Size Packets

As we briefly mentioned earlier, most previous approaches consider only issues related to the scheduling of fixed size packets, and do not address the problems caused by arriving traffic being composed of variable size packets with the local packet size distribution being generally not known, as is the case in IP networks. This mismatch between arrival and transfer units requires packets to be fragmented into envelopes. A simple fragment-and-schedule packing of packets can lead to significant loss of fabric bandwidth when the packet size is not an integral multiple of envelope size [2]. It was suggested that increased speed-up could be used to compensate for this bandwidth loss. If packets are always larger than envelopes, it can be easily seen that the maximum loss of bandwidth occurs when all the packets are of length $(L_e + 1)$, where L_e is the envelope-length. Then, as suggested in [2], we could use a $\frac{2L_e}{L_e+1} \approx 2$ times additional speed-up to offset the bandwidth loss caused by mismatch of packet and envelope sizes. However, this is neither a satisfactory nor a general solution, since increasing the bandwidth requirements translates to a further increase in the reconfiguration frequency of the crossbar, and this is not acceptable.

Figure 3 shows the average packet delays, obtained by simulation, for various envelope-sizes. We assume a 16x16 switch, using the 16 iterations of the iSLIP scheduling algorithm [4]. Assume that L_e denotes the envelope-size and L_p denotes the packet size. Packet size distribution is based on measurements from [8] with 60% of the packets being 44 bytes, 20% of the packets 552 bytes and the rest 1500 bytes. The minimum packet size is 44 bytes, denoted by $min(L_p)$

in the figure, while the average packet size is approximately 436 bytes, denoted by $mean(L_p)$ in the figure. Packet interarrival times are exponentially distributed and traffic is symmetric across all inputs and outputs. Delay is in terms of the transmission time of an average size packet on the input/output links.

We can notice that contrary to the constant size packet case, for arbitrary packet sizes, the system may become unstable, particularly when the envelope size is much larger than the minimum packet size, as evident in Figure 3. From the figure, we see that when the envelope size is equal to the minimum packet size, we obtain almost 100% throughput, while when the envelope size is equal to the mean packet size or larger, the system becomes unstable at about 50% load or even less. It can be easily seen that if the speed-up is κ and the link speed is r, and all the packets are of the same length L_p, the maximum load that can be supported is $\frac{\kappa L_p}{L_e} r$, for $L_e > L_p$, and $\frac{\kappa L_p}{\lceil \frac{L_p}{L_e} \rceil L_e} r$ for $L_e < L_p$.

C. Envelope Size Choice and Trade-offs

The discussion in the previous two sections illustrates a clear trade-off. On one hand, in order to reduce the bandwidth loss due to fragmentation, the envelope size must be matched to the predominant small packet size. For IP networks, which have a large number of small packets due to TCP acknowledgments [8], the optimal envelope size will be 40 or 64 bytes (for Ethernet interfaces). On the other hand, technology issues prevent us from using small packet sizes in the case of optical fabrics, since a large percentage of time would be wasted in reconfiguring the fabric.

In addition to the optical fabric limitations, small envelope sizes make the implementation of the switch arbiter a challenging task as well. The complexity of the arbitration algorithm can range from $O(\log N)$ for simple bipartite matching to $O(N^3 \log N)$ for maximum weighted matching, where N is the number of ports [6], [7]. If a speed-up of two is utilized, a simple bipartite matching algorithm can provide 100% throughput, but the time available for the arbitration algorithm to complete a scheduling decision is reduced to half [9]. An input-output buffered switch can provide the same performance as an output buffered switch with a speed-up of two, but the arbitration algorithm has complexity of at least $O(N^2)$ [10], [11], [12]. As in the previous case, the speed-up requirement translates to a smaller time available for the execution of the arbitration algorithm. Note, that in a hardware implementation, reduction of the available time by a factor of two poses a substantial problem, although the difference does not seem significant asymptotically. It pretty much translates to a requirement of doubling the operating frequency of the arbiter, which might not be practically achievable. A method that can reduce the reconfiguration frequency will also provide a longer time interval for completing an arbitration decision, and thus allow system designers to implement sophisticated arbitration algorithms. This will result to better system performance.

Our goal in this work is to be able to demonstrate the feasibility of using ultra-fast optical crossbars with sophisticated arbitration mechanisms as packet fabrics with the following properties:

1. Reconfiguration frequency that is independent of the switch bandwidth and sufficiently low to allow the use of efficient scheduling algorithms and advanced transmission technology.
2. Efficient bandwidth management for variable size packets, without bandwidth loss and with QoS guarantees.

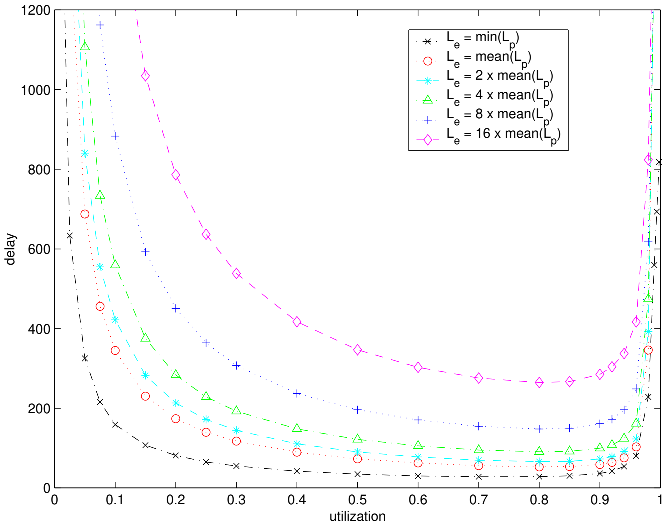

Fig. 4. Delay vs. Utilization when only full envelopes are scheduled

3. Power and space efficiency.
4. Low cost.

III. Achieving maximum throughput with large envelopes

There is a very simple packetization method that allows us to avoid the fragmentation problem and at the same time reduce the required reconfiguration frequency. The loss of bandwidth occurs when some partially filled envelopes are transferred over the fabric. These correspond to the last fragment of a packet. The first step to prevent this loss then is to allow the packing of multiple packets into an envelope. This may require that a packet be fragmented at an arbitrary position to pack it into an envelope and we assume this for the rest of the paper. Thus, if $L_p = 1.5L_e$, then we can fit 2 packets in 3 envelopes, and if $L_p = 0.5L_e$, we can fit 2 packets in 1 envelope.

We can avoid loss of fabric bandwidth by using the following method. Multiple packets are packed into an envelope, and an envelope is eligible for transmission only when it is full. Partially-full envelopes do not participate in the scheduling process and are never transmitted over the fabric. Thus, all the envelopes scheduled across the switch are full envelopes. Figure 4 shows the avg delay vs. utilization plots for this scheme for various envelope sizes, when the fabric scheduler is iSLIP [4], and all other conditions are similar to those for Figure 3. As expected, this simple modification translates to high switch fabric utilization, irrespective of the envelope size.

However, this simple approach has an obvious drawback: a packet may have to wait for a very long time (even for ever), if the corresponding envelope is not full, and therefore it is never eligible for transmission. This explains the high delays seen in Figure 4 at low loads (under heavy load, most envelopes are immediately filled up, and thus do not need to wait long to become eligible for transmission).

Note that when a particular port is lightly loaded, we can release partially filled envelopes to prevent delay accumulation. At low loads, wasting port bandwidth is not an issue since there is plenty of bandwidth available. One approach is to use the “excess switch bandwidth” to serve partially filled envelopes. In this approach, we first arbitrate between input and output ports accounting only for full envelopes. If some ports are still unmatched after this process is complete, we repeat the arbitration process accounting for partially full envelopes as well. This scheme leads to better delay performance over a broad range of utilization factors. (This scheme is discussed and evaluated in more detail later in this paper.)

Another alternative for preventing delay accumulation is to incorporate a time-out for partially filled envelopes. Expiration of this timer causes the arbiter to release partially filled envelopes. An intelligent setting of this timer is possible if the second-order statistics of the traffic is known. Without traffic assumptions, adaptive mechanisms are needed to configure the time-out value and it is not clear how this can be done. However, a main drawback of this mechanism is that it can not guarantee stability under all traffic patterns.

