HAL
open science

A survey on core switch designs and algorithms

Dinil Mon Divakaran, Sébastien Soudan, Pascale Primet, Eitan Altman

To cite this version:

Dinil Mon Divakaran, Sébastien Soudan, Pascale Primet, Eitan Altman. A survey on core switch designs and algorithms. [Research Report] RR-6942, INRIA. 2009. inria-00388943

HAL Id: inria-00388943

https://inria.hal.science/inria-00388943

Submitted on 28 May 2009

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

A survey on core switch designs and algorithms

Dinil Mon Divakaran, Sebastien Soudan, Pascale Vicat-Blanc Primet, Eitan Altman

N° 6942

May 2009

Thème COM

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--6942--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
GRENOBLE - RHÔNE-ALPES

A survey on core switch designs and algorithms

Dinil Mon Divakaran, Sebastien Soudan, Pascale Vicat-Blanc Primet, Eitan Altman

Thème COM — Systèmes communicants
Équipe-Projet RESO

Rapport de recherche n° 6942 — May 2009 — 26 pages

Abstract: Tremendous amounts of effort have gone into research on switch designs and architectures. This survey attempts to sketch the evolution of the modern switch architectures. The survey covers the literature over the period 1987-2008 on switch architectures. Starting with the simple crossbar switch, we explore various architectures such as Output queueing, Input queueing, Combined Input/Output queueing, buffered crosspoint etc., that have evolved during this period. We discuss the pros and cons of these designs, so as to shed light on the path of evolution of switch architecture.

Key-words: switch, architecture, crossbar

Centre de recherche INRIA Grenoble – Rhône-Alpes
655, avenue de l'Europe, 38334 Montbonnot Saint Ismier
Téléphone : +33 4 76 61 52 00 — Télécopie +33 4 76 61 52 52

Etude des algorithmes et des architectures utilisés dans les switchs de coeur de réseau

Résumé : La recherche sur la conception et l'architecture de commutateurs a été l'objet de beaucoup d'effort. Cet état de l'art tente d'esquisser l'évolution de l'architectures des commutateurs modernes. Ce document couvre la période 1987-2008. En partant d'un simple commutateur crossbar, nous explorons différentes évolutions de cette architecture telle que OQ, IQ, CIOQ, buffered crosspoint, etc en soulignant les avantages et inconvénients.

Mots-clés : switch, architecture, crossbar

1 Introduction

The success of the Internet can be attributed to packet switching as it enables any device to communicate with virtually any number of peers at the same time.

While the end-hosts in a packet-switched network put messages into multiple packets and send them through their interfaces, the network multiplexes packets coming from different end-hosts at the interconnects formed using *switches* and *routers*. The switches and routers basically store, route and forward these packets before they reach the destination. One core functionality of such switches (a layer 2 switch, or a layer 3 IP router) is to transfer the packets from the input port to one of the output ports. This functionality, called *switching*, though appears simple, is such a challenging problem to solve at line rates that, there is a wealth of literature on this topic. In this survey, we explore the various switch designs and algorithms, and shed light on the way they have evolved. Unless mentioned otherwise, our discussion revolves around homogeneous switches, that is, switches with same rates (capacities) for all input/output lines.

As we go over different switch architectures, we will see that the evolution is driven by two factors: (1) the performance demands from a network operator's point of view, and (2) IC and VLSI technology. An operator expects high utilization of a switch that is deployed, besides looking to provide QoS guarantees to the customers. Though it has been initially impractical to provide both high throughput and delay guarantees, with the advancement of technology, architectures meeting both requirements are now feasible.

In ancient switches, the input output ports communicated using a single shared bus. Consequently this bus was a limitation, as not more than one pair of ports can communicate at a time. The classical crossbar switch overcame the bottleneck imposed by this shared bus architecture that restricted the use of N input-output port pairs in parallel. The crossbar switch is an NxN matrix of $2N$ buses, connecting input output ports as shown in Fig. 1. Each of the N^2 *crosspoints*, where the bus lines intersect, needs a control line from the scheduler to turn it on or off, thereby allowing the corresponding input-output port communication. Evidently, one input can communicate to only one output at any given time. For example, if an input line card wants to communicate to an output line card, then the scheduler, based on some criteria (which we will discuss later), has to *turn on* the crosspoint formed by the corresponding input and output lines [1]. If all the input ports have packets to unique output ports, then all those packets can be switched in a single time slot. Observe that this is an ideal condition, where the traffic pattern is of uniform nature, and hence N input ports can communicate to N output port simultaneously.

In scenarios that deviate from the ideal traffic pattern, it is inevitable to buffer some packets, as not all packets can be switched and transmitted as soon as they arrive. This reflects the real-world scenario where traffic pattern is essentially non-uniform in nature leading to queueing of packets. Depending on where the buffers are, there are different kinds of switch architectures, namely, *input queued (IQ), output queued (OQ), combined input output queued (CIOQ), crosspoint queued*, etc. Each architectures has its own advantages and drawbacks that will be discussed here.

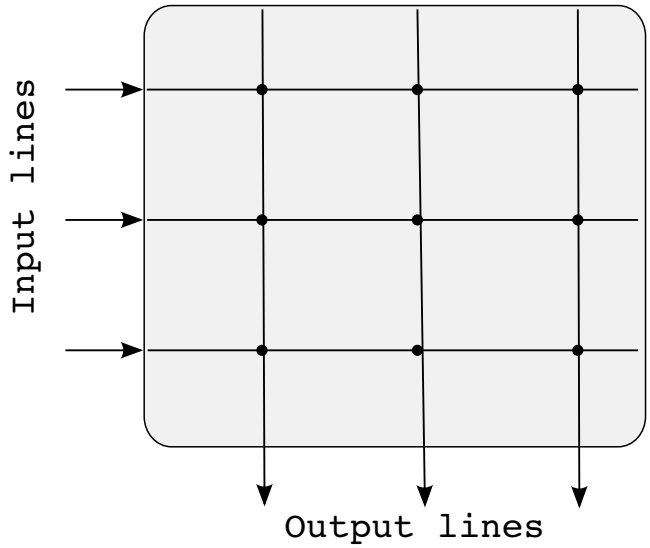


Figure 1: A crossbar architecture

1.1 Organisation

We start by presenting some basic features and functions of a switch in Section 2. Over there, we describe a long existing trend in switch design: *synchronous switching*. Section 3 takes a tour on various architecture that deploys synchronous switching. It includes the well-known Virtual Output Queueing archticture. We also discuss the commonly used arbitration algorithms such as iSlip, as well as some recent randomized algorithms. In section 4, we look into *asynchronous switching* architecture, which has gained attention in the recent times, thanks to the progress made in technology. Finally, we discuss about the current trends that would influence future switch designs in Section 5.

2 Preliminary background

This section describes how the fixed size of switching units and synchronized arbitration are correlated. After briefing on traffic characteristics, we give some background information on the arbitration itself.

2.1 Switching units

In order to explain some of the basic details on switching, let us, for the time being, assume that every input has a packet to be switched to a unique output port; meaning there is no contention for the output. The optimal use of switch matrix happens when all the bus lines (horizontal and vertical) are used at the same time. Therefore, the best one can do is to switch all the packets at the input to the corresponding output, so that none of the bus lines goes under utilized. If the switch fabric is able to switch packets from all inputs to all outputs at a rate not slower than the transfer rate of packets at the output,

then the output port will have packets to send as soon as its transfer is over. This ensures 100% utilization of the output link.

But since packets are of varying sizes, switching time for packets of different sizes will be different; and hence the switch fabric will experience bandwidth wastage. To avoid this, traditional packet switches, switch packets in fixed size smaller blocks called *cells*. The cell size usually used in commercial switches is the size of a minimum-sized packet (64 bytes). The time to transfer a cell is known as *cell slot*. As said before, to achieve 100% output utilization, the time to switch a cell should not be greater than the time to transfer it at the output port. Therefore, the cell size along with the output line rate defines a cell slot. For example, a switch with 40 Gbps line cards using a cell size of 64 bytes has to switch cells within $(64 \times 8)/(40 \times 10^9) = 1.28 \times 10^{-8}$s. Note that, the switch fabric has to do two distinct functions during a cell slot: (i) It has to decide which inputs will communicate to which outputs (arbitration), and (ii) it has to transfer cells between these matched input-output ports. We refer to crossbars that makes such synchronous transfers as *synchronous switches*. We study different kinds of switch designs that fall in the category in Section 3. The *asynchronous switch architecture*, where packets are switched between ports asynchronously, attempts to provide better QoS performance compared to synchronous switches. We study them in Section 4.

