1

Architectures of Internet Switches and Routers

Xin Li, Lotfi Mhamdi, Jing Liu, Konghong Pun, and Mounir Hamdi

The Hong-Kong University of Science & Technology. {lixin,lotfi,liujing, konghong,hamdi}@cs.ust.hk

1.1 Introduction

Over the years, different architectures have been investigated for the design and implementation of high-performance switches. Particular architectures were determined by a number of factors based on performance, flexibility and available technology. Design differences were mainly a variation in the queuing functions and the switch core. The crossbar-based architecture is perhaps the dominant architecture for today's high-performance packet switches (IP routers, ATM switches, and Ethernet switches) and owes its popularity to its scalability (when compared to the shared-bus/shared-memory architectures), efficient operation (supports multiple I/O transactions simultaneously) and simple hardware requirements. The architecture includes the input-queued (IQ) crossbar fabric switch with its variations (Output-queued, OQ, switch and Combined Input–Output-queued, CIOQ, switch) and the internally buffered crossbar fabric switch (BCS).

IQ switches have gained much interest in both academia and industry because of their low cost and scalability. The IQ switch has a low internal speedup because the crossbar fabric has the same speed as that of the external line. Although the head-of-line (HoL) blocking problem limits the achievable throughput of an IQ switch to approximately 58.6% [1], the well-known virtual output queuing (VOQ) architecture [2] was proposed and has improved switching performance by several orders of magnitude, making IQ switches more desirable. However, the adoption of VOQ has created a more serious problem, namely, the centralized scheduler. An arbitration algorithm examines the contents of all the input queues, and finds a conflict-free match between inputs and outputs. The well-known optimal algorithms (*i.e.* maximum-weight-matching or MWM) are too complex to implement at high speed while the iterative algorithms, proposed as an alternative to the MWM algorithms, fail to perform well under real world input traffic conditions.

As bufferless scheduling algorithms reach their practical limitations due to higher port numbers and data rates, internally buffered crossbar switches (BCS) have started to attract researchers because of the great potential they have in solving the complexity and scalability issues faced by their bufferless predecessors. The increas-

ing demand for terabit switches and routers means that future commercial packet switches must be implemented with reduced scheduling complexity. The buffered crossbar architecture can inherently implement distributed scheduling schemes and has been considered a viable alternative to bufferless crossbar switches to improve performance. The presence of internal buffers drastically improves the overall performance of the switch as it offers two distinct advantages. First, the adoption of internal buffers makes the scheduling totally distributed, dramatically reducing the arbitration complexity. Second, and most importantly, these internal buffers reduce (or avoid) output contention as they allow the inputs to make cell transfers concurrently to a single output. While there have been many architectures for the (BCS) [3, 4, 5], our focus in this chapter is on BCS with VOQs (denoted by VOQ/BCS).

Despite the advantages that the VOQ/BCS architecture offers with regards to the bufferless architecture, both are seen as unscalable and unable to keep up with Internet growth in the foreseeable future. In fact, with progress in wavelength division multiplexing (WDM) technology and optical fiber transmission, switches and routers are becoming the bottleneck of the overall system. The increasing data rates on the Internet are causing a scalability challenge for these crossbar-based architectures. This scalability limitation can be attributed to the nature of crossbar-based fabric, as a quadratic growth in the number of crosspoints puts a limit on chip area and/or pin counts. As a result, there is an urgent need for truly scalable high-performance packet switches that has motivated researchers and industry to look for alternative switch fabrics. In particular, architectures such as the Clos-network switch architecture and the optical-electronic switch architecture are starting to receive attention because of their great potential in entirely solving the scalability issues.

The Clos-network switch architecture is the most popular example among numerous examples of multi-stage switches in the communication world. It is mainly a three-stage switch as shown in Figure 1.11. There are switching elements (SE) at each stage which are connected. While the SEs can be any interconnect, typically they are crossbars of smaller sizes. The SE at each stage of the Clos network is capable of transmitting a packet from an input port to any of the output ports. Clos-network switches can be classified as buffered or bufferless, where the former uses buffers to store packets in the second-stage SEs and the latter does not [6, 7]. Buffers in the second-stage SEs can help resolve contention among packets from different first-stage modules but may cause a sequencing problem. While the Clos-network switches have been proposed for a some time now, relatively little research (especially compared to crossbars) has been conducted on making them scalable and high-speed packet switches that can serve the needs of the Internet. One of the goals of the current survey is to fill this gap.

Given the effort put by both industry and academia into the all-electronic switch/router design, numerous studies show that all-electronic technology can no longer be a viable solution for the design of scalable switches/routers (*i.e.* 256×256 and beyond). Alongside the scalability challenge, an electronic switch fabric becomes very costly beyond a reasonable size (*e.g.* 128×128) and data rate (*e.g.* 10 Gb/s) due to the ever-increasing power and chip count requirements for its implementation. Moreover, current WDM systems offer 32–64 wavelengths at 2.5–10Gb/s/wavelength,

approaching a 1 Tb/s capacity, while research-level systems already exceed multi-terabits in a single fiber. As a result, traffic growth will not be limited by fiber bandwidth and/or optical components in the links. The total data rate of a single fiber is increasing at a rate faster than the switching and routing equipment that terminates and switches traffic at a carrier's central office or point of presence (POP). In particular, switches and routers are becoming the bottlenecks of the overall system. While there has been a lot of attention paid to all-optical switches as a solution to this bottleneck problem, most current POP switching equipment is electronic switches/routers with optical I/O, and an all-optical solution in not expected to be viable in the foreseeable future. For this reason, research focus is shifting towards a solution that takes advantage of the strengths of both electronics and optics, with the ultimate goal of designing practical switches and routers that can scale with the Internet traffic growth as well as keep up with the advances in WDM fiber transmission. Recently, attention has been given to hybrid optical–electronic architectures where the linecards and switch scheduler are designed using electronics and the switch fabric is designed with optical technology. This switch architecture provides advantages such as scalability, lower power consumption, and lower cost. However, a hybrid opto-electronic packet switch/router presents a unique challenge: the reconfiguration time of an optical switch fabric is much longer than that of an electronic fabric, and the end-to-end clock recovery in such a system adds to the reconfiguration overhead.

The goal of this chapter is to survey each of the above presented architectures, and will be organized as follows. Section 1.2 presents the bufferless switching architecture with its different types and scheduling schemes. Section 1.3 presents the buffered crossbar switch architecture and discusses its scheduling processes and proposed schemes. In Section 1.4, we present the Clos-network architecture and discuss its variants and scheduling. Section 1.5 presents the optical switching trend and the potential for building such cost-effective and highly scalable switches. Section 1.6 concludes the paper and suggests problems for further research.

1.2 Bufferless Crossbar Switches

1.2.1 Introduction to Switch Fabrics

Crossbar switch fabric is an active non-blocking switch fabric. In crossbar architecture, each line card is connected by a dedicated point-to-point link to the central switch fabric. The inputs and outputs are connected at switching points, called crosspoints. It is the scheduler's responsibility to set configurations for the crosspoint matrix and for each configuration. One input/output can only be connected to one output/input. The basic architecture of a crossbar switch is shown in Figure 1.1.

For bufferless crossbar switches, there is no buffer at the crosspoint. However, buffers may be placed at the input side, output side, or both. Based on where the buffers are placed, bufferless crossbar switches are categorized into input-queued switches, output-queued switches, and combined input–output queued switches.