A. Stability of Common Scheduling Algorithms

It is straightforward to extend the results for stability derived for constant packet size crossbar switches in [9] to the case of envelope switches as described above. The results in [9] only require convergence of the arrival process, i.e.,

$$\lim_{n\to\infty} \frac{A_{(i,j)}(n)}{n} = \lambda_{(i,j)}, \tag{1}$$

where $A_{(i,j)}(n)$ is the cumulative number of packets that have arrived at input i destined to output j until time n, and $\lambda_{(i,j)}$ is the average rate between input i and output j. If no input or output port is oversubscribed, the above condition is sufficient to prove stability of input buffered switches with no speed-up and deploying maximum weighted bipartite matching, or of an input/output buffered switch with a speed-up of 2 and deploying maximal weighted matching. Assume an input queued switch as described above, where envelopes are eligible for transmission only when they are full. It is easy to see that the envelope arrival process $B_{(i,j)}(n)$ in the virtual output queues will satisfy the convergence requirement if the packet arrival process does so. This follows from the fact that for every n,

$$A_{(i,j)}(n) - L_e \leq B_{(i,j)}(n) \leq A_{(i,j)}(n), \tag{2}$$

where L_e is the envelope size. With the envelope arrival process satisfying the needed technical conditions, and the envelope switch being similar to the switch considered in [9] and under the same load constraints, it is possible to achieve 100% throughput for the envelope switch. If the envelope switch is stable, the total amount of data buffered in the envelope switch is finite. We also have data buffered in partially filled envelopes. Since the amount of data buffered at each input that is not eligible for transmission is less than one envelope for each input/output pair (or N^2 envelopes total), and thus finite as well, the total queue size in the system is also finite. Thus, the whole system is stable.

If we use the proposed packetization technique, any input buffered or input-output buffered switch that can provide 100% throughput for constant size packets will also provide stability for variable size packets and arbitrarily large envelopes, without any additional speed-up requirements. Note, however, that stability does not necessarily mean small delays or bandwidth guarantees, and this is the topic of the next section.

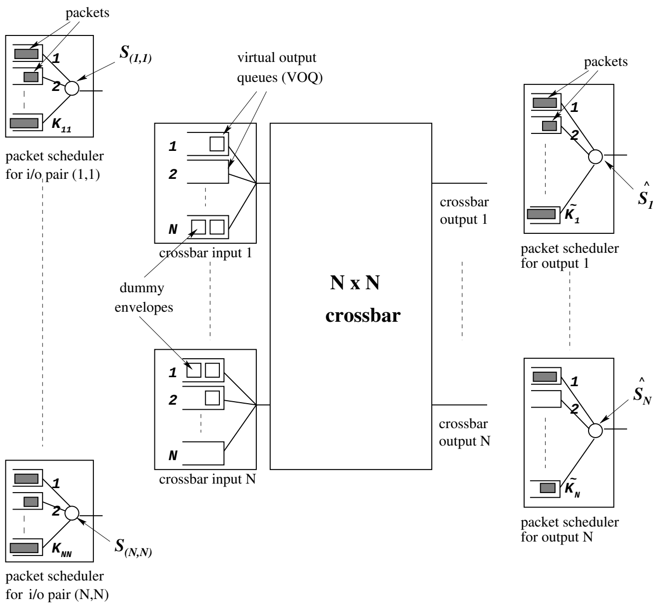

Fig. 5. The overall architecture of the switch for achieving bandwidth and delay guarantees

IV. ACHIEVING BANDWIDTH AND DELAY GUARANTEES

In this section, we first describe a switch architecture that can provide delay and bandwidth guarantees. We then derive a scheduling model for this switch architecture and use it to prove delay bounds for individual flows as a function of the scheduler properties such as service burstiness, and worst-case fairness index. Our goal in providing these bounds is to identify properties that the individual schedulers in our switch architecture must satisfy so as to enable the switch to provide good delay bounds despite using large envelopes. We do not give a specific choice of schedulers in this section. Specific scheduler choices so as to achieve delay bounds independent of the number of switch ports are given in the next section. Since providing delay guarantees requires that flows have bounded burstiness, we will assume that flows are leaky-bucket constrained, although a different measure of burstiness can be used as well.

A. Switch Architecture for Bandwidth and Delay Guarantees

Figure 5 shows the basic architecture that we consider. The switch is based on a crossbar fabric, and does virtual output queueing, i.e., separate virtual queues are maintained for every output at each input. For each input-output pair (i, j), there is a dedicated packet-scheduler $S_{(i,j)}$. The scheduler $S_{(i,j)}$ maintains separate queues for each individual class or flow of packets between input i and output j. The virtual output queues (VOQs) of the switch do not buffer real packets, but are used to buffer *dummy* envelopes. These dummy envelopes, each of length L_e, are just logical entities which are inserted into the VOQs in such a way that whenever the scheduler $S_{(i,j)}$ is backlogged with packets, the VOQ for (i, j) also remains backlogged with these dummy envelopes. The size of the dummy envelope queue for (i, j), denoted as $DQ_{(i,j)}$ is equal to $DQ_{(i,j)} = \lceil Q_{(i,j)}/L_e \rceil$, where $Q_{(i,j)}$ is the total queue size of packets between i and j. The crossbar arbiter makes its scheduling decisions based on these dummy envelopes. When the crossbar arbiter allots a slot for the transfer of an envelope of (i, j) across the switch fabric, the scheduler $S_{(i,j)}$ is responsible for scheduling an envelope-length of packets in that slot, choosing packets from the flows between input i and output j. Note that $S_{(i,j)}$ serves at a constant rate κr during that slot, where κ is the speed-up of the switch and r is the link speed, and hence can transfer packets up to a total length of L_e during the slot. However, the slot may not be utilized completely, if all the queues at $S_{(i,j)}$ are empty for some interval during that slot. Note that in the process of transferring packets across the switch fabric, the packets can be fragmented. Therefore, after an entire packet has been transferred, it is reassembled (the reassembly buffers are not shown in the figure), and placed in the per-flow queues at the corresponding output. The output packet scheduler, $\hat{S}_j$ (there are N of them, one for each output), is responsible for scheduling the packets of flows destined for output j, from the per-flow queues at the output onto the output link.

For the rest of the paper, unless mentioned otherwise, just by "envelope" we will refer to the dummy envelopes, and by "envelope-queue" we will refer to the VOQs, which buffer the dummy envelopes.

B. Scheduling Model for Switch Architecture

An abstract model of the overall switch scheduling system, as seen by the packets of a flow n going between input port i and output port j, is shown in Figure 6. Here, the crossbar arbiter (scheduler) is modeled as a single server S. The server is associated with N^2 envelope-queues, and serves envelopes (note that S can schedule up to N envelopes at the same time). During the time when an envelope of (i, j) is being served by S, the server $S_{(i,j)}$ serves packets of flows going from input i to output j. Note that both the servers S and $S_{(i,j)}$ are, in general, variable-rate servers. If the total amount of packet-traffic scheduled by $S_{(i,j)}$ during an envelope-slot is equal to L_e, then we call the corresponding envelope a *full* envelope, otherwise we call it a *partially filled* envelope. The "loss of bandwidth" is essentially because of the partially filled envelopes.

As mentioned before, whenever $S_{(i,j)}$ is backlogged, the envelope-queue for (i, j) at S remains backlogged too. However, the converse is not necessarily true. When S is "busy serving the empty portion of an partially filled envelope" of (i, j), the envelope-queue for (i, j) is backlogged at S, but $S_{(i,j)}$ is not. Note that the schedulers S and $S_{(i,j)}$, representing the transfer of packets across the switch fabric, form a two-level hierarchical system, the server S being at the upper level, and the servers $S_{(i,j)}$ being at the lower level (for a detailed discussion on hierarchical fair queueing systems, see [13]). However, there is an important difference between our two-level hierarchical scheduler (which, for the rest of the paper, will simply be referred as *switch scheduler*) and those described [13] : in this case, the scheduler at the upper level serves in terms of fixed size envelopes, and not packets. Note that this operation may result in significant bandwidth loss and unfairness if the crossbar scheduler S is not selected correctly.

C. Delay Analysis and Required Scheduler Properties

Before we proceed to derive the packet delay-bound for the scheme just described, we state the assumptions, the notation and some definitions to be used in the analysis.