2.2 Traffic characterization

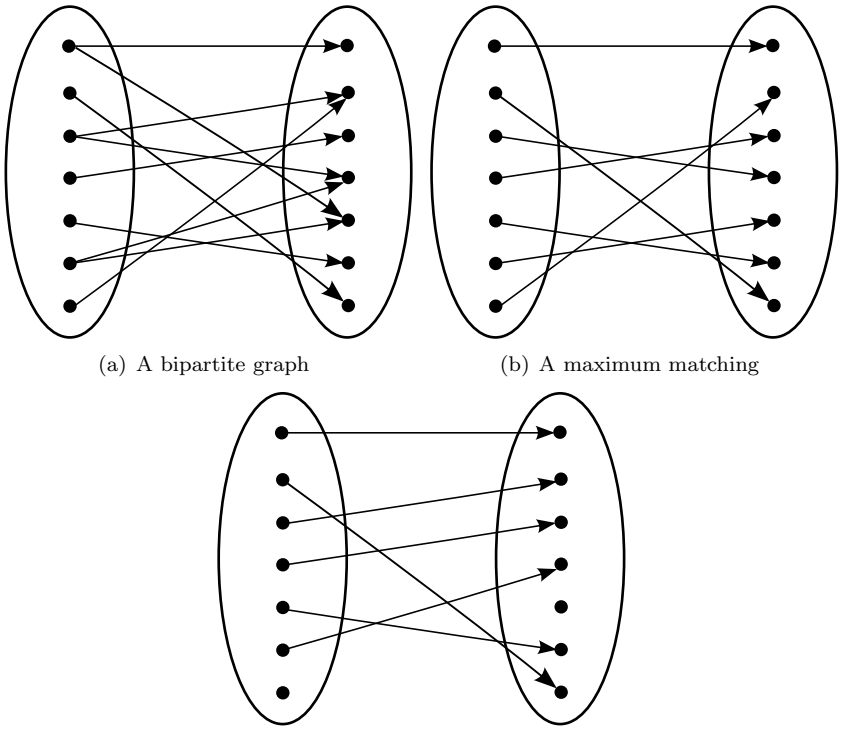
Using the above definitions, we now refer to traffic characterization. If λ_{ij} represents the average cell arrival rate at input i to output j, then the incoming traffic is called admissible if $\sum_{i=1}^{N} \lambda_{ij} < 1$ and $\sum_{j=1}^{N} \lambda_{ij} < 1$; i.e., if none of the input or output ports is over-subscribed. Traffic is uniform if all arrival processes have the same arrival rate, and destinations are uniformly distributed over all outputs. Otherwise, the traffic is non-uniform. An important performance metric, *throughput*, of a switch, is the average rate at which bytes leave the switch per port, expressed as percentage of the line capacity. Thus, a 100 % throughput means the output ports are fully utilized. But this depends not only on the switch internals, but also on the arrival pattern. Hence, designers of scheduling algorithms not only try to achieve 100% throughput, but also *stability*. For an admissible traffic, a scheduling algorithm is stable if the expected queue length is finite. We use this notion of throughput in this paper.

2.3 Contention resolution

Since the crosspoints couple the input and output ports, the key issue in such crossbar switches is to resolve the contention arising at the inputs for output. As the vertical bus is shared by all the input ports, only one input can access it at a time. If there are multiple transfer requests to a single output port, then the switch fabric arbitrator has the responsibility to select one of the contending input ports for sending packet in the next time slot. In fact, the arbitrator has to match different input ports to different output ports for transfer in the following time slot. This translates to a *matching* problem in a bipartite graph, where input ports and output ports form two sets of disjoint nodes, and the requests form the edges. In Fig. 2 is seen a bipartite graph (showing requests from input ports), and two of the possible matchings. Finding a matching given

input requests, is precisely what an arbitration algorithm does. To attain high performance, an arbitration algorithm tries to find a matching which maximizes the number of input-output port pairs that communicate in parallel.

Note that, in the literature, the word *scheduling* is also used to refer to arbitration. The other interpretation of scheduling comes when we talk about deciding which packet should be selected to be sent out through the output link. We stick to the term *arbitration* for the input-output matching, unless explicitly specified. In the following, we present some definitions on different matchings from graph theory.



(a) A bipartite graph (b) A maximum matching

(c) A maximal matching

Figure 2: An example of different matchings

Maximum Weight Matching The edges connecting the nodes of a bipartite graph can have weights associated with them. In switch arbitration, these weights can correspond to queue lengths, for example. In this case, it might be interesting to find a matching that maximizes the sum of weights of the matched edges covering all the (connected) vertices. This is called Maximum Weight Matching (MWM). The complexity of solving MWM is $O(N^3)$. A maximum matching is one that matches the maximum number of inputs and outputs; i.e, it is special case of MWM with unit weights. Such algorithms that find the match containing the maximum number of edges are also called *Maximum Size Matching algorithms*. The complexity is $O(N^{2.5})$ for maximum size matching [2]. Fig. 2(b) is a maximum matching for the graph in Fig. 2(a).

Maximal matching A maximal matching is a matching where no more matches can be made without modifying the existing matches. PIM (Parallel iterative matching) and iSLIP algorithms described below, belong to this category. For example, Fig. 2(c) is a maximal matching corresponding to the graph of Fig. 2(a). To improve the number of matches, it is necessary to modify the current matches. Note that a maximal matching need not be maximum; but, a maximum matching is always maximal.

Having shed lights on the necessary terminologies, we now move on to various switch architectures.

3 Synchronous switching

In this section, we describe different architectural designs under synchronous switching. Recollect that, the important feature of synchronous switching architectures is, an arbitration decision is taken once every time interval, and cells of fixed size are switched between input-output pairs, all at the same time.

3.1 Output Queueing

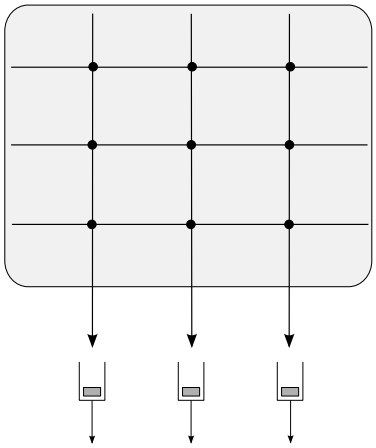


Figure 3: Output Queueing architecture

A distinct and important architecture is the *output queued* (OQ) architecture. It is distinct in the sense, it is practically impossible to realize an OQ architecture for reasonable size switches (as we will see later). But still, since it is considered as an ideal switch for comparing performances, it remains important. The idea is to have a queue at each output port (as shown in Fig. 3), and the packets arriving at the input ports have to be sent without any delay to the output queue. As there is no blocking at the input, packets are queued at the output. Therefore, an OQ architecture can always achieve 100% throughput for all kinds of traffic. For the same reason, different scheduling schemes can be deployed at the output to meet the required QoS guarantees. *In fact, most of the QoS guaranteeing scheduling algorithms proposed have been designed to be used at OQ switches* [3], [4], [5], [6].

But the disadvantage of output queueing is that, to match the rate of the line cards, the switching fabric should run at N times faster than the link rate. That is, in the case of all N links having data to be sent to the same output in a given cell slot, the fabric should run N times faster than the input links to perform this parallel transfers. More importantly, this also means that the buffer at the output should be $(N + 1)$ times faster than the link rate. This requirement is called the internal speedup of the switch [7], and poses a challenge in the implementation of pure output queueing. Besides, link speeds are increasing much faster than memory speeds. Therefore, pure output queueing is too expensive and infeasible to scale to the future demands of bandwidth.

Nevertheless, this architecture is used as a comparison model for its desirable characteristics like high throughput and low latency. For the same reason, several new switch architectures proposed in the literature attempt to *emulate* an OQ switch. By emulation, we mean, under identical input traffic, identical packets leave both (proposed and OQ) switches at the same time.

A typical scenario showing the speedup requirement in OQ switches, is a client/server architecture, where a server is connected to single switch port, and packets from various clients arrive at multiple (possibly all) ports of the switch destined to the server. This is also an example of non-uniform traffic.