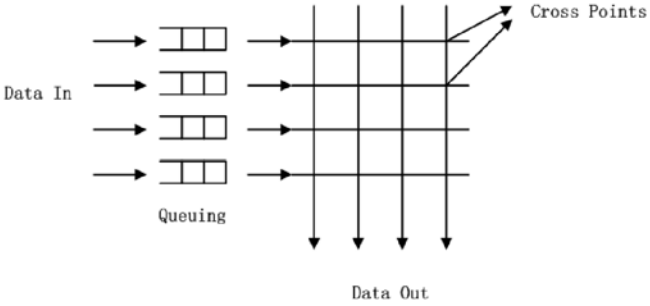


Figure 1.1. Crossbar architecture

1.2.2 Output-queued Switches

Traditionally, switches and routers have been most often designed with an output queuing strategy. This strategy has advantages in that guaranteed qualities-of-service (QoS) can be provided for the system, which is able to control packet departure times [8, 9]. An output-queued switch is attractive as it can always achieve 100% throughput. However, since there are no queues at the inputs, all arriving cells must be immediately delivered to their outputs. The necessary simultaneous delivery of all arriving cells to the outputs becomes a disadvantage if the requirements for internal interconnection bandwidth and memory bandwidth are too great. For a switch with N input ports, there can be up to N cells, one from each input, arriving at any one output simultaneously. Thus, in order to receive all the cells at one time the memory needs a bandwidth of N cell time write accesses. This requirement is referred to as the internal speedup of the switch [10], so an output-queued switch has an internal speedup of N. The current demand for bandwidth is growing rapidly and as switch sizes continue to increase, memory bandwidth will be insufficient for output queuing to be practical.

1.2.3 Input-queued Switches

Memory bandwidth is not a problem with the input queuing strategy. In input-queued switches, arriving cells are buffered on the input side and extracted to pass through the switch fabric according to some arbitration algorithm. Contention within the switch fabric and input/output interfaces is resolved by the arbitration algorithm (each input can deliver at most one cell to the switch in one cell time and each output can accept no more than one cell in one cell time).

Queuing Strategies

- *Single FIFO and Head-of-line Blocking*: It is relatively easy to implement single FIFO (first-in-first-out) switches. A single FIFO queue is used at each input

where only the first cell in each queue is eligible to be forwarded. Head-of-line (HoL) blocking may occur where no cell can be transmitted from an input because of a contention of the first packet in the FIFO queue. Though the next cell in the queue may be without contention, it cannot be transmitted as this would disrupt the queue. Single FIFO switches suffering from HoL blocking result in poor performance. It is well known that such a switch with Bernoulli I.I.D. arrivals under uniform traffic can only achieve a maximum throughput of 58.6% when the number of ports is large [1]. For periodic traffic, HoL blocking can lead to even worse performance [11].

- *Windowing Mechanism*: This mechanism enables the scheduler of the switch to look ahead w (window size) cells in the queue at a time, relieving the HoL blocking problem and increasing performance by allowing cells behind the head of the queue to be extracted. If the cell at the head of the queue cannot be transferred to the intended output because of contention, the second cell is considered, and so on, up to the wth queue position. Note that when $w = 1$, it is single FIFO queuing. The performance of input-queued switches adopting this queue organization grows with w, however, this gain is limited as the complexity of the queuing mechanism increases.
- *Virtual Output Queuing*: HoL blocking can be completely eliminated by using a virtual output queuing [2] architecture at the input side. Rather than maintaining a single FIFO queue for all cells, each input maintains a separate queue for each output as shown in Figure 1.2. There are thus a total of N^2 input queues, where each separate queue is called a VOQ and operates according to the FIFO discipline. The scheduler will select among the HoL cells of each VOQ and transmit them. HoL blocking is eliminated as no cell can be held up by a cell ahead of it that is destined for a different output. When virtual output queuing is employed, the performance of the switch depends on the scheduling algorithm that decides which cells should be transmitted during a cell time under the condition that only one cell can be delivered to each input and only one cell can be accepted at each output. With suitable scheduling algorithms, an input-queued switch using virtual output queuing can increase the throughput from 58.6% to 100% for both uniform and non-uniform traffic [12].

1.2.4 Scheduling Algorithms for VOQ Switches

Per one time slot, each input can send at most one cell and each output can receive at most one cell. Scheduling algorithms are necessary for crossbar switches to find a proper one-to-one match between inputs and outputs in order to configure the crossbar. A variety of scheduling algorithms are proposed for the VOQ architecture. This section presents an overview of some popular and effective schemes.

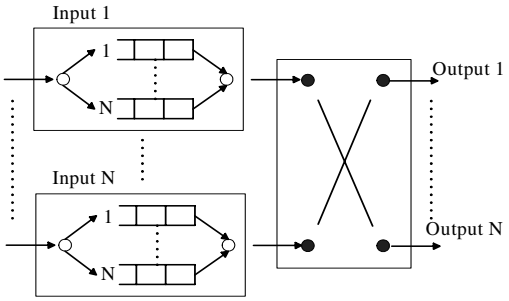


Figure 1.2. Virtual output queuing

Maximum Weight Matching Scheduling Algorithms

In maximum weight matching algorithms, a weight is attached with each request from inputs to outputs. The weight could be the number of cells in the VOQ, the waiting time of the HoL cell in the VOQ, etc.

- *LQF*: The longest queue first (LQF) algorithm [13] takes the number of cells in each VOQ as weight. The algorithm picks a match such that the sum of served queues' lengths is maximized. LQF can lead to the starvation of some queues. Because it does not consider the waiting time of cells, queues with short length may be starved even though the wait time of their HoL cells surpasses the total weight time experienced by cells in a longer queue.
- *OCF*: The oldest cell first (OCF) algorithm [13] uses the waiting times of HoL cells as weights. The OCF algorithm selects a match such that the sum of all served queues' waiting time is maximized. Unlike the LQF algorithm, the OCF algorithm does not starve any queue and unserved HoL cells will eventually become old enough to be served.
- *LPF*: Both LQF and OCF have high computation complexity of $O(N^3 \log N)$ and it is impractical to implement them in hardware. The longest port first (LPF) algorithm [14] is designed to overcome the complexity problem of LQF and OCF and can be implemented in hardware at high speed. The weight of the LPF algorithm is the total number of cells queued at the input and output interfaces: $w_{ij} = \sum_{k=1}^{N} L_{i,k} + \sum_{k=1}^{N} L_{k,j}$ (L_{ij} is the number of cells in VOQ_{ij}). This sum is called port occupancy, which represents the workload or congestion that a cell faces as it competes for transmission. It is proved in [14] that the maximum weight match found by LPF is also a maximum size match. Thus, a modified maximum size matching algorithm, which makes LPF less complex than LQF, is used to implement LPF.

MWM scheduling algorithms achieve 100% throughput under any admissible traffic. However, their good performance and stability come at the expense of high computation complexity.

Approximating Maximum Size Matching Scheduling Algorithms

Approximating maximum size matching algorithms are fast and simple to implement in hardware with today's technologies. They provide 100% throughput under uniform traffic and fairly good delay performance as well. However, they are not stable under non-uniform traffic. Most of the approximating maximum size matching algorithms are iterative algorithms.

- *PIM*: The Parallel Iterative Matching (PIM) [15] algorithm attempts to approximate a maximum size matching algorithm by iteratively matching the inputs to the outputs until it finds a maximum size match. Each iteration consists of three steps:
  Step 1. *Request*. Each unmatched input sends a request to every output for which it has a queued cell.
  Step 2. *Grant*. If an unmatched output receives any requests, it grants one by randomly selecting from all requests.
  Step 3. *Accept*. If an input receives more than one grant, it accepts one by random selection.
  The PIM algorithm faces some problems. First, randomness is difficult and expensive to implement at high speed. Each arbiter must make a random selection among the members of a time-varying set. Second, when the switch is oversubscribed, PIM can lead to unfairness between connections. Finally, PIM does not perform well for a single iteration: it limits the throughput to approximately 63%, only slightly higher than a FIFO switch.

- *Round-robin scheduling*: All existing round-robin scheduling algorithms have the same three steps as PIM. Instead of randomly matching cells, the input and output arbiter adopts a round-robin scheme where the inputs and outputs take turns. Round-robin scheduling overcomes two problems in PIM: complexity and unfairness. Implemented as priority encoders, the round-robin arbiters are much simpler and can perform faster than random arbiters. The rotating scheme makes the algorithm assign bandwidth equally and more fairly among requests.