Assumptions and Notation: In the following, we consider a packet to have arrived at (departed from) a server when the last bit of the packet has arrived at (departed from) the server. Also, we consider a packet to be backlogged at a server

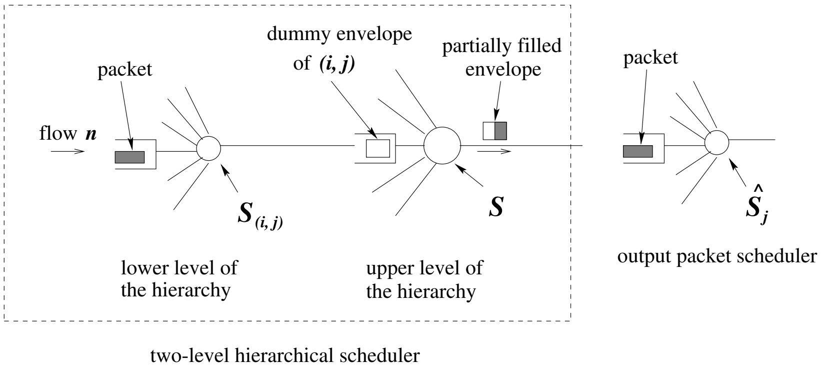

Fig. 6. Scheduling model of the architecture. The two-level hierarchical scheduler models the transmission of the packets over the switch fabric. The output packet scheduler models the transmission of the packets on the output links, after reassembly.

when it is queued at the server, including both the arrival and departure instants. A flow is backlogged when any of its packets are backlogged, while a server is backlogged when any of its flows are backlogged. In the following, $Q_m(t)$ denotes the queue-length for flow m at time t. For any server s, let $W_s(t_1, t_2)$ denote the amount of traffic served by the server in the interval $[t_1, t_2]$, and for any flow m, let $W_m(t_1, t_2)$ denote the amount of flow m traffic served by the server during $[t_1, t_2]$. All the service processes are assumed to be continuous functions. The departure, arrival, and the queue-length processes are right-continuous functions. The departure process of a flow at a particular server, matches with the service process of that flow at that server at the departure instants. Note that the departure process of a flow at a particular server is also the same as the arrival process of that flow at the next server on the path of the flow. Thus if a flow m goes through servers s_1 and s_2, in that order, and $W_m^{(1)}(t_1, t_2)$ denotes the service process of flow m at s_1 and $A_m^{(2)}(t_1, t_2)$ denote the arrival process of flow m at s_2, for any time interval $[t_1, t_2]$, then

$$W_m^{(1)}(t_1, t_2) - L_{m,max} \leq A_m^{(2)}(t_1, t_2) \leq W_m^{(1)}(t_1, t_2) + L_{m,max} \tag{3}$$

where $L_{m,max}$ is the maximum length of a packet of flow m.

Note, that a packet may complete its transmission at any instant within an envelope-slot (not necessarily at the end of the envelope-slot). We will assume that when a packet has been transmitted completely, it is reassembled and placed in per-flow packet queues at the output instantaneously. Also, though in our analysis we would not assume whether the scheduler $S_{(i,j)}$ interleaves packets between different flows or not, in a practical scenario, schedulers that do not interleave packets (i.e. $S_{(i,j)}$ continues to transmit the same packet until completed) are preferable, since in that case we would need to maintain only one reassembly buffer for each input-output pair, instead of maintaining a separate reassembly buffer for each flow.

Let r_n be the rate guaranteed to flow n. Let $r_{(i,j)}$ be the total guaranteed rate of the packet-traffic between input i and output j, i.e. $r_{(i,j)} = \sum_{n \in F_{(i,j)}} r_n$, where $F_{(i,j)}$ is the set of flows between (i, j). We will assume that the scheduler S serves the envelope-flow of (i, j) at a rate $r_{(i,j)}$. We will also assume that $S_{(i,j)}$ offers a service share of $\frac{r_n}{r_{(i,j)}}$ to flow n. Now let $W_{S_{(i,j)}}(t_1, t_2)$ denote the amount of *packet-traffic* of (i, j) served by $S_{(i,j)}$ during $[t_1, t_2]$, while $W_{(i,j)}(t_1, t_2)$ denote the amount of *envelope-traffic* of (i, j) served by S during $[t_1, t_2]$. Note that all the work done on the envelope-traffic of (i, j) may not be translated into the work done on the packet-traffic between input i and output j; therefore, in general, $W_{S_{(i,j)}}(t_1, t_2) \leq W_{(i,j)}(t_1, t_2)$. However, if $S_{(i,j)}$ is continuously backlogged during $[t_1, t_2]$, then no bandwidth is wasted during the interval, and thus $W_{S_{(i,j)}}(t_1, t_2) = W_{(i,j)}(t_1, t_2)$ for such an interval. Let $W_n(t_1, t_2)$ denote the amount of packet-traffic of flow n served by $S_{(i,j)}$ (and hence by the switch scheduler) during $[t_1, t_2]$.

In deriving the worst-case delay bounds, some of the definitions and analysis techniques used by Cruz in [14] and Bennett & Zhang in [13] have been very useful to us. For completeness, we state two definitions from [13].

Definition 1: A server node s is said to guarantee a Worst-case Fairness Index (WFI) of α_m to flow m, offering a service share of $\frac{r_m}{r_s}$ to the flow, if for any interval $[t_1, t_2]$ for which the flow is continuously backlogged, the following holds:

$$W_m(t_1, t_2) \geq \frac{r_m}{r_s} W_s(t_1, t_2) - \alpha_m \tag{4}$$

Definition 2: A server s is said to guarantee a Service Burstiness Index (SBI) of γ_m to flow m, offering a service share of $\frac{r_m}{r_s}$ to the flow, if for any time t_2 for which the flow is backlogged at s, there exists a time instant t_1 within the server's busy period that includes also t_2, where $t_1 < t_2, Q_m(t_1^-) = 0$ and $Q_m(t_1) \neq 0$, such that the following holds:

$$W_m(t_1, t_2) \geq \frac{r_m}{r_s} W_s(t_1, t_2) - \gamma_m \tag{5}$$

For a constant-rate server serving at rate r_s, $W_s(t_1, t_2) = r_s(t_2 - t_1)$, and so the quantity $\frac{r_m}{r_s} W_s(t_1, t_2)$ reduces to $r_m(t_2 - t_1)$. Note that a flow's guaranteed WFI is also the flow's guaranteed SBI. However, the converse is not necessarily true. Weighted Fair Queueing (WFQ) [15] and Worst-case Weighted Fair Queueing (WF^{2}Q) [16] have the same SBI, which is independent of the number of sessions sharing the link; however, WFI for WFQ depends on the number of sessions sharing the link, but not for WF^{2}Q.

For a packet of flow n, the entire system of Figure 6 can be looked upon as two schedulers - the switch scheduler and the output packet scheduler, connected in tandem. We will analyse the switch scheduler first. We prove the following theorem giving a worst-case bound on the service offered to the packets of a flow n by switch scheduler.

Theorem 1: The two-level hierarchical system representing the switch, where scheduler S guarantees a WFI of $\alpha_{(i,j)}$ to the envelope-flow of (i, j), serving it at a rate $r_{(i,j)}$, and scheduler $S_{(i,j)}$ guarantees a SBI of γ_n to flow n, offering it a service share of $\frac{r_n}{r_{(i,j)}}$, guarantees a Service Burstiness Index (SBI) of

$$\frac{r_n}{r_{(i,j)}} \alpha_{(i,j)} + \gamma_n$$

to flow n, serving it at rate r_n.

Note that the above result is very similar to that for a two-level hierarchical system where all the servers serve in terms of packets [13]. The proof of the above theorem is stated in Appendix A, and is along the same lines as that for the packet-based hierarchical system. Since the SBI offered by a server to a packet-flow is a measure of the worst-case service offered by the server to the flow, it also determines the worst-case delay bound for that flow, as we will see shortly. Note that it is the SBI of the scheduler $S_{(i,j)}$ that appears in the expression above, while for the scheduler S, it is the WFI that appears.

As we have stated before, the output packet scheduler $\hat{S}_j$ is in tandem with the switch scheduler. Let $\hat{S}_j$ guarantee an SBI of $\hat{\gamma_n}$ to the flow n. Now, in order to derive the delay bounds, we will assume that the flow n is leaky-bucket constrained. Let (σ_n, r_n) be the traffic descriptor of flow n. Thus, during any interval $[t_1, t_2]$, the total traffic of flow n arriving at the switch, denoted by $A_n(t_1, t_2)$, is bounded as: $A_n(t_1, t_2) \leq \sigma_n + r_n(t_2 - t_1)$. Now we state a lemma which gives the delay bound on the packets of a flow going through a two servers connected in tandem, in terms of their SBIs, and the leaky-bucket traffic descriptor of the flow.

Lemma 1: Consider a flow m that is leaky-bucket constrained by (σ_m, r_m). Let servers s_1 and s_2 guarantee SBIs of $\gamma_m^{(1)}$ and $\gamma_m^{(2)}$, respectively, to the flow, serving it at a rate r_m. Then, a network of the two servers, connected in tandem, guarantees a delay bound of $\frac{\sigma_m + \gamma_m^{(1)} + \gamma_m^{(2)} + L_{m,max}}{r_m}$ to flow m, where $L_{m,max}$ is the maximum length of a packet of flow m.