Shared memory was proposed for OQ architectures. In a shared-memory switch, memory is shared by all input and output lines. The packets arriving at the input ports are written to the shared memory, which in due time, are read out and sent to the output lines. If R represents line rate, the characteristics of shared memory switches that make it difficult to scale to their capacity can be summed up as [8]:

- As line rates (R) increase, the memory bandwidth of the switch should also increase linearly (2NR),
- Memory has to be accessed twice in every cell slot. For a cell size of 64B, with $N = 32$ and $R = 10$ Gbps, the access time is just 800 ps,
- Memory size requirement increases with line rate.

Hence, shared memory architecture is not attractive for high-speed large size (in number of line cards) switches. For practical purposes, the simplicity in architectures that have queues at the input, has attracted the research community and industry alike. In the following, we look into such switch architectures that are more practical.

3.2 Input Queueing

A simple scheduler used in the early crossbar switches is the *Take-a-ticket* scheduler, implemented in DEC's Gigaswitch [9]. Each input line card maintains a queue where the incoming packets are stored. In the literature, this is often referred to as the *input queueing* (IQ) architecture. The architecture is shown in Fig. 4. If an input line card wants to send a packet to an output line card, it first makes a *request* over a separate control bus to the output; whereupon the output sends a ticket to the input over the control bus. The input line then monitors the control bus. The output sends the current ticket number it is serving, as soon as it has finished processing a packet. When the input

notices that its number is being served, it places its packet on the input data bus for the specific output line. At the same time, the output ensures that the corresponding crosspoint is turned on. This mechanism avoids contention at the output port.

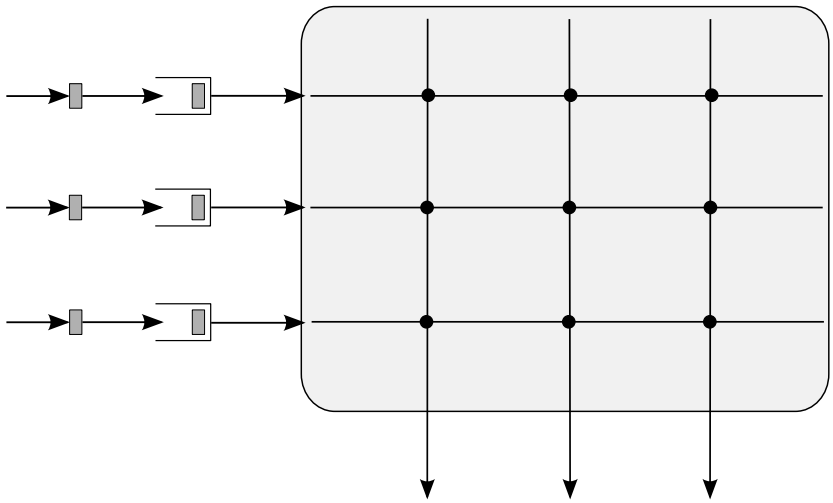


Figure 4: Input Queueing architecture

For each incoming packet in a time slot, an input buffer has to make two operations per time slot: (i) write the incoming packet, and (ii), copy a buffered packet on to the switch fabrics. IQ architecture is thus attractive in high speed networks, as the buffers that queue the incoming packets need run only at twice as fast as the line rates.

IQ architecture as such is uninteresting due a major disadvantage it holds: the *head-of-line* (HOL) blocking. A packet at the head of the queue, destined to a busy output port, blocks all the packets behind it in the queue (even if they are destined to other free output ports). This reduces the throughput to 58.6% for Bernoulli packet arrivals with uniformly randomly selected output port [10].

3.2.1 Virtual output queueing

On the other hand, IQ architecture is attractive in high speed networks, as the buffers that queue the incoming packets need run only at twice as fast as the link rates. Researchers proposed new methods to combat the HOL blocking without resorting to output queueing [11, 12, 13, 14]. The essence is to decompose the single input queue in the IQ architecture into multiple queues at the input. More precisely, there will be N queues at each input port, one for each output. These are called virtual output queues (VOQs) Fig. 5 is an example of a 3×3 VOQ switch. With the VOQs, a cell at the head of a queue can no more block any cell destined to another output.

3.2.2 Parallel iterative matching

Parallel iterative matching, or PIM [15], was one among those that came with the idea of using VOQ to achieve high throughput. It consists of three phases. In the *request* phase, the scheduling requests for an input port is communicated using a bitmap of N bits. A 1 in the i-th position of the bitmap implies that there is a cell destined to i-th output port. The contention at the output port (arising when multiple input ports requests to the same output port) is solved by randomization; this forms the *grant* phase, where one of the inputs is given grant to send data. Similarly, if an input port receives grants from more than one output port, it randomly accepts one and signals the same - and this forms the final phase, *accept* phase. To improve the throughput, the algorithm iterates converging to a maximal match, on an average, in $O(logN)$ iterations. Note that, even if there is no centralized scheduler, the switching is still synchronized due to the iterations; in practice, after a fixed number of iterations, the matching is used to switch cells in the next time slot. *As PIM can not totally avoid contention, it can not achieve 100% throughput under admissible uniform traffic.* There is also a cost involved in generating random numbers at a very good pace.

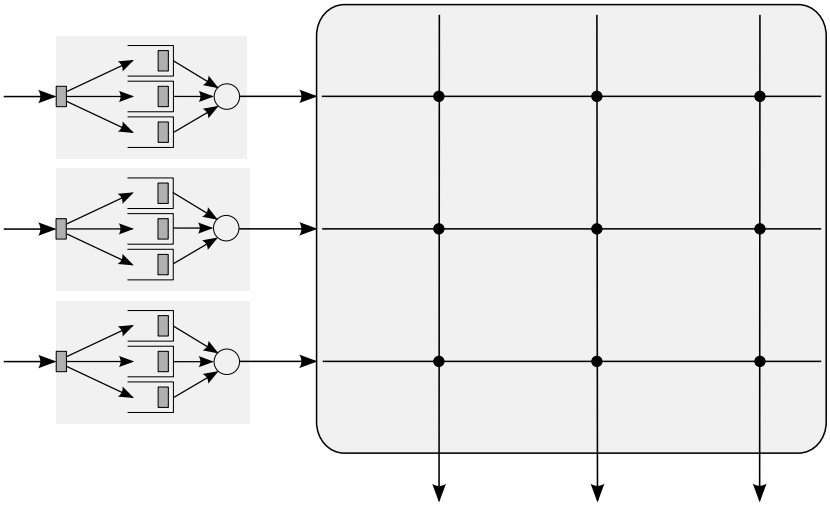


Figure 5: VOQ architecture

3.2.3 Achieving 100% throughput using maximum weight matching

Two Maximum Weight Matching (MWM) algorithms were proposed in [16]. In an MWM algorithm each input-output pair {i,j} (which is treated as an edge) is assigned a weight that is a measure of congestion. The algorithm selects a set of input-output pairs with the highest sum weight. The weights determine various strategies. The *longest queue first* (LFQ) strategy uses the number of cells in each VOQ as weight. Similarly, the wait time of the HOL cells is taken as weights in *oldest cell first* algorithm (OCF) [17]. It has been proved that the MWM algorithms achieve 100% throughput under Bernoulli i.i.d packet arrival process, be it uniform or non-uniform [18], [16]. Later, the results were

also extended for more general arrival processes and admissible traffic in [19]. Additionally the algorithms also provide low delay. However, it is challenging to compute the maximum weight matchings at line speeds. *The high complexity involved makes MWM scheduling algorithms prohibitively expensive for practical implementations in high speed switches.*

3.2.4 The iSLIP scheduling algorithm

As the implementation of MWM algorithms posed significant challenges, researchers proposed many practical algorithms such as MUCS [20], iSLIP [21], and RPA [22]. Here, we will look into the iSLIP algorithm proposed by McKeown for input-queued switches [21].

iSLIP diverts from PIM when it comes to resolving contentions. When PIM uses randomness to select one of the contending ports, iSLIP uses round-robin to choose on port among those contending. This permits simpler hardware implementations compared to PIM, besides making iSLIP faster. Basically, each input and output ports maintains *accept* and *grant* pointers to ports. When there are multiple input requests at an output port, the output chooses the next input port following the one pointed by the *grant* pointer. The same logic is used by the input port to accept one of the grants from multiple output ports. iSLIP achieves close to maximal matches after just one or two iterations. The round robin policy ensures fairness among contenders (a variant of iSLIP, called FIRM [23] was later proposed to improve fairness of iSLIP by approximating FCFS in a better way). Though iSLIP achieves 100% throughput under uniform traffic, it is unable to sustain the same throughput under non-uniform traffic, like MUCS and RPA.