  – iSLIP: One iteration of iSLIP [2] consists of three steps:
    Step 1. *Request*. Each unmatched input sends a request to every output for which it has a queued cell.
    Step 2. *Grant*. If an output receives any requests, it chooses the one that appears next in a fixed, round-robin schedule starting from the highest priority element. The output notifies each input whether or not its request was granted. The pointer to the highest priority element of the round-robin schedule is incremented (modulo N) to one location beyond the granted input if and only if the grant is accepted in Step 3. If no request is received, the pointer stays unchanged.
    Step 3. *Accept*. If an input receives a grant, it accepts the one that appears next in a fixed round-robin schedule starting from the highest priority element. The pointer to the highest priority element of the round-robin schedule

is incremented (modulo N) to one location beyond the accepted one. If no grant is received, the pointer stays unchanged. iSLIP updates the grant pointers only when the grant is accepted. In this scheme, starvation is avoided.

– FIRM: FCFS in round-robin matching (FIRM) [16] achieves improved fairness (as it approximates FCFS) and has tighter service guarantee than iSLIP. The only difference between iSLIP and FIRM lies in Step 2. FIRM moves the grant pointer regardless of whether the grant is accepted or not, *i.e.*:
Step 2. *Grant*. If an output receives any requests, it chooses the one that appears next in a fixed, round-robin schedule starting from the highest priority element. The output notifies each input whether or not its request was granted. The pointer to the highest priority element of the round-robin schedule is incremented (modulo N) to one location beyond the granted input if and only if the grant is accepted in Step 3. If the grant is not accepted, the pointer is placed to the granted input. If no request is received, the pointer stays unchanged.

The modification of the grant pointer results in enhanced fairness in terms of FCFS service, *i.e.* FIRM approximates FCFS closer than iSLIP with the use of the round-robin pointers. FIRM achieves this as it forces the scheduler to issue the next grant to the unsuccessful request until the grant is accepted while iSLIP may take the newly arrived request first.

Randomized Scheduling Algorithms

The motivation for proposing randomized algorithms is to overcome the complexity of MWM algorithms and to achieve stability under any admissible traffic.

- TASS: TASS [17] is the basic randomized algorithm proposed by Tassiulus. The steps for this algorithm are as follows:
(a) Let $S(t)$ be the schedule used at time t.
(b) At time $t + 1$ choose a match $R(t + 1)$ uniformly at random from the set of all $N!$ possible matches.
(c) Let $S(t + 1) = arg \max < S, Q(t + 1) >$ ($Q(t + 1)$ is the queue-lengths matrix at time $t + 1$.)
It was proven that if the probability of $R(t + 1)$ being equivalent to the maximum weight matching is lower bounded by some constant c, the algorithm is stable. Although TASS achieves stability under any admissible traffic, it does not have good delay performance, especially under non-uniform traffic. Recently, a group of randomized algorithms including APSARA, LAURA, and SERENA were proposed [18]. The motivation behind these algorithms is to improve the delay performance by exploiting some features of the switching problem while maintaining stability.

- APSARA: APSARA [18] selects a match for the current time slot based on three inputs: the match of the last time slot $S(t)$, neighbors of the match $S(t)$, and a randomly chosen match $Z(t+1)$. A neighbor of a match S, denoted as $N(S(t))$, is obtained by exchanging the input/output pair of two connections in the match. The weight is then computed from these inputs and chosen as the maximum match time at $t + 1$. The neighbor is used here with the objective of searching the space matches in parallel.

- LAURA: LAURA [18] differs from APSARA in that LAURA uses a non-uniform random sampling which has the purpose of keeping heavy edges in the matching procedure and a corresponding merging procedure for weight augmentation. By merging the matches of the randomly chosen $Z(t+1)$ with the match of last time slot $S(t)$, a match with increased weight is obtained. Notice that when compared with TASS, 'merge' is used instead of 'max' due to the performance improvement obtained with the merging algorithms.

- SERENA: SERENA [18] makes some modifications to LAURA where it uses the arrival information in the merging procedure. The arrival graph, $A(t+1)$, is one component in the merging procedure and is directly used when it is a match. Otherwise, modification to $A(t+1)$ is required in order to yield a match. This information is used based on the observation that the accumulation of queue lengths is due to arrival events, which also serve as a source of randomness.

Performance Comparison among Algorithms

MWM algorithms perform very well in terms of both delay and stability. However, the computation complexity required to implement them is generally too high to be practical. The more practical approximating MSM algorithms perform well under uniform traffic, but are not stable under non-uniform traffic. Randomized algorithms are linear in complexity and provide the benefit of being stable under any admissible traffic as well. However, the delay encountered is higher than that of approximating MSM algorithms, as randomized algorithms have been designed with objectives of stability rather than small average delay. Figure 1.3 shows the average delay under uniform traffic for these typical algorithms. Figure 1.4 compares the average delay of the same algorithms under diagonal traffic.

1.2.5 Combined Input–Output-queued Switches

VOQs and an increase in the internal speedup of a switch are used to solve the HoL blocking problem. If the switch fabric has an internal speedup, *i.e.* a few times faster than the line rate, buffers are required at both input and output sides. This is a combined input–output queued (CIOQ) switch, which can emulate an OQ switch with a small speedup. There are two emulation schemes: emulating an OQ switch on throughput and emulating both throughput and cell delivery order.

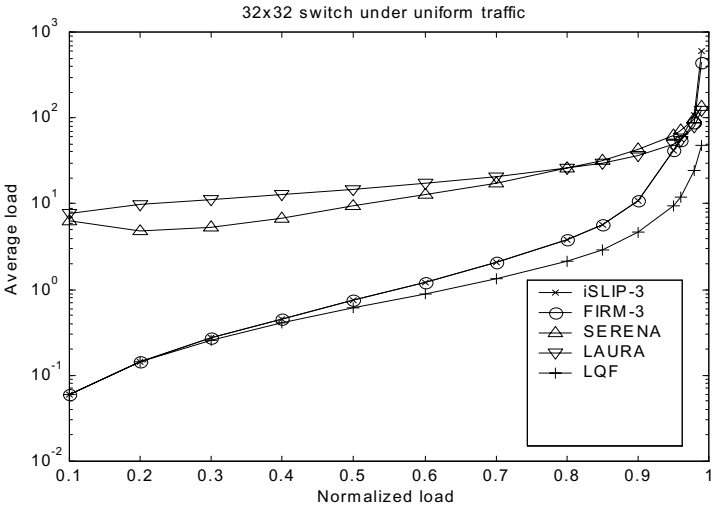


Figure 1.3. Average delay performance for iSLIP (run with three iterations), FIRM (run with three iterations, SERENA, LAURA and LQF under uniform traffic

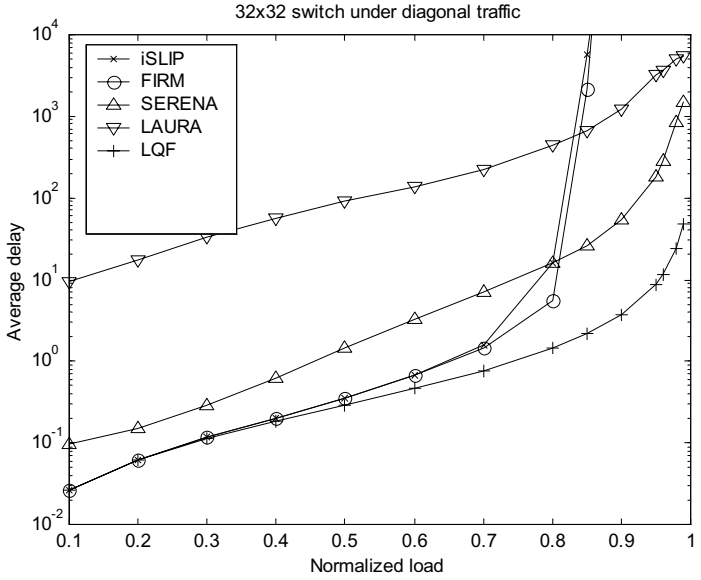


Figure 1.4. Average delay performance for iSLIP, FIRM, SERENA, LAURA and LQF under diagonal traffic