The proof of Lemma 1 is in Appendix A. The above result can also be easily extended to a network of k servers connected in tandem (see [14] for a similar result; [13] also states the result for the special case of single server).

Now, using Lemma 1 and Theorem 1 we can bound the total delay seen by packets of flow n in the system shown in Figure 6.

Corollary 1: The total delay of a packet of flow n in the entire switch is upper-bounded by

$$\frac{\sigma_n + \gamma_n + \hat{\gamma}_n + L_{n,max}}{r_n} + \frac{\alpha_{(i,j)}}{r_{(i,j)}}$$

where $\alpha_{(i,j)}$ is the WFI guaranteed to the envelope-flow of (i,j) by the server S, and γ_n and $\hat{\gamma}_n$ are the SBIs guaranteed to flow n by the servers $S_{(i,j)}$ and $\hat{S}_j$, respectively, and $L_{n,max}$ is the maximum length of a packet of flow n.

Typically the term $\alpha_{(i,j)}$ will be dependent on the envelope size L_e, while γ_n and $\hat{\gamma}_n$ will be independent of L_e. Thus, typically the envelope size will influence the overall delay bound only through the term $\alpha_{(i,j)}$.

V. Delay guarantees under various switch scheduling algorithms

In this section, we use the results from the previous section to compute packet delay bounds for several switch scheduling algorithms, which have been shown in the literature to provide bandwidth guarantees to the envelope-traffic. We also give an example choice of schedulers which in combination with our scheme provides delay bounds that are small and independent of the fabric size.

A. TDM scheduling

In [6], the authors propose a simple method, similar to time division multiplexing, for providing bandwidth guarantees. In this method, transmission time is split into *frames* of equal size and an integral number of envelopes can potentially be transmitted during each frame. Bandwidth for an aggregate flow between an input-output pair is guaranteed by reserving adequate number of envelopes for that flow in each frame (the procedure for computing the schedule is in [6]). A new static schedule is calculated every time a new flow arrives or departs from the switch. If an input-output pair has no envelopes to transmit during its assigned slot in the frame, then the idle capacity is given to other ports using an iterative matching algorithm.

Treating the switch as a server, it is easy to show that the WFI guaranteed by the switch to the flows is $2F$ (F is the frame size) [17]. Corollary 1 gives the delay bound on a packet by replacing $\alpha_{(i,j)}$ by $2F$ and γ_n by the appropriate SBI of the packet scheduler $S_{(i,j)}$. Thus the delay of a packet will depend on the frame size. The frame size is determined by the minimum bandwidth allocation over the fabric and the number or fabric ports since every port must be able to receive at least the minimum bandwidth. Thus the delay bound is of the order of $O(N \times L_e)$ where N is the number of ports in the fabric.

Therefore, although the TDM approach provides a mechanism for guaranteeing delay bounds in an input/buffered switch with variable size packets, these bounds can be rather large since they depend both on the number of ports of the scheduler and the envelope size (i.e. for larger, and higher capacity fabrics, the delay bounds will increase). Ideally, we would like to design a scheduling algorithm where the delay bounds will be independent of the fabric size and will not grow substantially with increasing envelope size.

B. *Output queueing emulator*

Chuang et. al. in [10], and Stoica & Zhang in [11] presented switch scheduling algorithms which can mimic output queueing switches using an input-output buffered crossbar with a speed-up of 2. These algorithms apply only to constant size envelope switches, and in addition, they require that corresponding output buffered system that they emulate must be *monotonic* (i.e. future arrivals will not change the order of transmission of packets already in the queues).

Our scheme extends this approach for the case of variable size packets, as shown in Figure 7. In order to work correctly, the switch algorithms in [10], [11] require that at most one (dummy) envelope arrives at each input of the crossbar in a single envelope-time. Note that in the scheme described in Figure 5, multiple envelopes may be placed into the envelope-queues at a particular input in a single envelope time. To see this through an example, let all the envelope-queues and the packet-queues associated with a particular input be initially empty. Now let N small packets arrive at that input in a single envelope-time, each destined for a different output. Then in the scheme described in Figure 5, we would need to insert one envelope in each of the N envelope-queues (and hence a total of N envelopes) at that input, in a single envelope-time. To enforce the criteria that at most one envelope arrives at the queues at each input of the crossbar, we introduce an input scheduler at each input, as shown in Figure 7, such that it serves exactly one envelope each envelope-time. Each input scheduler maintains N envelope-queues, one for each output. The envelope-queue for (i, j) at the corresponding input scheduler is kept backlogged with envelopes whenever the packet-scheduler $S_{(i,j)}$ is backlogged with packets. An envelope for (i, j), on being scheduled by an input scheduler at input i, is placed

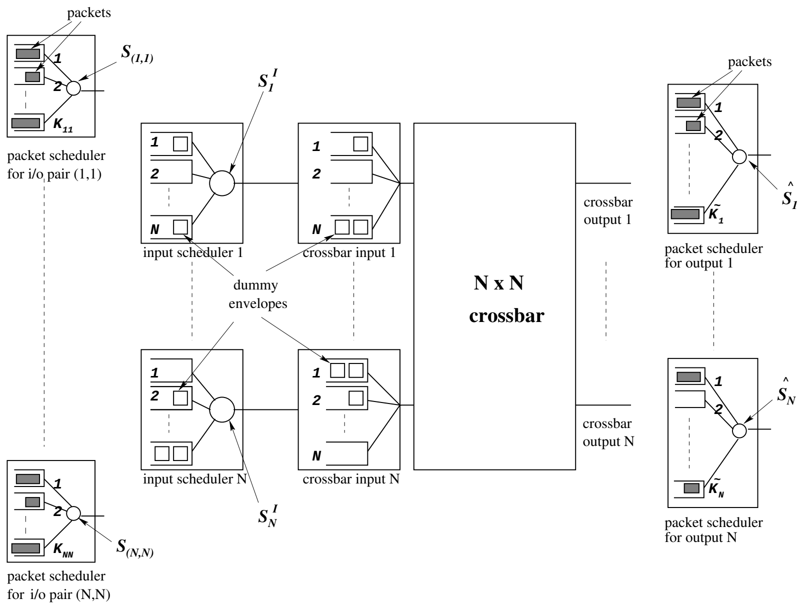

Fig. 7. The overall architecture of the switch for achieving bandwidth and delay guarantees on the packets for the output queueing emulation algorithms

in the queue for (i, j) at the crossbar inputs. When the crossbar allots a slot for the transmission of a envelope of (i, j), then instead of the transferring the envelope, the packet-scheduler $S_{(i,j)}$ is used to transfer packets across the switch in that slot, as we have described in the last section.

The scheduling model of the switch scheduler in this case is slightly different from the one described in Figure 6. The two-level switch scheduler, as seen by a packet of flow n going between input i and output j, is shown in Figure 8. Since the crossbar emulates an output buffered system, to the envelopes of flow (i, j), the crossbar looks just like a single scheduler, which schedules the envelopes of the N flows destined to output j, the scheduling discipline being the same as the one that the crossbar emulates. Thus for a packet going from input i to output j, the upper level of the switch scheduler can be modeled as two envelope-schedulers connected in tandem: (i) the input scheduler, which is denoted as S_i^I, and (ii) the output scheduler, that the crossbar emulates, which is denoted as S_j^O. Note that the envelope-queue for (i, j) at S_i^I remains backlogged whenever $S_{(i,j)}$ is backlogged. Also, $S_{(i,j)}$ schedules packets across the switch only when S_j^O decides to schedule an envelope of (i, j). We will assume that both S_i^I and S_j^O serves the envelope-flow of (i, j) at a rate $r_{(i,j)}$, the aggregate rate of packet-traffic between (i, j).

Let us assume that S_i^I guarantees a WFI of $\alpha_{(i,j)}^I$, and that the emulated scheduler guarantees a WFI of $\alpha_{(i,j)}^O$, to the envelope-flow of (i, j). The worst-case service offered by the switch scheduler to the envelopes of (i, j) is given by the following lemma (proved in Appendix B).