3.2.5 Randomized algorithms for scheduling

Despite being unstable, iSLIP is commonly implemented in commercial switches due to its simplicity. Research works continue to explore the possibility of new scheduling algorithms that are both simple and stable. One way is to use randomized algorithms to perform approximate MWM [24, 25, 26, 27, 28, 29]. Most of these approaches are based on the following three observations [24]:

- The queue lengths do not change much between two consecutive time slots
- Given a matching at time t, a randomly generated matching can be used to improve it, and obtain a matching for $t + 1$
- For randomization, either a matching can be generated at random, or edges at random. Intuitively, keeping (remembering) a few good edges of the previous matching(s), and randomly generating a matching for the remaining edges would turn out to be more useful.

For example, L. Tassiulas exploited the first two observations and came up with the following randomized algorithm [30]: Generate a random matching Q uniformly from $N!$ possible matchings at time $t+1$. If S is the matching used at time t, compare weights of Q with that of S, and use the more heavily weighted matching for scheduling cells at time $t + 1$. Though this algorithm achieves up to 100% throughput, the delay experienced by packets tend to be longer.

Shah *et al.* improved this randomized algorithm by exploiting all the three observations given above [24]. The proposed algorithm provides good delay performance, apart from achieving throughput of up to 100% (this was shown using simulations using i.i.d Bernoulli arrival processes). The skeleton of their approach is based on the following: Instead of generating a random matching at time $t+1$, the algorithm stores a set of edges in the matching S that carries at least a fraction, say η, of the total weight of S. The matching Q uses these edges, and connects the remaining input/outing nodes using a randomly chosen matching. Finally, the heavier (in weights) of Q and S are chosen for time slot $t+1$. An enhanced version of this algorithm, called *Serena*, has a complexity of $O(N)$ [25]. In this version, an 'arrival graph' is formed for time t using edges e_{ij}, such that, e_{ij} is present if there was an arrival at input i to output j at time t. From this arrival graph, for each output, the heaviest weighted edge is retained, and the other edges are generated randomly to obtain the matching Q. The matching for the time slot $t+1$ is obtained by merging Q and the S (matching of the previous time slot) in such a way that the resultant matching has weight equal or greater than the maximum weights of S and Q.

Though the randomized algorithms are simple and achieve high throughput, theoretical bounds on the delay have not been shown yet.

3.2.6 Pros and cons

As seen in the above algorithms, one of the challenges in the VOQ architecture, is to have a scheduler that resolves the input-output match, so that, at each time slot, each input sends one cell to at most one output, and each output receives one cell from at most one input (in other words, IQ-VOQ architecture requires switch matrix scheduling algorithms to achieve high throughput). Overcoming these two constraints at the same time, leads to complex scheduling algorithms. This is equivalent to choosing a permutation of an NxN matrix with at most one 1 in each row, and one 1 in each column; and there are $N!$ ways of choosing one such matrix. To stress on the time constraint, consider a switch with 40 Gbps port. If the cells are of 64 bytes, the time to transmit a cell, or the cell slot is 12.8 ns. Assuming arbitration may take up half of a cell slot (cell transmission and memory write are also done during this time slot, though they might be pipelined), a matching has to be found within 6.4 ns. Therefore, not only the throughput, but also the delay, largely depend on the scheduling algorithm. *The arbitration scheme, that determines which packets from which input queues should be sent to the output, also governs the wait time of the packets in the queue.*

Another major criticism facing IQ (IQ-VOQ) architecture is that its performance do not match that of the OQ architecture. This is where it lags behind the OQ architecture. In the OQ architecture, it is much more convenient to provide QoS guarantees for architectures with queueing at the output. An architecture that take advantages of both input queueing and output queueing can be a better option.

3.3 Combined Input/Output Queueing (CIOQ)

CIOQ architecture combines the main features of input queueing and output queueing by having queues both at the inputs and the outputs. If input queueing

is more practical for implementation, the scheduling algorithms proposed for output queueing help in realizing essential QoS guarantees. Where the switch fabric in IQ architecture doesn't require any speedup, an ideal OQ architecture requires a speedup of N. For a speedup value in between 1 and N, packets have to be buffered at the inputs before switching, and outputs after switching. Such an architecture is termed as CIOQ architecture (refer Fig. 6). This approach was first proposed by Prabhakar and McKeown [31], and they showed that a CIOQ switch with a speedup of four can behave identical to a FIFO-OQ switch for arbitrary input traffic. Later work showed that a CIOQ switch running at approximately twice the line rate can emulate an output queued switch, using a *stable marriage matching algorithm* for any traffic arrival pattern [7] Therefore the FQ (Fair Queueing) scheduling algorithms can be used to provide QoS guarantees. It should, however, be noted that the speedup also implies a speedup of memory at the inputs and outputs, as well as at the switch fabrics. With speedup, the access time of memory is shortened, and the scheduling decisions should be performed in shorter time intervals. The algorithm used to find a matching of input-output ports has a complexity of $O(N)$. This algorithm has to be run twice every time-slot; and hence is inapt for high speed switches with large number of ports.

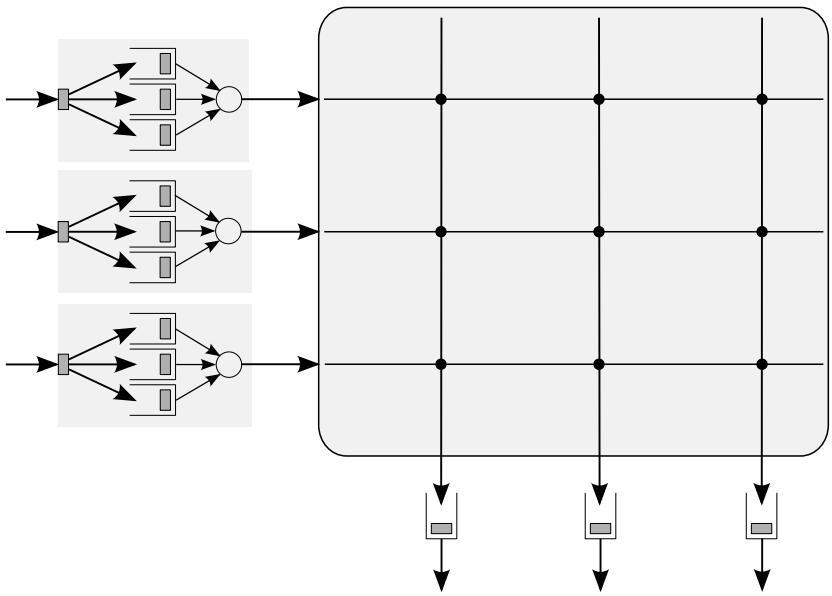


Figure 6: A Combined Input Output Queueing architecture

3.3.1 Parallel packet switching (PPS)

One of the toughest constraints in designing high speed switches is memory speed. All though memory size is also considered as one of the issues, the required amount of buffering is decreasing, and the VLSI density is increasing. Whereas, slow memory will always remain a bottleneck for packets flowing at a higher line rate. A recent study to approach the performance of OQ switch with no speedup of memories is the *parallel packet switching* architecture (PPS).

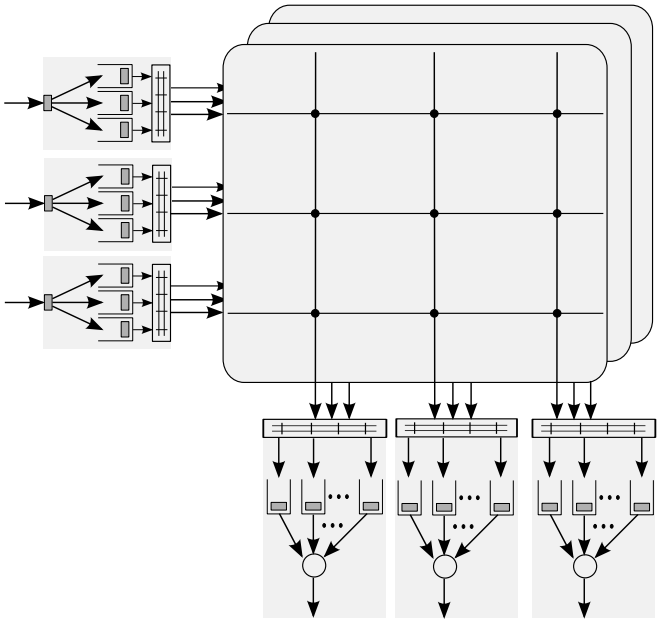


Figure 7: A PMIOQ switch architecture with three crossbars

PPS architecture consists of multiple identical lower-speed parallel switches, operating independently and in parallel [32], [33]. The buffering happens only in these lower-speed switches. Broadly, the operations can be branched into three stages. In the first stage, the incoming stream of packets are spread packet-by-packet by a demultiplexer across k parallel slower switches. In the second stage, the packets are switched to the output port; and finally in the third stage, the packets are recombined by a multiplexer at the output line for transmission. During congestion, cells are stored at the output queues of the slow-speed switches in the second stage. The multiplexer chooses one cell from among the k output queues, when the corresponding output line is free. The slower switches can be CIOQ switches, and need operate only at a rate of $S * R/k$, where R is the line rate and S is the speedup required. The authors prove that if $S \geq 2$, the PPS switch can emulate a FIFO OQ switch. Finally in [34], the authors conclude that the same can be achieved using parallel switches operating at R/k within a bounded delay of $2N$ time slots. Since the packets are buffered at the slower speed switches, buffers in a PPS need not run at line rate. Note that, such architectures lead to mis-sequence of cells, and therefore has to be taken care in the design.