- Exact Emulation of an OQ Switch

  Consider an OQ switch whose output buffers are FIFO. A CIOQ switch is said to behave identically to an OQ switch if, under identical inputs, the departure time of every cell from both switches is identical. This is also called the exact emulation of the OQ switch.
  It was first formulated by Prabhakar and Mckeown in [10] that a CIOQ switch with a speedup of 4 can behave identically to a FIFO-OQ switch. This result holds for switches with an arbitrary number of ports and for any traffic arrival pattern. Their proof is based on the scheduling algorithm called Most Urgent Cell First (MUCFA). The urgency value of a cell used in the MUCFA is defined as the distance it clones from the head of the output buffer in the shadow switch. The cells in any output buffer of the CIOQ switch are arranged in increasing order of urgencies, with the most urgent cell at the head. Once a cell is forwarded to its output in the CIOQ switch, its position is determined by its urgency. MUCFA works as follows:
  (1) At the beginning of each phase, each output obtains its most urgent cells from the inputs.
  (2) If more than one output requests an input, then the input will grant the output whose cell has the smallest urgency number. Ties are broken by the smallest port number.
  (3) Each unmatched output obtains its next most urgent cell from another unmatched input. Then go to step 2.
  (4) When the matching of inputs and outputs is no longer possible, cells are transferred and MUCFA goes to step 1 for the next phase.
  The way in which MUCFA matches inputs and outputs is a variation of the stable marriage problem, which was first introduced by Gale and Shapley and can be solved by a well-known algorithm called GSA. GSA can find a stable match in N^2 iterations [19]. MUCFA is extended to achieve the exact emulation by a speedup of only 2 by a new algorithm called Joined Preferred Matching (JPM) algorithm [20]. Similarly, Critical Cell First (CCF) has been proposed to emulate an OQ switch with speedup 2 and with different output scheduling disciplines [10]. There are two main disadvantages for those GSA-based algorithms mentioned above. First, the stable match in each phase can take as many as N^2 iterations. Second, the algorithms have a high information complexity: they need to maintain a large amount information which is not locally available. The DTC strategy is proposed to reduce the number of iterations and the GBVOQ algorithm is proposed to avoid using global information [10]. However, these two solutions cannot be combined.

- Work-conserving

  A switch is work-conserving if and only if each output is active at the end of the time slot T, where there are cells (either at input or at the output buffers) at the beginning of that time slot. A work-conserving switch provides the same

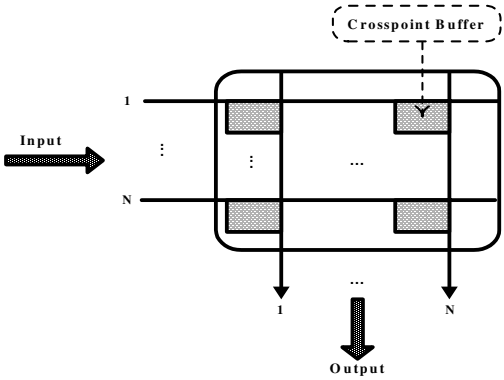


Figure 1.5. Pure buffered crossbar architecture

throughput performance as an OQ switch. The Lowest Occupancy Output First Algorithm (LOOFA) has been proposed in [21] with the work-conserving property in a CIOQ switch with a speedup of 2. An integer variable, occupancy, is associated with each output queue in LOOFA. The occupancy of an input j at any time is simply the number of cells currently residing in output j's buffer.
(1) Each unmatched input selects the non-empty VOQ going to an unmatched output with the lowest occupancy and sends a request to that output.
(2) The output, upon receiving requests from multiple inputs, selects one and sends a grant to that input.
(3) The switch returns to step 1 until no more matches can be made.
This algorithm essentially gives priority to output channels with low occupancy, thereby attempting to simultaneously maintain work conservation across all output channels. The work conserving feature of the switch is independent of the selection algorithm used at the outputs.

1.3 Buffered Crossbar Switches

1.3.1 Buffered Crossbar Switches Overview

For many years, buffered crossbar switches have been considered a viable solution to improve the switching throughput as an alternative to bufferless crossbar switches. Buffered crossbar switches have been studied for at least two decades [4, 22, 23]. In an architecture called *pure* buffered crossbar, as shown in Figure 1.5, buffering occurs *exclusively* at the crosspoints and is utilized to minimize cell loss. The number of ports is limited by the memory amount that can be implemented in a module chip. An example of this architecture was proposed in [4], where a 2×2 switch module with a crosspoint memory of 16 Kbytes each was implemented.

Unfortunately, at that time, it was not possible to embed enough buffering on-chip and therefore this architecture was unable to comply with the required cell

loss rate. In order to overcome the excessive on-chip memory requirement, buffered crossbar switches with input queues were proposed [23, 24]. While there have been many studies analyzing the performance of input FIFO based buffered crossbar switches, a recent result proved that a buffered crossbar employing FIFO queues as its input queuing can achieve 100% throughput under uniform traffic arrivals [25]. As with traditional IQ switches, buffered crossbar switches with input VOQs were researched in order to improve the switching performance.

Buffered crossbar switches with input VOQs (denoted VOQ/BCS) have recently received a lot of interest from both research and industrial communities. Numerous research groups have studied the VOQ/BCS architecture [3, 5, 26]. A representative example from industry that implemented VOQ/BCS is [27]. The first fixed-size-cell VOQ/BCS architecture was introduced in [3] and was shown to outperform conventional IQ switching performance. A Combined Input-One-cell-Crosspoint Buffered Switch (CIXB-1) with VOQs at the input employing a round-robin arbitration scheme was proposed in [28], and was shown to achieve 100% throughput under uniform traffic. The same architecture was extended in [29] to support more than a one cell internally buffered crossbar with input VOQs along with round-robin arbitration. This architecture shows improvement over that of [28] by achieving 100% under uniform traffic as well as non-uniform traffic. VOQ/BCS architecture operating on variable length packets has been studied and was introduced in [30, 31]. When directly operating on variable length packets, a VOQ/BCS no longer requires a segmentation and reassembly (SAR) circuitry and no speedup to compensate for it [31].

VOQ/BCS have also been studied in an attempt to emulate OQ switching. Very interesting results have been proposed showing that a VOQ/IBC, running twice as fast as the line rate, can emulate a FIFO OQ switch [32]. Other results [33], proved that a VOQ/IBC with a speedup of two can emulate a broad class of algorithms (*i.e.*, FCFS, Strict Priority, Early Deadline First) operating on an OQ switch. A more recent and extended result [26] showed that a VOQ/IBC switch with two times speedup can provide 100% throughput, rate and delay guarantees.

As will be shown, the VOQ/BCS has many advantages that alleviate the scheduling task and make it simple. The increasing demand for terabit switches and routers means that future commercial packet switches must be implemented with reduced scheduling complexity, and buffered crossbar architecture, inherently, can implement distributed scheduling schemes. In fact, buffered crossbar switches have been considered as a viable alternative to bufferless crossbar switches to improve the switching performance.

1.3.2 The VOQ/BCS Architecture

In this section, we present the buffered crossbar architecture (VOQ/BCS) using the notation and terminology that will be referred to in the rest of this chapter. We present the switch model and its components such as inputs, internally buffered crossbar fabric, and outputs. Then, we present the three steps that a scheduling cycle consists of: input scheduling, output scheduling, and delivery notification.