Lemma 2: For any interval $[t_1, t_2]$ for which the envelope-queue for (i, j) is continuously backlogged at the corresponding input scheduler, the envelope-traffic of (i, j) served by the switch, denoted by $W_{(i,j)}(t_1, t_2)$, is bounded as:

$$W_{(i,j)}(t_1, t_2) \geq r_{(i,j)}(t_2 - t_1) - \alpha_{(i,j)}^I - \alpha_{(i,j)}^O - L_e \tag{6}$$

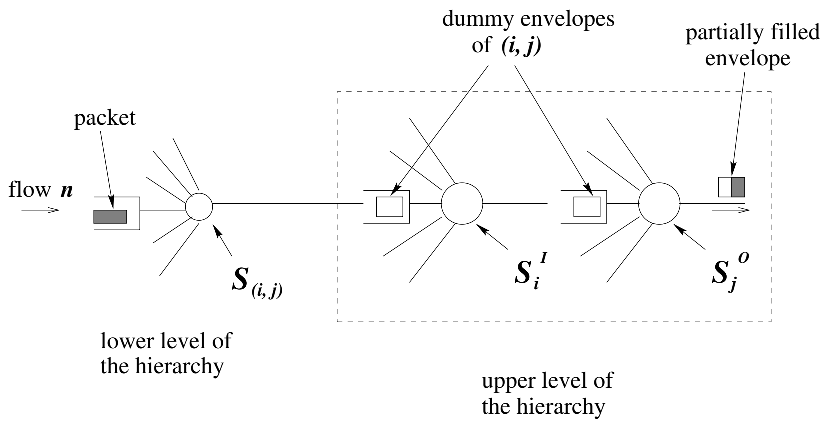

Fig. 8. Scheduling model of the two-level switch scheduler

The above lemma allows us to view the combination of the two schedulers, S_i^I and S_j^O, as a single scheduler with a WFI equal to the sum of the WFIs of the two schedulers, plus an additive term of L_e. Note that during any interval $[t_1, t_2]$ for which $S_{(i,j)}$ is continuously backlogged, S_i^I is backlogged too, and there is no bandwidth loss; thus the above lemma implies that during such an interval, $W_{S_{(i,j)}}(t_1, t_2) = W_{(i,j)}(t_1, t_2) \geq r_{(i,j)}(t_2 - t_1) - \alpha^I_{(i,j)} - \alpha^O_{(i,j)} - L_e$. The expression for the overall delay bound on a packet of flow n then follows directly from Corollary 1, and is obtained by replacing $\alpha_{(i,j)}$ in the expression by $(\alpha^I_{(i,j)} + \alpha^O_{(i,j)} + L_e)$.

C. Achieving Delay Bounds Independent of N

¿From the above discussion, we see that in order to provide a delay bound which is independent of the switch size N, both the input scheduler and emulated scheduler must guarantee WFIs independent of N. WF^{2}Q (or its low complexity version WF^{2}Q+ [16], [18]), as well as Shaped Virtual Clock (Shaped VC) [18], provide WFIs that are independent of the number of flows sharing the scheduler. However, it is easy to see that these algorithms are not *monotonic*, and hence cannot be emulated using the crossbar scheduler described in [10], [11].

However, we will argue that if we implement Shaped VC at the input schedulers, we do not need the emulated algorithm to have a WFI independent of N, and an algorithm that just has an SBI independent of N (like WFQ, or even (unshaped) Virtual Clock) is sufficient. Let $D^O_{(i,j)}$ be an upper-bound on the delay of an envelope of (i, j) from the time it arrives at the crossbar input queue to the time it is scheduled to be transferred across the crossbar. Now consider the following lemma (proved in Appendix B) which gives the WFI of the combination of the schedulers S_i^I and S_j^O in terms of $\alpha^I_{(i,j)}$ and $D^O_{(i,j)}$, instead of $\alpha^I_{(i,j)}$ and $\alpha^O_{(i,j)}$.

Lemma 3: For any interval $[t_1, t_2]$ for which the envelope-queue for (i, j) is continuously backlogged at the corresponding input scheduler, the envelope-traffic of (i, j) served by the switch, denoted by $W_{(i,j)}(t_1, t_2)$, is bounded as:

$$W_{(i,j)}(t_1, t_2) \geq r_{(i,j)}(t_2 - t_1) - \alpha^I_{(i,j)} - r_{(i,j)} D^O_{(i,j)} - L_e \qquad (7)$$

When the input schedulers implement Shaped VC, the burstiness of the envelope-traffic leaving the input schedulers is small, and is upper-bounded by L_e [18]. Thus from Lemma 1 of [13] (the equivalent of Lemma 1 of this paper for the case of a single server), $D^O_{(i,j)} = \frac{L_e+\gamma^O_{(i,j)}}{r_{(i,j)}}$, where $\gamma^O_{(i,j)}$ is the SBI guaranteed to the envelope-flow of (i,j) by S^O_j. Thus $D^O_{(i,j)}$, and hence the overall WFI of the combination of the servers S^I_i and S^O_j, is independent of N if $\gamma^O_{(i,j)}$ is independent of N.

D. Example with Delay Bound Independent of N

Now we will calculate the delay bound for one specific case, where the the schedulers $S_{(i,j)}$ and $\hat{S}_j$ follow WFQ, the schedulers S^I_i follow Shaped VC, and the scheduling algorithm that the crossbar emulates is WFQ. ¿From results in [15], [17], it follows that the SBI (calculated according to Definition 2) guaranteed to the flows by a server following WFQ is L_{max}, the maximum length of a packet being served by the server. Similarly, from results in [13], [18], it follows that the WFI (calculated according to Definition 1) guaranteed to the flows by a server following Shaped VC is also L_{max}. Thus, $\alpha^I_{(i,j)} = L_e$, $\gamma^O_{(i,j)} = L_e$, $\gamma_n = L_{(i,j),max}$, and $\hat{\gamma}_n = L_{j,max}$, where $L_{(i,j),max}$ denotes the maximum packet-length amongst flows between input i and output j, and $L_{j,max}$ denotes the maximum packet-length amongst flows destined for output j. Therefore, $D^O_{(i,j)} = \frac{L_e+\gamma^O_{(i,j)}}{r_{(i,j)}} = \frac{2L_e}{r_{(i,j)}}$. The overall delay bound follows from Corollary 1 and Lemma 3.

Corollary 2: If the packets schedulers at the input and the output follow WFQ, the input schedulers follow Shaped VC, and the crossbar emulates WFQ, then the total delay of a packet of flow n in the system is upper-bounded by

$$\frac{\sigma_n + L_{(i,j),max} + L_{j,max} + L_{n,max}}{r_n} + \frac{4L_e}{r_{(i,j)}}$$

The delay bound is independent of N, and the envelope-size affects the delay bound only through the term $\frac{4L_e}{r_{(i,j)}}$. Note, $\sigma_n \geq L_{n,max}$, where $L_{n,max}$ is the maximum packet size for flow n. Let $L_{n,max} = L_{(i,j),max} = L_{j,max} = L_p$. Let us assume that 100 flows of equal rate are multiplexed between input i and output j. Then, $r_{(i,j)} = 100r_n$, and the term $\frac{4L_e}{r_{(i,j)}}$ becomes comparable to the others in the delay bound only when L_e is equal to $100 \times L_p$ or more. Therefore, we can increase L_e (and hence reduce the scheduling frequency) considerably, compared to the maximum packet size, without affecting the overall delay bound significantly. Also, L_e can be made fairly large, still keeping the delay bound on a packet of the same order of magnitude as $(\frac{\sigma_n}{r_n} + \frac{L_{j,max}}{r})$, the delay bound for a packet of flow n in a completely output-buffered system.

Shaped VC is a non-work-conserving scheduling discipline. However, this should not be problem in a practical scenario. If some of the fabric capacity remains idle, it can be used for best-effort traffic. This “excess switch bandwidth” can also be used to transfer the packets of the guaranteed flows across the switch fabric, without violating the delay guarantees of other guaranteed flows.

VI. Simulation Study

In this section, we present a simulation study of the schemes proposed in the Section V-B. In particular, we study the effect of increasing the envelope size (i.e. decreasing the scheduling frequency) on the average delays. Here we are only interested in average delays since the worst case delay bounds are already proven in the previous section. All the simulations are for the scheme where the crossbar scheduler emulates output queueing with a speedup of 2 in accordance with [10], [11]. In all of the simulation results shown here, the incoming traffic is symmetric across all inputs, and an incoming packet chooses its destination randomly amongst all outputs. Thus the average traffic for all input-output pairs is the same. The inter-packet times are exponentially distributed.

A. Simulation results

Some representative results are shown in Figures 9 & 10. The scheduling algorithm at the input schedulers is Shaped VC. The switch has a speedup of 2 and emulates Starting Potential-based Fair Queueing (SPFQ) [19]. SPFQ is a simpler version of WFQ that retains all the important properties of WFQ, but has much lower complexity of timestamp computations as compared to WFQ. The per-flow packet schedulers both at the input and the output side of the switch follow SPFQ.