3.3.2 Multiple input/output-queued switch

Along the lines of PPS architecture, [35] proposes a 'parallel multiple input/output queued' (PMIOQ) switch that exactly emulates an OQ switch with a broad class of service scheduling algorithms. An example of of a PMIOQ switch architecture is given in Fig.7. A PMIOQ switch consists of k parallel crossbars, where each crossbar is connected to each input and each output. The architecture uses a *stable strategic alliance* different from the stable marriage matching al-

gorithm, in the sense that the former is a many-to-many matching problem. This is because, multiple cells from the same input can depart at the same time slot, but to different outputs. To elaborate, if there are k parallel switches, up to k cells can be transmitted at the same time, provided they are destined to k different outputs. Similarly, an output port can receive up to k packets at the same time slot, provided they are from k different input ports. To cope up with the switching rate, buffers are maintained at the inputs (VOQs) and outputs. Though the scheduling decision has to be taken just once a time slot, the runtime complexity (of the stable strategic alliance algorithm) is $O(N^2)$.

3.4 Disadvantages of synchronous switching

The synchronous switching of cells between input-output pairs brings in some disadvantages. One major negative point we have seen is that, coupling of input-output pairs increases the complexity of the arbitration algorithms, making fast contention resolution a challenge. It is practically difficult to achieve high throughput and tight QoS guarantees, both, at the same time. We discuss how this barrier can be circumvented (to an extent) by introducing buffers at the switch crosspoints, in Section 4.1

Other disadvantages are associated with one of the common practices in packet switch design. All the scheduling algorithms we discussed until now belong to a class of *cell-mode scheduling* algorithms, wherein packets at the input are segmented to cells of smaller fixed sizes, switched, and reassembled at the output. Reassembly units are required at the output, as cells belonging to different packets can be interleaved at the same output. This indeed gives the switch, its nonblocking capability and simplicity.

On the negative, besides the complexity involved in the segmentation and reassembly, there are mainly two disadvantages associated with the characteristics of cell-mode scheduling: (1) Since packet sizes are generally not multiples of cell size, the segmentation into cells wastes switch bandwidth; or in other words, the crossbar is required to have higher bandwidth in order to compensate the *lost* bandwidth when switching incomplete cells (in fact, such cells are padded with zeroes). In the worst case, a packet might be just one byte greater than a cell size, effectively wasting (almost) half the switch bandwidth. (2) If cells are switched independent of packet boundaries, delay guarantees become hard to achieve. For example, if there are N packets of size K cells, from N different inputs, all to the same output, the transmission of a complete packet can be made only after $N \times (K - 1)$ cell slots.

As described in [7], the second disadvantage can be overcome by using a *Push-In-First-Out* (PIFO) queue. In a PIFO queue, incoming packet is *pushed in* anywhere in the queue based on some criteria, whereas the retrieval is always from the head of the queue. Using PIFO, an arriving cell can be pushed behind the previous cell of the same packet, ensuring the continuous placement of cells of same packet in the queue. Once such a system is in place, the line card can start transmitting the cells of a packet without needing to wait for all the cells forming the packet. Researchers have also used *packet-mode scheduling algorithm* that schedules cell trains (cells belonging to a single packet) contiguously, thus simplifying the complexity at the output [36].

But one major disadvantage, namely the need for higher switch bandwidth, still remains. In [37] Kar *et al.* came up with the solution of having a larger

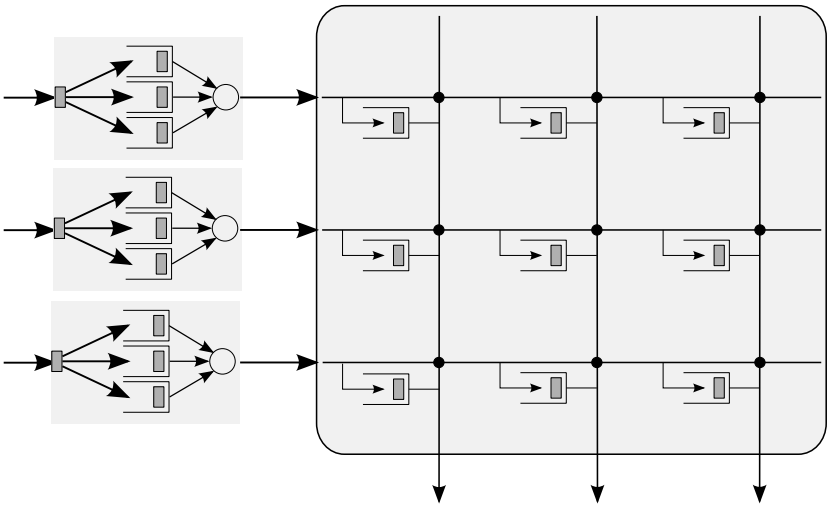


Figure 8: CICQ switch with VOQ

switching unit, which they refer as *envelope*. An envelop is different from a cell in the sense that, it can carry more than one packet, or parts of one or more packets, in general. The goal was to reduce the frequency of scheduling decisions, and one way to achieve this is to increase the size of the switching entity. Since increasing the cell size will result in increase of "wasted" bandwidth, an efficient method is to have an envelope in which the remaining empty space can be filled up by the backlogged packet(s). This gives freedom to increase the envelope size in order to reduce the scheduling frequency. The limitation in the envelop size is revealed under varying traffic characteristics. Under low load, when there is no backlogged packet, envelopes can not wait until packets arrive (as that increases the delay), and hence are switched in partial sizes. Hence, a noted negative point of this method is the increase in jitter with increase in envelope size. In Section 4.2, we look into schemes that switches in *packets* instead of cells.

4 Asynchronous switching

As described earlier, the synchronization in switch arbitration was brought by the need to achieve high throughput when the input and output lines are coupled. One simple way out of this problem, is to introduce buffers at the crosspoints. Such an architecture is called Crosspoint queued (or Crosspoint buffered) architecture. With the introduction of crosspoint buffers, coupling no more exists, and it is now possible to switch packets asynchronously. Later, as we will see, this also removes the constraint on having to switch fixed sized cells.

4.1 Crosspoint queued switches

As the name indicates, crosspoint queued switches have buffers at the crosspoints of a crossbar. Similarly, a Combine Input Crosspoint Queued (CICQ) switch has buffers at the inputs as well as at the crosspoints. Doi and Yamanaka

proposed such an architecture in the early nineties, where each input port has a large buffer, and each crosspoint has a small buffer [38]. Therefore the crossbar buffer size is quadratic in N. Ironically, this is also considered as the major drawback of the buffered crossbar architecture.

Just like the IQ architecture, this architecture suffers from HOL blocking; and hence a CICQ switch with VOQ was proposed in [39]. The architecture is illustrated in Fig.8. Such a switch uses two scheduling algorithms. One is used to determine the cell to be transmitted from the input buffer to the crosspoint (input scheduling); and the second algorithm determines the cell to be transmitted from the crosspoint buffer to the output port. The scheduling algorithms presented in [39] selects the cells (to be scheduled) based on delay times at the buffers.

There are at least two advantages of using internal buffered crossbar. First, the scheduling becomes completely distributed due to the crosspoint buffers. The processing can run on each input and output, thus eliminating the need for a centralized scheduler. Second, and equally important, multiple input buffers can transfer cells to the same output, i.e., simultaneous cell transfers to the same output can be made as the cells can be stored at the crosspoint buffers.