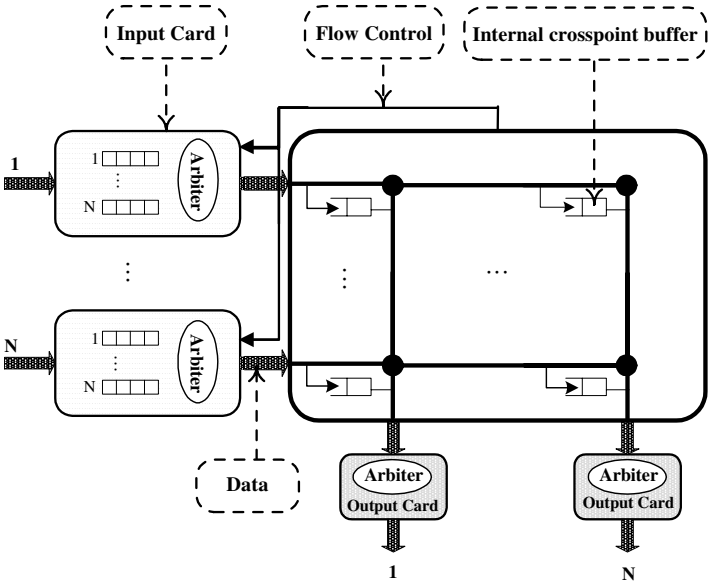


Figure 1.6. The VOQ/BCS architecture

Architecture and Terminology

As shown in Figure 1.6, an $N \times N$ VOQ/BCS switch architecture consists of N input cards, with each maintaining N VOQs, one for each output. The buffered crossbar fabric component is the main characteristic of the VOQ/BCS that differentiates it from the conventional IQ/VOQ architecture, where small buffers are added per crosspoint. Fixed size packets, or cells, are considered. Upon their arrival at the switch, variable length packets are segmented into cells for internal processing and re-assembled before they leave the switch. A processing cycle has a fixed length, called cell or time slot. The architecture can be divided into three major parts and described as follows:

- **Input Cards:** There are N input cards; each one maintains N logically separated VOQs. When a packet destined to output j, $0 \leq j \leq N-1$, arrives at the input card i, $0 \leq i \leq N-1$, it is held in VOQ_{ij}. A VOQ_{ij} is said to be eligible for being scheduled in the input scheduling process if it is not empty and the internal buffer XP_{ij} is empty.
- **Buffered Fabric:** The internal fabric consists of N^2 buffered crosspoints (XP). Each crosspoint has a one-cell buffer. A crosspoint XP_{ij}, holds cells coming from input i and going to output j.
  XPB_i is the set of the internal buffers for all outputs j ($XP_{i0}+\cdots+XP_{i(N-1)}$).
  XPB_j is the set of the internal buffers for all inputs i ($XP_{0j}+\cdots+XP_{(N-1)j}$).
- **Flow Control:** Between each two time slots a flow control mechanism is performed for information exchange between the crosspoints and the input cards.

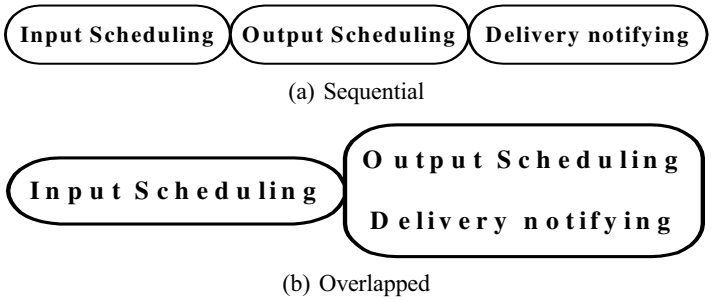


Figure 1.7. Scheduling cycle for the VOQ/BCS architecture

Each crosspoint XP_{ij} tells its corresponding input card i whether or not it is ready to receive a cell during the next time slot.

It is of note that the arbiters at the inputs and the outputs are totally distributed. There are as many arbiters as input ports, and the same applies for the outputs. The arbitration made by any arbiter does not depend on the other arbiter's decisions. At every input port, all an arbiter i needs to maintain is the state of the VOQs belonging to its port i and the internal buffer's XPB_i state corresponding to these VOQs. However, for each output arbiter, it is even simpler. All an output arbiter j needs to maintain is the occupancy of its corresponding internal buffers XPB_j.

Scheduling Process

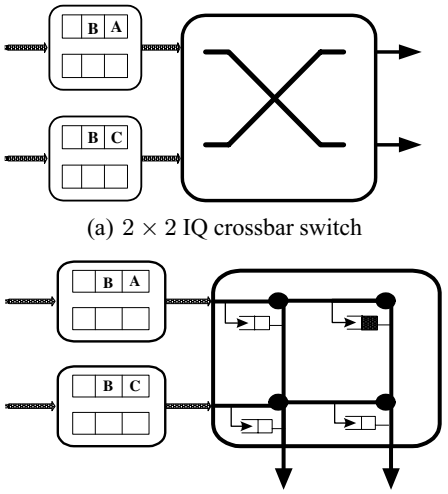
With the structure described above and the corresponding notation, a scheduling cycle consists of three steps:

1. **Input scheduling:** each input selects one cell, in a predefined manner, from the HoL of an eligible VOQ.
2. **Output scheduling:** each output selects one cell, in a predefined manner, from all the internally buffered cells in the crossbar to be delivered to the output port.
3. **Delivery notification:** for each delivered cell, inform the corresponding input of the internal buffer status; that is, change the status of the corresponding VOQ to be eligible.

Figure 7(a) illustrates a scheduling cycle consisting of the three above-mentioned phases. The input scheduling is performed first, followed by the output scheduling, and finally the delivery notifying. It is of note that for fast implementation purposes, the output scheduling and the delivery notifying steps can be overlapped, as shown in Figure 7(b).

Features of the VOQ/BCS Architecture

This section presents the features that the VOQ/BCS offers. We will present these characteristics in terms of comparison with the input queued and the shared memory



(a) 2×2 IQ crossbar switch

(b) 2×2 buffered crossbar switch

Figure 1.8. Architecture comparison of bufferless and buffered crossbar

architecture. Compared to the existing architectures, the VOQ/BCS architecture has substantial advantages. They can be summarized as follows:

- **Better performance:** In the IQ/VOQ architecture, a scheduling algorithm makes a match between inputs and outputs. An input can select an output provided that this output is not occupied by other inputs. This matching mechanism is usually done through the request–grant–accept phases. As was shown in [34], the decision time spent by the grant and the accept arbiters takes approximately 75% of the whole iteration time. This is due to high information exchange between the input and output schedulers, which enforces a high dependency between them, hence making their design complex and centralized. While in the VOQ/BCS architecture, if an output is not ready to receive the cell, the input can still transmit it to an internal buffer provided the internal buffer is empty. In other words, the choices of inputs and the choices of outputs needn't be synchronous. This can entirely resolve the problem of output contention, since an input can send to more than one output (via the internal buffer) and likewise, more than one output can receive cells coming from the same input. Figure 8(b) shows an example of output contention resolution. Adopting the VOQ/BCS architecture, the input schedulers and the output schedulers are totally distributed. There is no information exchange among them at all. This in turn, reduces the queuing delay and increases the throughput, hence improving dramatically the overall performance of the switch.
  The example shows a 2×2 IQ switch, Figure 8(a), and a buffered crossbar switch, Figure 8(b). In the first case, Figure 8(a), the two HoL cells (A and C) have

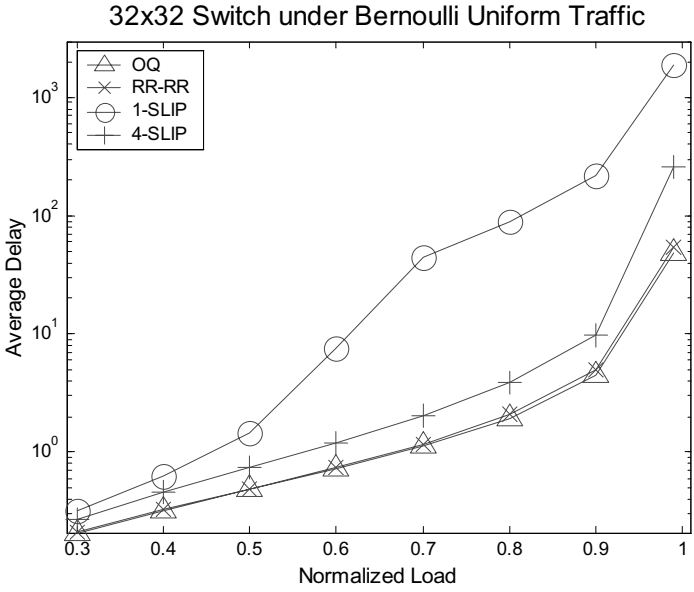


Figure 1.9. Delay performance under Bernoulli IID uniform traffic

the same destination output, and one of them can be forwarded because of the output contention. However, in Figure 8(b), the same input HoL cells A and C can be transmitted, simultaneously, to the internal buffers, since only $IB_{1,2}$ is full. Therefore the output contention problem does not occur at all for the buffered crossbar case, so long as the internal buffer is free.