Note that the shapes of the delay curves will depend on the packet-size to envelope-size ratio rather than on the absolute packet size. We are interested in the case where the scheduling interval is large (i.e. the scheduler is slowed down considerably) since this is what we need to design fast-switches. This means that the envelope size is large compared to the packet size and so there can be considerable bandwidth loss. In Figure 9, we have plotted the total (over all schedulers) packet delay, D_{tot}, averaged over a large number of packets, against the offered load, for cases when the envelope size is equal to or larger than the mean packet size. The packet size distribution is the same as that for Figure 3, as described in Section I. The delay shown is in units of the transmission time of an average size packet on the input/output links.

Note, the total delay of a packet in the switch, D_{tot}, is the sum of two components – D_{in}, the delay suffered at the per-flow queues at the input side of the switch while waiting to be scheduled across the switch, and D_{out}, the delay at the per-flow queues at the output side while waiting to be scheduled on the output link. To give an idea of how much each of these components contribute to the overall delay, Figure 10 shows D_{tot} and D_{in} against the offered load, when the envelope size is much larger than the mean packet size.

¿From Figures 9 and 10 we can easily make the following observations. (a) The simulations show that we always get high throughput, as expected, even when the envelope size is much larger than the packet size. (b) The total delay is roughly proportional to L_e. This can easily be observed in the flatter portion of the delay curve, particularly for large envelope sizes. (c) Figure 10 shows that the major component in the total delay of a packet is D_{in}. This is again expected, since the bandwidth loss occurs at the input side of the switch, and not on the output side. It is the component D_{in} that is directly affected by the envelope size L_e. Besides we would expect the traffic entering the output buffers to

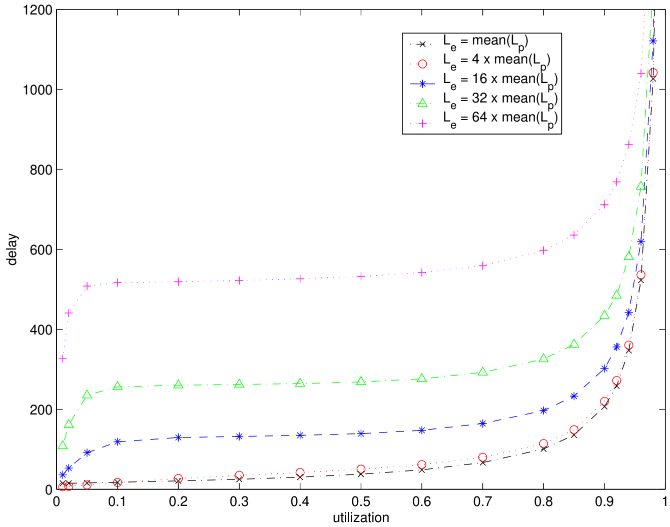

Fig. 9. D_{tot} vs. Utilization for various envelope sizes

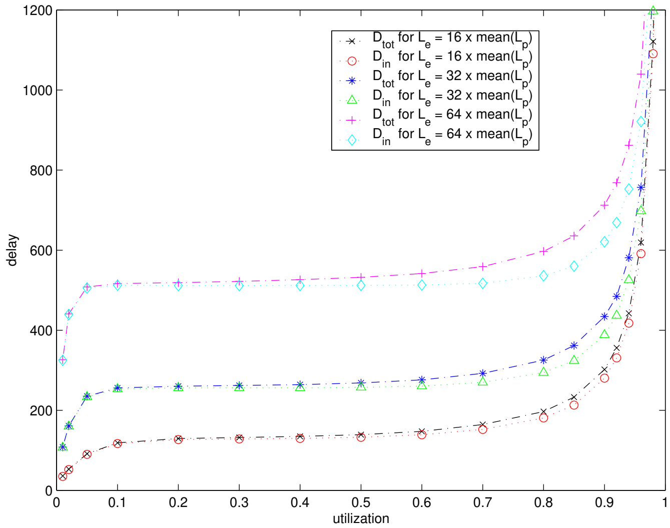

Fig. 10. D_{tot} and D_{in} vs. Utilization for various envelope sizes

be much more “shaped” and hence they will depart soon after their arrival at the output buffers.

A minor problem with the proposed scheme, evident from the previous results, is that the average delay of the packets in the system increases with the envelope size. We argue, however, that this increased delay is not practically significant very high-speed systems. Assume that in the above examples, the link speed is 40Gbps (approximately 80ns to service for a 436 byte packet). ¿From the plots in Figure 9 the average delay can be estimated to be approximately 16 microseconds when the envelope size is selected 16 times larger than the mean packet size. This delay is not much larger than the processing delays that a packet encounters in any high-speed router. Note that this is only a small penalty compared to the significant throughput loss penalty due to fragmentation shown in Figure 3. Furthermore, from an implementation viewpoint, a 10ns scheduling time (when the envelope size is equal to the minimum packet size, which is assumed to be 64 bytes) makes it very difficult to implement efficient scheduling algorithms. However, when the

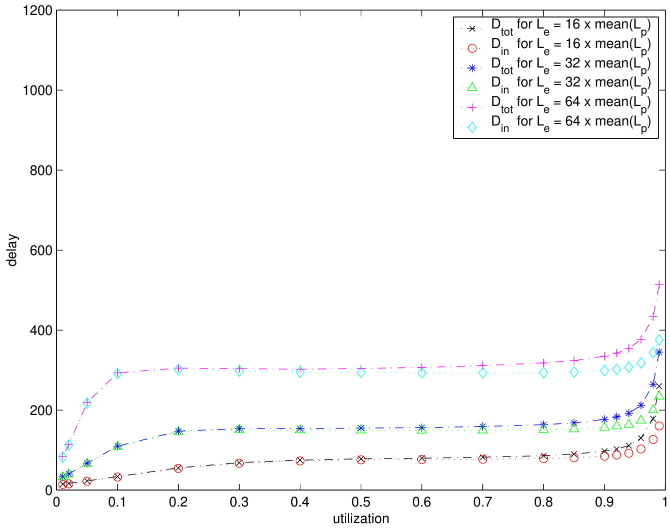

Fig. 11. D_{tot} and D_{in} vs. Utilization for various envelope sizes, with ESBU

envelope size is increased to 16 times the average packet size or approximately 128 times the minimum packet size, the available scheduling time of 1.2 microseconds makes the implementation of the schedulers easier.

Note that a $N \times N$ switch with a speedup of 2 can transfer a maximum of $2N$ envelopes across the fabric in a single envelope-time. However, in the scheme of Figure 7, the N input schedulers can forward only up to N envelopes, in total, in a single envelope-time. Thus, in the average case, we would expect that there could be many idle fabric slots, even though there are envelopes waiting at the input schedulers (and hence packets waiting to be scheduled across the fabric) which could have potentially utilized these slots. At low loads, this problem is worsened by the fact that the input schedulers follow Shaped VC, since Shaped VC is non-work-conserving at low loads. So a good method for utilizing this "excess switch bandwidth" must be incorporated into the scheduler, in order to reduce the average delays further. This is also needed to accommodate best effort traffic.

B. Excess Switch Bandwidth Utilization

To avail of the excess bandwidth of the switch, after each run of the matching algorithm, the crossbar scheduler can use the empty slots to schedule envelopes that are waiting at the envelope-queues at the input schedulers. As before, the slot scheduled for a particular envelope is actually utilized for transferring packets, using the corresponding packet scheduler. The switch bandwidth that we are distributing in this way is the excess bandwidth, and therefore does not affect the worst-case delay guarantees. A discussion about excess bandwidth distribution in fair-queueing schedulers can be found in [20]. Here the authors argue that redistributing of the excess bandwidth in a Longest-Queue-First-like (LQF) way can improve the average delay and tail of delay distribution considerably, without sacrificing the worst-case delay bound and the worst-case fairness properties.