4.1.1 Combined Input Crosspoint Buffered Switch

The CICQ switch is also known as Combined Input Crosspoint Buffered (CIXB) switch. The initial algorithms for CIXB switch with VOQ that achieved 100% throughput assumed infinite buffer size at the crosspoints [40], [41]. The authors of [42] came up with a one-cell crosspoint (X) buffer (B) architecture, called CIXB-1 architecture. It uses round-robin input and crosspoint arbitrations to achieve 100% throughput under uniform traffic. The input arbiter selects a VOQ in a round-robin fashion, to determine the cell to be sent to the corresponding crosspoint buffer. Likewise, the crosspoint buffer eligible to send cell to the output is chosen in round-robin. Later in [43], the authors show that the throughput is as high as 84% under non-uniform traffic, and increases slowly and asymptotically as the crosspoint buffer size increases. A similar round-robin scheme was also proposed by K. Yoshigoe and K. J. Christensen in [44] (refer Fig. 9). The authors confirmed that the RR/RR (round-robin at input and crosspoints) CICQ switch has lower delay at high offered loads (for both Bernoulli arrivals and bursty arrivals, with uniformly randomly chosen output port) than IQ switch with PIM or iSLIP scheduling [45].

Recently, a fair scheduling algorithm was proposed for CICQ switches with fixed-size crosspoint buffers in [46]. The algorithm allocates fair quotas to ports depending on its queue length. The more quotas a port receives, the more it lags behind, and hence it gets prioritized. A threshold based mechanism is used to determine the input-output pairs that need prioritization. Dual round-robin pointers are used for input scheduling, and RR is used for output scheduling. One of the dual pointers points to the highest priority VOQ, and the other to a non-prioritized VOQ. The input scheduler selects the first pointer, as long as it is valid. This scheme is shown to achieve over 99% throughput and relatively low mean delay under different traffic patterns, besides providing max-min fairness.

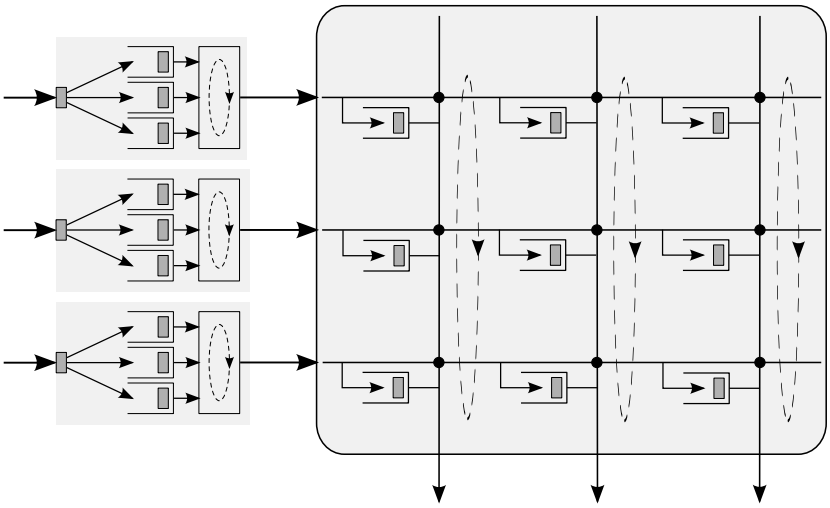


Figure 9: A round robin scheme for a CIXB-1 switch architecture

4.1.2 Combined Input-Crosspoint-Output Buffered

As the CIXB-1 switch doesn't provide 100% throughput for unbalanced traffic, Rojas-Cessa *et al.* extended the work on buffered crossbar to an architecture with buffers not only at the inputs and crosspoints, but also at the outputs [47]. It is referred to as CIXOB-k architecture, where k is the size of the crosspoint buffer. The proposed architecture depicted in Fig. 10, uses a small speedup along with round-robin arbitrations to to achieve 100% throughput under both uniform as well as non-uniform traffic. The speedup factor and the value of k are inter-dependent here. Still, to achieve 100% throughput with $k = 1$, the speedup factor is lesser than that required by an unbuffered crossbar using maximal-matching arbitration scheme.

Later, in [48], the authors prove that a CIXOB switch can achieve 100% throughput with any Bernoulli i.i.d admissible traffic. They also show that such an architecture can provide both rate and delay guarantees.

4.2 From cell switching to packet switching

Distinguishing from cell-mode scheduling, variable length packet scheduling, or *packet scheduling* for short, transmits packets through the switch fabrics. This kind of switching removes the disadvantage of cell-mode switching, and achieves greater throughput compared to cell scheduling. Moreover, packet scheduling also completely removes segmentation and reassembly, in the process removing the need for buffers that were otherwise required for segmentation and reassembly. These reduce delay and hardware cost. In the following, we look into some advance in asynchronous packet switching architecture. Note that, since there is no more centralized arbitration function, and instead what we have are packet schedulers at inputs, crosspoints and outputs, we refer to the function as *scheduling*.

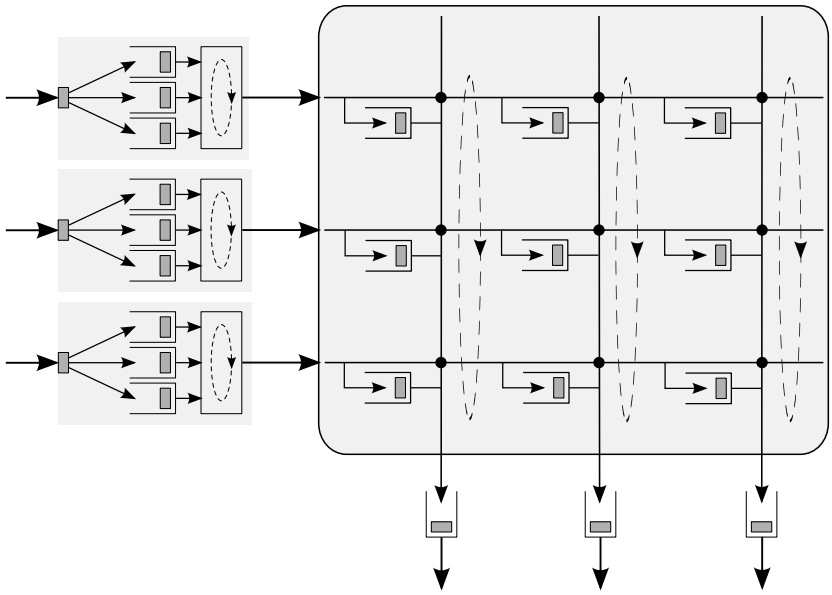


Figure 10: Combined Input Crosspoint Output Buffered Architecture

4.2.1 Asynchronous packet scheduling in CICQ architecture

A variable-size packet scheduling scheme for CICQ architecture was proposed in [49]. The authors present chip layout, and consider the hardware implementation and give cost numbers (in terms of power consumption and silicon). The schedulers are placed on the input line cards and output line cards. The input scheduler is notified of the crosspoint buffer occupancy and the corresponding departing output port. The output scheduler can be made as sophisticated as desired, with the common requirement that information from the buffers be fed to it. With no speedup, the authors show that the proposed design outperforms synchronized unbuffered crossbar (with iSLIP) in terms of delay and throughput.

4.2.2 Asynchronous packet scheduling in CIXOB architecture

In [50], Deng Pan and Yuanyuan Yang proposed LAPS, *localized asynchronous packet scheduling*, in crossbar switches (CIXOB architecture). The switch fabric has a speedup of two. At the input, a packet is transmitted from a backlogged VOQ (using arbitration rule such as RR) to the corresponding crosspoint buffer when it is free, and the output selects a crosspoint buffered packet and saves it in the output queue; both these functions being executed independently. The scheme also uses *cut-through* switching, in which buffering is minimized by sending a packet directly, say, from a VOQ to the output queue, or even to the output line card (if it supports such a mechanism) without being buffered at the crosspoint buffer or output queue. The authors prove that LAPS achieve 100% throughput for any admissible traffic, with a buffer length equal to the maximum packet size at the crosspoints, and a speedup of two.

Turner studied the performance of two algorithms that switch packets asynchronously in [51, 52]. The conditions for preserving packet arrival order (while forwarding) and for the scheduler to be work conserving was revealed in this work. It was proved that for packets of maximum length L, with a speedup of two and buffer sizes greater than or equal to $2L$, a buffered crossbar (CIXOB) scheduler can provide work conserving guarantees. These schedulers can also emulate an OQ switch within a class of restricted queueing disciplines.