To show the far better performance a VOQ/BCS exhibits when compared to a bufferless architecture, we plot the performance study of [28]. A 32×32 buffered crossbar switch, using a simple round-robin arbitration scheme, was simulated under Bernoulli IID uniform traffic and compared to that of a bufferless architecture using iSLIP with one and four iterations, as in Figure 1.9.

- **Higher speed and simpler implementation:** Many scheduling algorithms for the IQ/VOQ architecture are iterative (*e.g.* iSLIP, FIRM). However, the performance of a single iteration is often not good enough. For iSLIP, usually four iterations are employed. These additional iterations result in additional complexity. For this architecture, we just run the above three steps. Input scheduling and output scheduling can be totally independent, reducing largely their arbitration complexity, which is linear on the input ports number N.
- **Lower hardware requirement:** With shared memory, a speedup of N is required to transfer all the incoming cells simultaneously to their outgoing ports. However, for the VOQ/BCS architecture, each internal buffer is separated from one another. An internal buffer will be accessed at most two times: once by an input and once by an output.

However, for a large number of inputs/outputs, the number of internal buffers becomes huge (since it equals N^2). Even though VLSI density increases make it possible to embed enough memory on chip, it can still be quite difficult to put so much memory on a crossbar. One way to make the implementation easier is by moving the internal buffers out of the crossbar and putting them at the input side of the chip and adding two multiplexers. The first multiplexer at each input chooses one among the N VOQs and sends the HoL packet to the internal buffers, and the second multiplexer chooses one internal buffer according to the packet's destination output in which to put the received packet. Another implementation variant is putting the internal buffers at the output side instead of the input. The multiplexer at an output chooses one internal buffer to receive a packet and then sends it out. It is quite clear that the two variants are equivalent to the original architecture. We believe that these two alternative layouts are less complicated and more scalable than the conventional one.

Scheduling Cells in the VOQ/BCS Architecture

As bufferless scheduling algorithms reached their practical limitations due to higher port numbers and data rates, buffered crossbars received increased interest because they have the potential to solve the complexity and scalability issues faced by their bufferless predecessors. The increasing need for terabit switches and routers means that future commercial packet switches must be implemented with reduced scheduling complexity, and buffered crossbar architectures, inherently, can implement distributed scheduling schemes.

In the past few years, many scheduling schemes have been proposed for the VOQ/BCS architecture. The first, [3], used input and output arbitration schemes based on the *Oldest Cell First*. The second scheme presented was based on *round-robin* arbitration in both input and output scheduling [28]. While these two schemes were proven to achieve 100% throughput under uniform traffic, they performed rather poorly under non-uniform traffic patterns. To overcome this, a scheme based on *Longest Queue First* (LQF) input scheduling followed by a round-robin arbitration scheme in the output side was proposed and demonstrated 100% throughput under uniform traffic patterns.

It is of note that the algorithms presented above were compared with bufferless schemes and were demonstrated, as expected, to have much better performance. However, as mentioned earlier, the VOQ/BCS has key advantages that ensure the scheduling algorithm is simple and efficient at the same time. So far, most existing algorithms (OCF, RR, and LQF) are just simple mappings of previously proposed algorithms for bufferless crossbar switches into the new VOQ/BCS architecture.

The presence of internal buffers significantly improves the overall performance of the switch due to the advantages it offers. A scheme that takes full advantage of the internal buffers was recently presented in [35]. This scheme was, in fact, an approximation of the *First Come First Served* (FCFS) policy. It was based on the *Currently Arrival First* (CAF) cell in the input scheduling followed by a priority

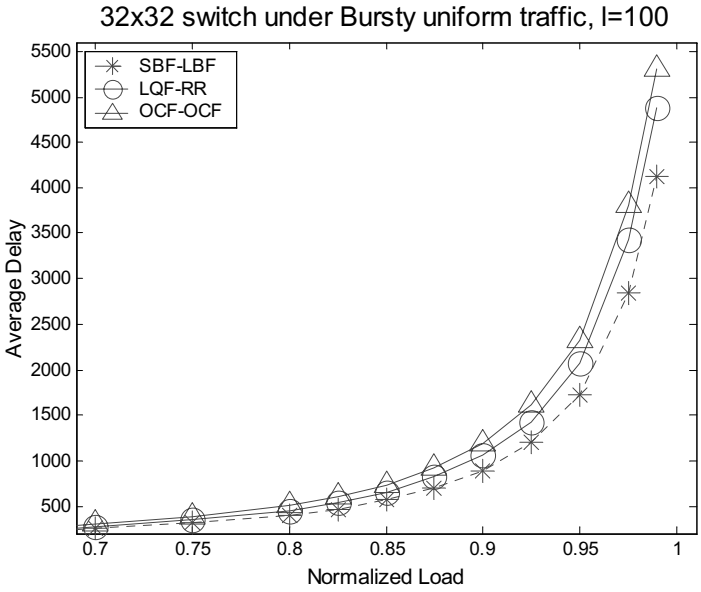


Figure 1.10. Delay performance under bursty uniform traffic, l =100

scheme called *Priority ReMoVal* (PRMV). A scheme that exclusively used the internal buffers was also proposed [36]. This scheme is called the *Most Critical internal Buffer First (*MCBF), and is based on the *Shortest internal Buffer First* (SBF) at the input scheduling and on the *Longest internal Buffer First* (LBF) at the output side. The authors addressed the important role that the internal buffer element plays in the scheduling process due to its shared nature. While being a stateless scheme, MCBF outperforms weighted scheduling schemes such as LQF and OCF. Figure 1.10 shows the delay performance of MCBF, LQF-RR and OCF-OCF under uniform burst traffic with burst length equal to 100.

1.4 Multi-stage Switching

1.4.1 Architecture Choice

Clos-network switch architectures can be categorized into two types. The first type has buffers in the second stage, such as the WUGS architecture in [6]. The function of the buffers is to resolve contention among cells from different first-stage modules. However, cells may be mis-sequenced at the output ports, requiring a re-sequence function, which is difficult to implement when the port speed increases. The second type of architecture has no buffers in the second stage and uses shared memory modules in the first and last stages to aggregate cells. The ATLANTA switch with

its Memory/Space/Memory (MSM) architecture is a commercially successful example [7]. This architecture is more promising as no mis-sequence problem exists. We describe several dispatching algorithms for the MSM architecture, including concurrent dispatching (CD), concurrent round-robin dispatching (CRRD), concurrent master–slave round-robin dispatching (CMSD), and concurrent static round-robin dispatching (SRRD).

However, one disadvantage of the MSM architecture is that the input and output stages are both composed of shared-memory modules, introducing a memory speedup problem. Although the speedup is smaller than that in output-queued switches, it definitely hinders the switch's ability to scale up to very large port numbers.

The memory speedup problem was solved by the bufferless Clos-network architecture proposed in [37]. This architecture contains only crossbar switching elements in all stages. All cells are stored in the input port cards, as done with the virtual output queuing structure in single-stage crossbar switches. Since the switching elements are fully distributed by smaller modules, this raises the challenge of how to design the scheduling algorithm in a fully distributed way. We then describe the Distro dispatching algorithm for the bufferless Clos-network architecture.

1.4.2 The MSM Clos-network Architecture

Switch Model

The switch architecture used in this paper is based on [37] and is shown in Figure 1.11. The input and output stages are both composed of shared-memory modules, each with n port interfaces. They are fully interconnected through a central stage that consists of bufferless crossbars of size $k \times k$. In the switch, there are k input modules (IM), m central modules (CM), and k output modules (OM).

An OM(j) has n buffered output ports, OP(j,h). Each output port buffer can receive at most m cells from m central modules and send at most one cell to the output line in one time slot.