The equivalent of LQF in the case of input buffered switches, would be Maximum Weighted Bipartite Matching on the unmatched inputs and outputs where the weight of the edge between an input and an output node is the total queue-

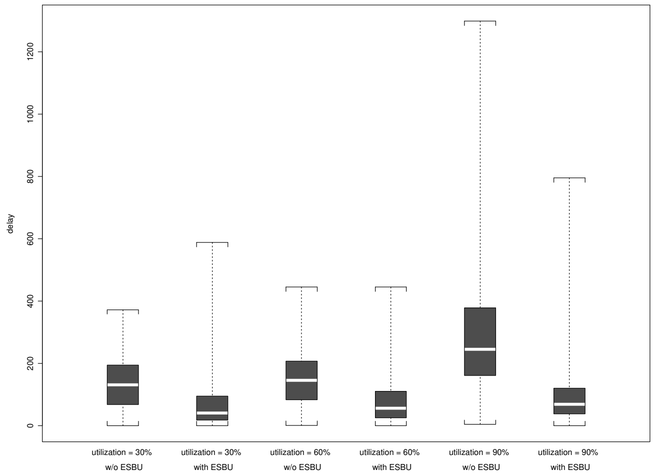

Fig. 12. Boxplots of the delays for with and w/o ESBU, for various envelope sizes, for utilization = 30%, 60%, 90%

length of packets of that input-output pair [7]. Maximum Weighted Bipartite Matching is computationally expensive and so we approximate it with a simple greedy algorithm instead. In this greedy matching algorithm, at each iteration, we greedily pick the input-output pair with the largest backlog, include it in the matching, and remove the input and the output from further consideration. It can be easily seen that this greedy algorithm is just a special case of the stable marriage algorithm (which forms the core of the matching algorithm described in [10], [11]). Figure 11 shows D_{tot} and D_{in} with Excess Switch Bandwidth Utilization ($ESBU$), when the excess switch bandwidth is distributed in the way just described. Comparing Figures 11 and 10, we see that D_{tot} has improved significantly with ESBU. This is mainly because D_{in} has improved considerably with ESBU. However, as we would intuitively expect, D_{out} could be more in this case, as compared to the case without ESBU, and could contribute significantly to the overall delay at very high loads. Figure 12 shows the boxplots of the delay distributions, with and without ESBU. The boxplot is a compact way of comparing distributions. The line in the middle of each box shows the median. The position of the edges of the box correspond to the upper and lower quantiles. The lines outward from the box show the other percentiles. The figure shows that both the means and the tails of the delay distributions have improved considerably with ESBU.

VII. Concluding Remarks

Although optical switches have excellent properties in terms of power consumption and space requirements, they have longer reconfiguration times compared to electronic switches, which limit their application in packet switching that requires fast reconfiguration. For optical technologies to be useful, the scheduling algorithms used must account for a slow reconfiguration frequency. We also note that some of the recently proposed fabric schedulers do not satisfactorily take into account the fact that arriving packets can be of variable sizes and therefore cause bandwidth loss in the fabric. Motivated by these observations, we propose schemes where the scheduling decisions of the crossbar can be reduced considerably, yet maintaining the stability properties of input buffered and input/output buffered switches. We also propose schemes that can provide bandwidth and delay guarantees, which for all practical purposes, are almost similar to that of output queued switches. Slowing down the frequency of scheduling decisions provides two advantages:

1. Allows the use of optical technologies.
2. Provides a comfortable margin for implementing sophisticated arbitration mechanisms.

REFERENCES

[1] M. Zirngibl, *Optical Fiber Communications*, chapter 8. Application of Optical Switch Fabrics, pp. 374–404, Kaminow/Li Academic Press, 2002.

[2] D. Stephens and H. Zhang, "A distributed scheduling algorithm in high speed input queued crossbars," in *Proceedings of IEEE INFOCOM*, April 1998, pp. 282–290.

[3] C. Dragone, C.A. Edwards, and R.C. Kestler, "Integrated optics nxn multiplexer and silicon," *Photonics Technology Letters*, vol. 3, no. 10, pp. 896–899, 1991.

[4] N. McKeown, "iSlip: A scheduling algorithm for input-queued switches," *IEEE/ACM Transactions on Networking*, vol. 7, no. 2, pp. 188–201, April 1999.

[5] N. McKeown, M. Izzard, A. Mekkittikul, B. Ellersick, and M. Horowitz, "The tinytera: A packet switch core," in *Proceedings of Hot Interconnects V*, August 1996.

[6] T. E. Anderson, S. S. Owicki, J. B. Saxe, and C. P. Thacker, "High speed switch scheduling for local area networks," *ACM Transactions on Computer Systems*, vol. 11, no. 4, pp. 319–52, November 1993, Also as Digital Systems Research Center Technical Report, No. 99.

[7] N. McKeown, V Anantharam, and J. Walrand, "Achieving 100% throughput in an input-queued switch," in *Proceedings of INFOCOM'96*, March 1996, pp. 296–302.

[8] Cooperative Association of Internet Data Analysis, "Packet size reports," http://www.caida.org.

[9] J. Dai and B. Prabhakar, "The throughput of data switches with and without speedup," in *Proceedings IEEE INFOCOM*, April 2000.

[10] S.-T Chuang, A. Goel, N. McKeown, and B. Prabhakar, "Matching output queueing with a combined input output queued switch," *IEEE J. Sel. Areas Commun.*, vol. 17, no. 6, pp. 1030–1039, December 1999.

[11] I. Stoica and H. Zhang, "Exact emulation of an output queueing switch with a combined input output queueing switch," in *Proceedings of IWQoS*, June 1998.

[12] A. Charny, P. Krishna, N. Patel, and R. Simcoe, "Algorithms for providing bandwidth and delay guarantees in input bbuffered crossbars with speed-up," in *Proceedings of IWQoS*, 1996.

[13] J.C.R. Bennett and H. Zhang, "Hierarchical packet fair queueing algorithms," in *Proceedings of ACM SIGCOMM'96*, Palo Alto, CA, August 1996, pp. 143–156.

[14] R.L. Cruz, "Service burstiness and dynamic burstiness measures: A framework," *Journal of High Speed Networks*, vol. 1, no. 2, pp. 105–127, 1992.

[15] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control - the single node case," in *Proceedings of INFOCOM '92*, May 1992, vol. 2, pp. 915–924.

[16] J.C.R. Bennett and H. Zhang, "WF^2Q: Worst-case fair weighted fair queueing," in *Proceedings of INFOCOM 96*, March 1996, pp. 120–128.

[17] D. Stiliadis and A. Varma, "Latency-rate servers: A general model for analysis of traffic scheduling algorithms," in *Proceedings of IEEE INFOCOM '96*, http://www.cse.ucsc.edu/research/hsnlab/publications/, April 1996, pp. 111–119.

[18] D. Stiliadis and A. Varma, "A general methodology for designing efficient traffic scheduling and shaping algorithms," in *Proceedings of IEEE INFOCOM'97*, 1997, pp. 326–335.

[19] D. Stiliadis and A. Varma, "Efficient fair queueing algorithms for packet switched networks," *IEEE/ACM Transactions on Networking*, vol. 6, no. 2, pp. 175–185, 1998.

[20] N. Duffield, T.V. Lakshman, and D. Stiliadis, "On adaptive bandwidth sharing with rate guarantees," in *Proceedings of IEEE INFOCOM'98*, April 1998, pp. 1122–1130.

Appendix A

Proof of Theorem 1: Let $Q_n(t)$ denote the queue-length of flow n at server $S_{(i,j)}$ at time t. Now consider any time t_2 for which the packets of flow n are backlogged at $S_{(i,j)}$. From the definition of SBI, there exists a time instant t_1 within $S_{(i,j)}$'s busy period that includes also t_2, where $t_1 < t_2, Q_n(t_1^-) = 0$ and $Q_n(t_1) \neq 0$, such that

$$W_n(t_1, t_2) \geq \frac{r_n}{r_{(i,j)}} W_{S_{(i,j)}}(t_1, t_2) - \gamma_n \tag{8}$$

Note that since $S_{(i,j)}$ is continuously backlogged during $[t_1, t_2]$, the envelope-flow for (i, j) is also backlogged at S during $[t_1, t_2]$. Thus from the definition of WFI,

$$W_{(i,j)}(t_1, t_2) \geq r_{(i,j)}(t_2 - t_1) - \alpha_{(i,j)} \tag{9}$$

Since $S_{(i,j)}$ is continuously backlogged during $[t_1, t_2]$, all the work done on the envelopes of (i, j) is transferred into actual work done by the packet server $S_{(i,j)}$. Thus

$$W_{S_{(i,j)}}(t_1, t_2) = W_{(i,j)}(t_1, t_2) \tag{10}$$

Combining (8), (9) and (10), we get

$$W_n(t_1, t_2) \geq r_n(t_2 - t_1) - \frac{r_n}{r_{(i,j)}} \alpha_{(i,j)} - \gamma_n \tag{11}$$