4.3 Pros and cons

The need for a synchronous centralized scheduler, that doesn't scale up with the number of ports, is removed with buffered crossbars. Though asynchronous mode of input-output transfers can be made with bufferless crossbar [53], the performance (in terms of throughput and delay) is better with crosspoint buffered crossbars. The addition of crosspoint buffers had been considered too expensive with the need to have $O(N^2)$ buffer space at the crosspoints. But as the authors in [48] argue, with today's technology, it is now practical to have buffers as the crosspoints. With buffered crossbars, using either cell-based switching or packet-based switching, it is possible to achieve strong performance guarantees, though with a speedup of two.

5 Conclusions

Table 1: A summary of different crossbar architectures

Arch.	**Advantage**	**Disadvantage**
OQ	100 % throughput for any traffic, QoS guarantees	Speedup of N required, impractical
IQ-VOQ	Simplicity, No HOL blocking	No performance guarantees
CIOQ	Emulation of OQ behaviour	Speedup, complex scheduling algorithms
CICQ	Decoupling of input and output, better performance than unbuffered crossbars	Compromise between crosspoint buffer size and performance
CIXOB	100 % throughput for any traffic, QoS guarantees, simple schedulers	Switch speedup

Table 1 summarizes the main advantages and disadvantages of different architectures. While the OQ switch achieve 100% throughput for any traffic, it is impractical. The simple IQ switch was extended with the VOQ structure to mitigate the HOL blocking problem. With internal speedup and output queues,

the CIOQ switch has been able to match the OQ switch in performance. Parallel packet switches used slower speed switches to emulate an OQ switch. The introduction of buffer at the crosspoints removed the coupling of input and output lines, making the arbitration decision completely decentralized. Switches with buffered crossbars are now feasible, and as a result recent research works have been using buffered crossbar to provide strong performance guarantees.

We see that the evolution of switch designs has been more or less driven by the need for performance, besides the advancement in technology. Achieving 100% throughput has been one important design criteria that most research works focussed on. The need to provide QoS guarantees also gained equal importance during these years. Similarly, technology has now facilitated the incorporation of queues at the crosspoints, although the idea was introduced in early nineties.

To get an insight into future switch designs and architectures, we need to understand the driving force. One such driving force is, perhaps, power consumption. Lately, power consumption at the interconnects has been drawing serious attention to such an extent that, operators have started to demand greener interconnects [54, 55]. Consequently, techniques for power-efficient computer architectures are being explored [56], and solutions such as dynamic speed scaling are being proposed [57, 58]. Therefore, it is possible that the future switch architectures will be evolving to meet the design criteria of lower power consumption. The reduction in power consumption might bring in performance degradation. The question then becomes, how much of degradation is the operator (and in turn, the users) ready to sacrifice to reduce energy cost.

Similarly, with the new trend in 'flow-aware' networking, researchers have maintained the need to provide QoS at flow level [59]. While all the switching designs explored in this article have focussed on packet-level QoS, the future might tend to operate at larger units which make sense for users, such as flows or sub-flows. In such scenarios, the definition of QoS will also turn out to be at flow level rather than at packet level.

Yet another possible step will be in the direction of optical switching. With the optical technologies such as OCS (Optical Circuit Switching), OPS (Optical Packet Switching) and OBS (Optical Burst Switching) attaining more and more importance in the research community, the eventuality of optical networks using optical cross-connects is certain. The cross-connect here resembles the switch in packet-switched network; and it can be realized in electronic domain, or in optical domain, or even using both electronic and optical modules to perform different tasks. All-optical switch fabrics are known to be power-efficient compared to their counterparts in the electronic domain. Besides, they do not have the scalability problem that electronic switches face in keeping up with the increasing capacity. Again, it will be technology that determines which among these three (OCS, OPS or OBS) paradigms or a hybrid will finally define the future networks[1]. If an all-optical switching network becomes a reality, the emergence of new switching designs will take place depending on factors such as existence or non-existence of memory (using delay loops) at the cross-connects, technological development of wavelength converters, the level of QoS guarantees required etc.

[1] We refer readers to [60] and references therein for related pointers.

References

[1] George Varghese, *Network Algorithmics: An Interdisciplinary Approach to Designing Fast Networked Devices (The Morgan Kaufmann Series in Networking)*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[2] Robert Endre Tarjan, *Data structures and network algorithms*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1983.

[3] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," in *SIGCOMM '89: Symposium proceedings on Communications architectures & protocols*, 1989, pp. 1–12.

[4] Abhay K. Parekh and Robert G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: the single-node case," *IEEE/ACM Trans. Netw.*, vol. 1, no. 3, pp. 344–357, 1993.

[5] Abhay K. Parekh and Robert G. Gallagher, "A generalized processor sharing approach to flow control in integrated services networks: the multiple node case," *IEEE/ACM Trans. Netw.*, vol. 2, no. 2, pp. 137–150, 1994.

[6] M. Shreedhar and George Varghese, "Efficient fair queueing using deficit round robin," in *SIGCOMM '95: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, 1995, pp. 231–242.

[7] S. T. Chuang, A. Goel, N. McKeown, and B. Prabhakar, "Matching output qeueueing with a combined input/output-queued switch," *IEEE J. Selected Area in Commun.*, vol. 17, no. 6, pp. 1030–1039, Jun 1999.

[8] Sundar Iyer and Nick McKeown, "Techniques for fast shared memory switches," Tech. Rep. HPNG Technical Report - TR01-HPNG-081501, Stanford University, 2001.

[9] Robert J. Souza, P. G. Krishnakumar, Cüneyt M. Özveren, Robert J. Simcoe, Barry A. Spinney, Robert E. Thomas, and Robert J. Walsh, "GIGAswitch System: A High-performance Packet-switching Platform," *Digital Technical Journal*, vol. 6, no. 1, pp. 9–22, 1994.

[10] M. Karol, M. Hluchyj, and S. Morgan, "Input versus output queueing on a space division switch," *IEEE Trans. Commun.*, vol. 35, pp. 1347–1356, Dec. 1987.

[11] Y. Tamir and G. L. Frazier, "High-performance multi-queue buffers for VLSI communications switches," *SIGARCH Comput. Archit. News*, vol. 16, no. 2, pp. 343–354, 1988.

[12] H. Obara, "Optimum architecture for input queuing ATM switches," *Electronics Letters*, vol. 27, no. 7, pp. 555–557, Mar. 1991.

[13] H. Obara and Y. Hamazumi, "Parallel contention resolution control for input queuing ATM switches," *Electron. Lett.*, vol. 28, no. 9, pp. 838–839, Apr. 1992.

[14] Thomas E. Anderson, Susan S. Owicki, James B. Saxe, and Charles P. Thacker, "High-speed switch scheduling for local-area networks," *ACM Trans. Comput. Syst.*, vol. 11, no. 4, pp. 319–352, 1993.

[15] Thomas E. Anderson, Susan S. Owicki, James B. Saxe, and Charles P. Thacker, "High speed switch scheduling for local area networks," in *Proceedings of the fifth international conference on Architectural support for programming languages and operating systems, ASPLOS-V*, 1992, pp. 98–110.

[16] N. McKeown, A. Mekkittikul, V. Anantharam, and J. Walrand, "Achieving 100% throughput in an input-queued switch," *IEEE Trans. Commun.*, vol. 47, no. 8, pp. 1260–1267, Aug. 1999.

[17] Adisak Mekkittikul and Nick McKeown, "A Starvation-free Algorithm for Achieving 100% Throughput in an Input-Queued Switch," in *Proc. ICCCN '96*, Oct. 1996, pp. 226–231.

[18] L. Tassiulas and A. Ephremides, "Stability properties of constrained queueing systems and scheduling for maximum throughput in multihop radio networks," *IEEE Transactions on Automatic Control*, vol. 37, no. 12, pp. 1936–1949, Dec. 1992.

[19] J. G. Dai and Balaji Prabhakar, "The Throughput of Data Switches with and without Speedup," in *Proc. IEEE INFOCOM*, 2000, pp. 556–564.

[20] H. Duan, J. Lockwood, and S. Kang, "Matrix unit cell scheduler (MUCS) for input-buffered ATM switches," *IEEE Commun. Lett.*, vol. 2, no. 1, pp. 20–23, 1998.

[21] Nick McKeown, "The iSLIP scheduling algorithm for input-queued switches," *IEEE/ACM Trans. Netw.*, vol. 7, no. 2, pp. 188–201, 1999.