An IM(i) has nk virtual output queues, VOQ(i, j, h), for storing cells that go from IM(i) to OP(j, h) at OM(j). Each virtual output queue can receive at most n cells from n input ports and send one cell to the central module. We use VOQ Group (i, j) to represent all VOQs storing cells from IM(i) to OM(j).

An IM(i) has m output links, LI(i, r), connecting to each CM(r). An CM(r) has k output links, LC(r, j), connecting to each OM(j).

Concurrent Dispatching (CD)

The distributed architecture implies the presence of multiple contention points in the switch. The ATLANTA switch proposed the CD algorithm with highly distributed nature [7, 38]. It works as follows.

In each time slot, each IM randomly selects up to m VOQs and randomly sends the requests to CMs. If there is more than one request for the same output link in

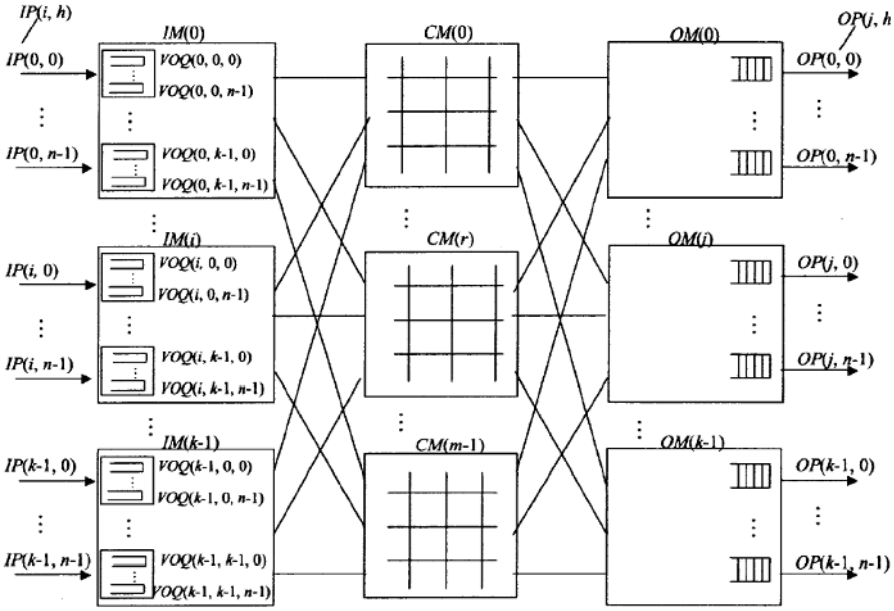


Figure 1.11. The MSM Clos-Network model

a CM, it grants one request randomly. Finally, the granted VOQs send to the corresponding OP in the next time slot.

However, the CD algorithm cannot achieve a high throughput unless the internal bandwidth is expanded. The original CD algorithm applies a backpressure mechanism in the dispatching process. We describe only its basic concept and characteristic in this paper. We also assume that the buffer size in the IMs and OMs is large enough to avoid cell loss. Hence we can focus the discussion on the properties of the dispatching algorithms.

Concurrent Round-robin Dispatching (CRRD)

In crossbar switches, round-robin arbitration has been developed to overcome the throughput limitation of the PIM algorithm. Similarly, the CRRD algorithm has been proposed in [39] to overcome the throughput limitation of the CD algorithm by using round-robin arbiters. It is based on the request-grant-accept (RGA) handshaking scheme, just as in the iSLIP algorithm. Since contention occurs in both output links of the IMs and CMs, two phases must be employed to resolve the contentions. In Phase 1, the algorithm employs iterative matching between VOQs and output links in each IM. Phase 2 then performs contention control for the output links in the CMs.

Phase 1: Iteractively Matching within IM:

- *Step 1 Request*: Each unmatched, non-empty VOQ(i, j, h) sends a request to every output link LI(i, r).
- *Step 2 Grant*: Each output link LI(i, r) selects one request with the round-robin arbiter and sends the grant to the selected VOQ.
- *Step 3 Accept*: Each VOQ V(i, j, h) selects one grant with the round-robin arbiter and sends an acceptance notice to the selected output link LI(i, r).

Phase 2: Matching between IM and CM:

- *Step 1 Request*: Each IM output link LI(i, r), which was accepted by VOQ(i, j, h) in Phase 1, sends the request to the CM output link LC(r, j).
- *Step 2 Grant*: Each CM output link selects one request with the round-robin arbiter. It then sends the grant to the selected IM.
- *Step 3 Accept*: If the IM receives the grant from the CM, it sends the head cell from the matched VOQs in the next time slot.

Note that CRRD uses three sets of round-robin arbiters to resolve the contentions in IMs and CMs. As with iSLIP, the round-robin pointer in the arbiters is updated to one position after the selected position if and only if the match within the IM is achieved in Phase 1 and the request is granted by the CM in Phase 2. The desynchronization effect of the round-robin pointers in CRRD works exactly as in iSLIP. As a result, CRRD provides 100% throughput under uniform traffic and burst traffic independent of the number of iterations in Phase 1.

Concurrent Master/Slave Round-robin Dispatching (CMSD)

The CMSD is an improved version of the CRRD. It employs two sets of arbiters in the IM, a master and a slave. They operate concurrently in a hierarchal round-robin manner. The CMSD differs from the CRRD only in the iterative matching process in Phase 1.

Phase 1: Iteractively Matching within IM:

- *Step 1 Request*: Each unmatched, non-empty VOQ(i, j, h) sends a request to the jth slave arbiters in every output link arbiter LI(i, r). At same time, each VOQ Group (i, j) that has at least one unmatched, non-empty VOQ sends a request to the master arbiter in every output link LI(i, r).
- *Step 2 Grant*: Each slave arbiter and master arbiter simultaneously selects the request in a round-robin fashion. At same time, each master arbiter selects one VOQ Group's request in a round-robin fashion. Finally, LI(i, r) sends the grant to VOQ(i, j, h) if and only if j has been selected by the master arbiter and h has been selected the jth slave arbiter.
- *Step 3 Accept*: Each VOQ(i, j, h) selects one grant in a round-robin fashion and sends an acceptance notice to the selected output link LI(i, r).

CMSD provides enhanced scalability while preserving the advantages of CRRD. In CRRD, each arbiter in LI(i, r) should make the decision among nk requests. In CMSD, master arbiters need only consider k requests and slave arbiters n requests. Since all arbiters can operate simultaneously, this will considerably reduce the arbitration time.

Concurrent Static Round-robin Dispatching (SRRD)

The Static Round-robin (SRR) scheme has been demonstrated to give very good delay performance in crossbar switches. The key idea is to desynchronize the pointers in the round-robin arbiters in a static way and to use a rotating-search technique to improve the performance under non-uniform traffic. Intuitively, we can apply the SRR technique into dispatching processes in the Clos-network switching system. The Static Round-robin Dispatching (SRRD) scheme was proposed in [40].

SRRD is exactly the same as CMSD except in its method for updating the round-robin pointers. The novelty of our SRRD scheme is the systematic initialization and intelligent updating of the pointers. All pointers are artificially set to be desynchronized to efficiently resolve the contentions in each time slot, and it is guaranteed that no VOQ will be starved in every N time slots.

Performance Evaluation

We compare the delay performance of the dispatching algorithms in the MSM architecture and in the single-stage crossbar switch architecture under uniform traffic. We use the Clos-network setting $m = n = k = 8$, which corresponds to a port size of $N = 64$ in the crossbar switch.

As shown in Figure 1.12, the average delay of the algorithms in the MSM architecture is larger than those in the crossbar switch when load is below 0.5. This is because most of the dispatching algorithms in the crossbar switch perform fairly efficiently under low traffic load. The MSM architecture introduces more contention points in the distributed modules, however, the performance of the architecture is improved significantly in the heavy load region, as shared memory modules are effectively used.

The randomness-base algorithms, PIM and CD, could only achieve about 60% throughput. All remaining round-robin algorithms can achieve 100% throughput under uniform traffic. SRRD has the lowest average delay and is much closer to the performance of an output-queued switch. This result shows the significant effect of the SRR technique in the MSM architecture.