Comparing (11) with the definition of SBI, the result follows. □

Proof of Lemma 1: Let $W_m^{(k)}(t_1', t_2')$ be the amount of traffic of flow m served by server s_k during $[t_1', t_2']$, $k = 1, 2$. Let $A_m^{(k)}(t_1', t_2')$ be the amount of traffic of flow m arriving at server s_k during $[t_1', t_2']$, $k = 1, 2$. Note that $A_m^{(2)}(t_1', t_2')$ is also the amount of traffic that departs from s_1 during $[t_1', t_2']$. Let $Q_m^{(k)}(t)$ be the queue-length of flow m at s_k at time t, $k = 1, 2$. Consider any packet of flow m. Let a be the time at which the packet arrives at s_1 and d be the time at which it departs from s_2. Let τ be the departure instant of the packet from s_1 (and therefore the arrival instant of the packet at s_2). Therefore, $a < \tau < d$. From the definition of SBI, since s_2 is backlogged at d, there exists a time instant t_1 within the busy period of s_2 that includes also d, where $t_1 < d, Q_m^{(2)}(t_1^-) = 0$ and $Q_m^{(2)}(t_1) \neq 0$, such that the following holds:

$$W_m^{(2)}(t_1, d) \geq r_m(d - t_1) - \gamma_m^{(2)} \tag{12}$$

Note that the above constraints on t_1 imply that t_1 must be the arrival instant of some packet of flow m at s_2. Therefore, t_1 must be the departure instant of that packet at s_1. Thus, s_1 is backlogged at t_1. Therefore, from the definition of SBI, there exists a time instant t_0 within the busy period of s_1 that includes also t_1, where $t_0 < t_1, Q_m^{(1)}(t_0^-) = 0$ and $Q_m^{(1)}(t_0) \neq 0$, such that the following holds:

$$W_m^{(1)}(t_0, t_1) \geq r_m(t_1 - t_0) - \gamma_m^{(1)} \tag{13}$$

It is easy to see that $t_1 \leq \tau$, and hence $t_0 < t_1 \leq \tau$. Since $Q_m^{(2)}(t_1^-) = 0$, and assuming that the packets of an individual flow are served in FIFO order,

$$W_m^{(2)}(t_1, d) = A_m^{(2)}(t_1, \tau) \tag{14}$$

¿From (3),

$$\begin{aligned} A_m^{(2)}(t_1, \tau) &\leq W_m^{(1)}(t_1, \tau) + L_{m,max} \\ &= W_m^{(1)}(t_0, \tau) - W_m^{(1)}(t_0, t_1) + L_{m,max} \end{aligned} \tag{15}$$

Since $Q_m^{(1)}(t_0^-) = 0$, and $A_m^{(1)} \sim (\sigma_m, r_m)$,

$$\begin{aligned} W_m^{(1)}(t_0, \tau) &= A_m^{(1)}(t_0, a) \\ &\leq \sigma_m + r_m(a - t_0) \end{aligned} \tag{16}$$

Combining (14), (15), (16) and (13), we obtain,

$$W_m^{(2)}(t_1, d) \leq \sigma_m + r_m(a - t_0) - r_m(t_1 - t_0) + \gamma_m^{(1)} + L_{m,max} \tag{17}$$

¿From (17) and (12), and rearranging terms, we get,

$$d - a \leq \frac{\sigma_m + \gamma_m^{(1)} + \gamma_m^{(2)} + L_{m,max}}{r_m} \tag{18}$$

□

Appendix B

Proof of Lemma 2: Let $W_{(i,j)}^I(t_1^{'}, t_2^{'})$, and $W_{(i,j)}^O(t_1^{'}, t_2^{'})$ denote the amount of envelope-traffic of (i, j) served S_i^I and S_j^O, respectively, during any time interval $[t_1^{'}, t_2^{'}]$. Note that $W_{(i,j)}(t_1^{'}, t_2^{'}) = W_{(i,j)}^O(t_1^{'}, t_2^{'})$. Let $A_{(i,j)}^O(t_1^{'}, t_2^{'})$ be the amount of envelope-traffic of (i, j) arriving at S_j^O during $[t_1^{'}, t_2^{'}]$. Note that $A_{(i,j)}^O(t_1^{'}, t_2^{'})$ is also the amount of traffic that departs from S_i^I during $[t_1^{'}, t_2^{'}]$. Now, consider the three different cases:

Case 1: The envelope-queue for (i, j) at the crossbar input is not backlogged at t_2:

In this case, $W_{(i,j)}(t_1, t_2)$ must at least be equal to $A_{(i,j)}^O(t_1, t_2)$. Using this fact, and (3),

$$\begin{aligned} W_{(i,j)}(t_1, t_2) &\geq A_{(i,j)}^O(t_1, t_2) \\ &\geq W_{(i,j)}^I(t_1, t_2) - L_e \end{aligned} \tag{19}$$

Since S_i^I is continuously backlogged during $[t_1, t_2]$, from the definition of WFI,

$$W_{(i,j)}^I(t_1, t_2) \geq r_{(i,j)}(t_2 - t_1) - \alpha_{(i,j)}^I \tag{20}$$

Combining (19) and (20),

$$W_{(i,j)}(t_1, t_2) \geq r_{(i,j)}(t_2 - t_1) - \alpha_{(i,j)}^I - L_e \tag{21}$$

Case 2: The envelope-queue for (i,j) at the crossbar input is continuously backlogged during $[t_1, t_2]$:
In this case, from the definition of WFI,

$$W_{(i,j)}(t_1, t_2) \geq r_{(i,j)}(t_2 - t_1) - \alpha^O_{(i,j)} \tag{22}$$

Case 3: The envelope-queue for (i,j) at the crossbar input is backlogged at t_2, and $\tau > t_1$ is the beginning of that backlogged period:
Note that the envelope-queue for (i,j) at the crossbar input is empty at τ^-. Thus, if $\tilde{A}^O_{(i,j)}(t_1, \tau)$ denotes the amount of envelope-traffic of (i,j) arriving at S^O_j during $[t_1, t_2)$, then

$$\begin{aligned} W_{(i,j)}(t_1, t_2) &= \tilde{A}^O_{(i,j)}(t_1, \tau) + W_{(i,j)}(\tau, t_2) \\ &\leq A^O_{(i,j)}(t_1, \tau) + W_{(i,j)}(\tau, t_2) \end{aligned} \tag{23}$$

¿From (3), the definition of WFI, and the fact that the envelope-queue for (i,j) at the input scheduler is continuously backlogged during $[t_1, \tau]$, it follows that,

$$\begin{aligned} A^O_{(i,j)}(t_1, \tau) &\geq W^I_{(i,j)}(t_1, \tau) - L_e \\ &\geq r_{(i,j)}(\tau - t_1) - \alpha^I_{(i,j)} - L_e \end{aligned} \tag{24}$$

Since the envelope-queue for (i,j) at the crossbar input is continuously backlogged during $[\tau, t_2]$,

$$W_{(i,j)}(\tau, t_2) \geq r_{(i,j)}(t_2 - \tau) - \alpha^O_{(i,j)} \tag{25}$$

Combining (23), (24) and (25), we get

$$W_{(i,j)}(t_1, t_2) \geq r_{(i,j)}(t_2 - t_1) - \alpha^I_{(i,j)} - \alpha^O_{(i,j)} - L_e \tag{26}$$

The result of Lemma 2 follows from the results of the three cases, as stated in (21), (22) and (26). □

Proof of Lemma 3: Let $W^I_{(i,j)}(t_1', t_2')$, and $W^O_{(i,j)}(t_1', t_2')$ denote the amount of envelope-traffic of (i,j) served S^I_i and S^O_j, respectively, during any time interval $[t_1', t_2']$. Note that $W_{(i,j)}(t_1', t_2') = W^O_{(i,j)}(t_1', t_2')$. Let $A^O_{(i,j)}(t_1', t_2')$ be the amount of envelope-traffic of (i,j) arriving at S^O_j during $[t_1', t_2']$. Note that $A^O_{(i,j)}(t_1', t_2')$ is also the amount of traffic that departs from S^I_i during $[t_1', t_2']$. Since the maximum delay of a envelope of (i,j) at S^O_j is $D^O_{(i,j)}$, the total amount of envelope-traffic of (i,j) served by S^O_j during the time interval $[t_1, t_2]$ is at least equal to the amount of envelope-traffic of (i,j) arriving at S^O_j during $[t_1, t_2 - D^O_{(i,j)}]$. Using this fact, (3), and the definition of WFI,

$$\begin{aligned} W_{(i,j)}(t_1, t_2) &\geq A^O_{(i,j)}(t_1, t_2 - D^O_{(i,j)}) \\ &\geq W^I_{(i,j)}(t_1, t_2 - D^O_{(i,j)}) - L_e \\ &\geq r_{(i,j)}(t_2 - D^O_{(i,j)} - t_1) - \alpha^I_{(i,j)} - L_e \\ &= r_{(i,j)}(t_2 - t_1) - \alpha^I_{(i,j)} - r_{(i,j)} D^O_{(i,j)} - L_e \end{aligned}$$

□