[22] M.A. Marsan, A. Bianco, E. Leonardi, and L. Milia, "RPA: A flexible scheduling algorithm for input buffered switches," *IEEE Trans. Commun.*, vol. 47, no. 12, pp. 1921–1933, 1999.

[23] Dimitrios N. Serpanos and Panagiotis Antoniadis, "FIRM: A Class of Distributed Scheduling Algorithms for High-Speed ATM Switches with Multiple Input Queues," in *Proc. IEEE INFOCOM*, 2000, pp. 548–555.

[24] Devavrat Shah, Paolo Giaccone, and Balaji Prabhakar, "Efficient Randomized Algorithms for Input-Queued Switch Scheduling," *IEEE Micro*, vol. 22, no. 1, pp. 10–18, 2002.

[25] Paolo Giaccone, Balaji Prabhakar, and Devavrat Shah, "Towards Simple, High-performance Schedulers for High-aggregate Bandwidth Switches," in *Proc. INFOCOM*, 2002.

[26] Gagan Aggarwal, Rajeev Motwani, Devavrat Shah, and An Zhu, "Switch Scheduling via Randomized Edge Coloring," in *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science, FOCS '03*, 2003, p. 502.

[27] Adnan Aziz, Amit Prakash, and Vijaya Ramachandra, "A near optimal scheduler for switch-memory-switch routers," in *Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures, SPAA '03*, 2003, pp. 343–352.

[28] Devavrat D. Shah, *Randomization and heavy traffic theory: new approaches to the design and analysis of switch algorithms*, Ph.D. thesis, Stanford, CA, USA, 2005, Adviser-Balaji Prabhakar.

[29] Petar Momčilović, "A distributed switch scheduling algorithm," *Perform. Eval.*, vol. 64, no. 9-12, pp. 1053–1061, 2007.

[30] L. Tassiulas, "Linear complexity algorithms for maximum throughput in radio networks and input queued switches," in *Proc. IEEE INFOCOM*, 1998, pp. 533–539.

[31] Balaji Prabhakar and Nick McKeown, "On the Speedup Required for Combined Input and Output Queued Switching," Tech. Rep., Stanford, CA, USA, 1997.

[32] Sundar Iyer, A. Awadallah, and N. McKeown, "Analysis of a packet switch with memories running slower than the line rate," in *Proc. IEEE INFOCOM*, 2000, pp. 529–537.

[33] Sundar Iyer and Nick McKeown, "Making Parallel Packet Switches Practical," in *Proc. IEEE INFOCOM*, 2001, pp. 1680–1687.

[34] Sundar Iyer and Nick W. McKeown, "Analysis of the parallel packet switch architecture," *IEEE/ACM Trans. Netw.*, vol. 11, no. 2, pp. 314–324, 2003.

[35] Hyoung-Il Lee and Seung-Woo Seo, "Matching output queueing with a multiple input/output-queued switch," *IEEE/ACM Trans. Netw.*, vol. 14, no. 1, pp. 121–132, Feb 2006.

[36] Marco Ajmone Marsan, Andrea Bianco, Paolo Giaccone, Emilio Leonardi, and Fabio Neri, "Packet-mode scheduling in input-queued cell-based switches," *IEEE/ACM Trans. Netw.*, vol. 10, no. 5, pp. 666–678, 2002.

[37] K. Kar, T. V. Lakshman, D. Stiliadis, and L. Tassiulas, "Reduced complexity input buffered switches," in *Proc. Hot Interconnects VIII*, Aug 2000.

[38] Y. Doi and N. Yamanaka, "A High-Speed ATM switch with Input and Cross-Point Buffers," *IEICE Transactions on Communications*, vol. E76-B, no. 3, pp. 310–314, Mar 1993.

[39] Masayoshi Nabeshima, "Performance Evaluation of a Combined Input- and Crosspoint-Queued Switch," *IEICE Transactions on Communications*, vol. vE83-B, no. 3, pp. 737–741, Mar 2000.

[40] F. A. Tobagi, "Fast Packet Switch Architectures for Broadband Integrated Services Digital Networks," *Proc. of the IEEE*, vol. 78, no. 1, pp. 133–167, 1990.

[41] R. Y. Awdeh and H. T. Mouftah, "Survey of ATM Switch Architectures," *Computer Networks and ISDN Systems*, vol. 27, pp. 47–93, 1995.

[42] R. Rojas-Cessa, E. Oki, Z. Jing, and H.J. Chao, "CIXB-1: Combined input-one-cell-crosspoint buffered switch," in *Workshop on High Performance Switching and Routing (HPSR2001)*, May 2001, pp. 324–329.

[43] R. Rojas-Cessa, E. Oki, and H. J. Chao, "On the combined input-crosspoint buffered switch with round-robin arbitration," *IEEE Transactions on Communications*, vol. 53, no. 11, pp. 1945–1951, 2005.

[44] K. Yoshigoe and K.J. Christensen, "A parallel-polled virtual output queued switch with a buffered crossbar," in *Proc. IEEE Workshop on High Performance Switching and Routing*, 2001, pp. 271–275.

[45] K. Yoshigoe and K. J. Christensen, "An evolution to crossbar switches with virtual output queuing and buffered cross points," *IEEE Network*, vol. 17, no. 5, pp. 48–56, 2003.

[46] N. Hua, P. Wang, D. Jin, L. Zeng, B. Liu, and G. Feng, "Simple and Fair Scheduling Algorithm for Combined Input-Crosspoint-Queued Switch," in *Proc. IEEE International Conference on Communications, ICC '07*, Jun 2007, pp. 6305–6310.

[47] R. Rojas-Cessa, E. Oki, and H. Jonathan Chao, "CIXOB-k: Combined Input-Crosspoint-Output Buffered Packet Switch," in *Proc. IEEE Globecom '01*, 2001, pp. 2654–2660.

[48] Shang tse Chuang, Sundar Iyer, and Nick Mckeown, "Practical algorithms for performance guarantees in buffered crossbars," in *IEEE INFOCOM*, Mar 2005, pp. 981–991.

[49] Manolis Katevenis, Georgios Passas, Dimitrios Simos, Ioannis Papaefstathiou, and Nikos Chrysos, "Variable packet size buffered crossbar (CICQ) switches," in *IEEE International Conference on Communications (ICC 2004)*, Jun 2004, pp. 1090–1096.

[50] Deng Pan and Yuanyuan Yang, "Localized asynchronous packet scheduling for buffered crossbar switches," in *Proceedings of the 2006 ACM/IEEE symposium on Architecture for networking and communications systems, ANCS '06*, 2006, pp. 153–162.

[51] Jonathan Turner, "Strong Performance Guarantees for Asynchronous Crossbar Schedulers," in *Proc. IEEE INFOCOM*, Apr 2006, pp. 1–11.

[52] Jonathan Turner, "Strong Performance Guarantees for Asynchronous Buffered Crossbar Schedulers," *IEEE/ACM Trans. Netw.*, 2009, To appear.

[53] G. Passas and M. Katevenis, "Asynchronous operation of bufferless crossbars," in *Workshop on High Performance Switching and Routing, HPSR '07*, Jun 2007, pp. 1–6.

[54] J. Baliga, R. Ayre, W.V. Sorin, K. Hinton, and R.S. Tucker, "Energy consumption in access networks," Feb. 2008, pp. 1–3.

[55] "Telcos demand greener network equipment," http://www.reuters.com/article/idUSN1847837420080619, 2008.

[56] Stefanos Kaxiras, *Computer Architecture Techniques for Power-Efficiency*, Morgan and Claypool Publishers, 2008.

[57] Sebastian Herbert and Diana Marculescu, "Analysis of dynamic voltage/frequency scaling in chip-multiprocessors," in *ISLPED '07: Proceedings of the 2007 international symposium on Low power electronics and design*. 2007, pp. 38–43, ACM.

[58] Adam Wierman, Lachlan L. H. Andrew, and Ao Tang, "Power-aware speed scaling in processor sharing systems," in *Proc. IEEE INFOCOM*, Apr 2009.

[59] T. Bonald, S. Oueslati-Boulahia, and J. Roberts, "IP traffic and QoS control: the need for a flow-aware architecture," in *World Telecommunications Congress*, Sep. 2002.

[60] Andrew Zalesky, "To burst or circuit switch?," *IEEE/ACM Trans. Netw.*, vol. 17, no. 1, pp. 305–318, 2009.

Centre de recherche INRIA Grenoble – Rhône-Alpes
655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399