1.4.3 The Bufferless Clos-network Architecture

Switch Model

One disadvantage of the MSM architecture is that the input and output stages are both composed of shared-memory modules. This results in a memory speedup of n

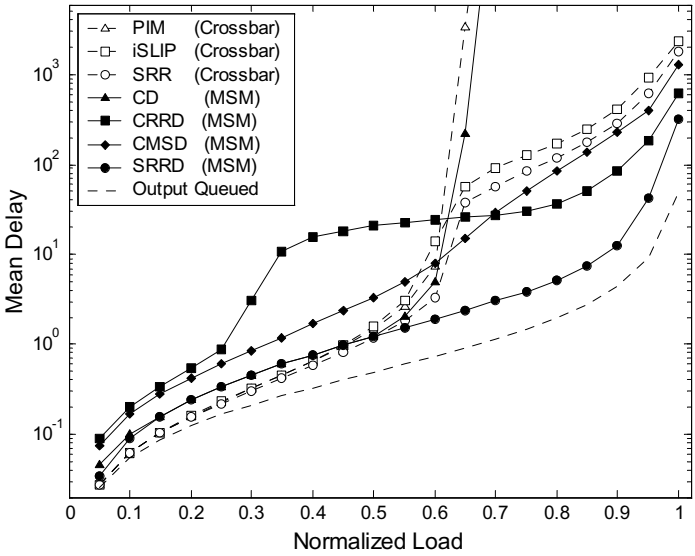


Figure 1.12. Delay comparison of crossbar and MSM

in each IM and k in each OM. In output-queued switches, the memory speedup is $N = n \times k$. Although the speedup of MSM is smaller than that in output-queued switches, it definitely hinders a switch's ability to scale up to very large port numbers. In [37], a bufferless Clos-network switching architecture has been proposed.

As depicted in Figure 1.13, the bufferless Clos-network architecture is slightly modified from the MSM architecture by replacing all shared memory modules by crossbars. All cells are stored in the input port cards, the same as the virtual output queuing structure in single-stage crossbar switches.

An IP(i, g) has N virtual output queues, VOQ(i, g, j, h), storing cells that go from IP(i, g) to OP(j, h) at OM(j). Each virtual output queue can receive at most one cell and send at most one cell. A VOQ Group (i, g, j) comprises all VOQs from IP(i, g) to OM(j). In this paper, i corresponds to an IM, g to a specific input port of an IM, j corresponds to an OM, and h to a specific output port of an OM.

Distro Dispatching Algorithm

Since contention points exist in all output links of the IPs, IMs, CMs and OMs, scheduling in the bufferless architecture is more challenging than in the MSM architecture. The Distributed Static Round-robin (Distro) dispatching algorithm has been proposed in [37] to dispatching cells in the new architecture. The Distro algorithm is a natural extension from algorithms in the MSM architecture.

The additional two phases are included in Distro because elimination of shared memory modules in the first and third stage introduces two more contention points.

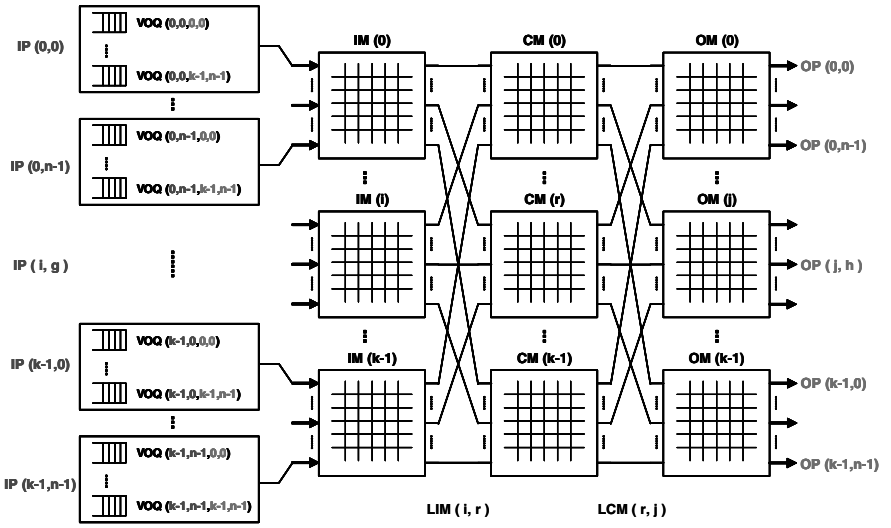


Figure 1.13. The bufferless Clos-network model

Note that the algorithms for the MSM architecture are based on the request-grant-accept (RGA) handshaking scheme. This approach is difficult to implement when too many contention points exist, and therefore the Distro algorithm adopts the request-grant (RG) scheme as proposed in the DRRM algorithm. Similar to SRRD, the Distro algorithm systematically initializes and intelligently updates the pointers. As a result, all pointers are artificially set to be desynchronized to efficiently resolve the contentions in each time slot, and it is guaranteed that no VOQ will be starved in every N time slots.

Each IP(i, g) is associated with Arbiter_j (i, g), VOQ Group in IP(i, g) with Arbiter_h (i, g, j), LI(i, r) with Arbiter_g (i, r), LC(r, j) with Arbiter_i (r, j), and OP(j, h) with Arbiter_r (j, h). All arbiters are making decisions in a round-robin fashion. Round-robin pointers of the highest priority are desynchronized at the start and will be updated systematically to keep the desynchronization in each time slot. The Distro algorithm works as follows:

- **Phase 1 Request selection in each IP(*i*,*g*):** Each Arbiter_j (i, g) selects an non-empty VOQ Group (i, g, j). At the same time, each Arbiter_h (i, g, j) selects a non-empty VOQ(i, g, j, h) within VOQ Group (i, g, j). Then each IP(i, g) sends the request $[j, h]$ to its output link.
- **Phase 2 Grant from LI(*i*,*r*):** Each LI(i, r) systematically chooses the request $[j, h]$ from IP(i, g) and sends the request to LC(r, j).
- **Phase 3 Grant from LC(*r*,*j*):** If LC(r, j) receives one or more non-empty requests from k LIs, it chooses the request $[j, h]$ in LI(i, r) with Arbiter_i (r, j) and sends the request to OP(j, h).
- **Phase 4 Grant from OP(*j*,*h*):** If OP(j, h) receives one or more non-empty requests from k LCs, it chooses the request $[j, h]$ in LC(r, j) with Arbiter_r (j, h).

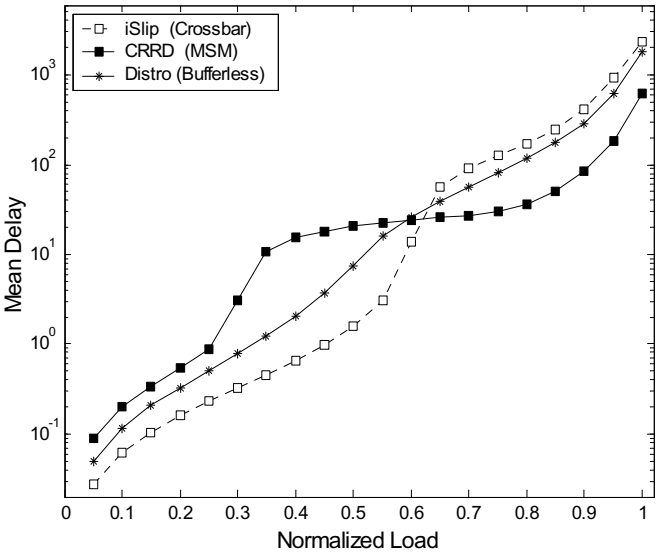


Figure 1.14. Delay comparison with crossbar and multi-stage scheduling algorithms

Finally, OP(j, h) notifies the IP(i, g) via the granted path, and the VOQ (i, g, j, h) will send to OP(j, h) in the next time slot.

As mentioned before, the bufferless Clos-network architecture is very scalable for port size. There are actually many flexibilities in the configurations. We can either scale up the port size by increasing the number of ports n per input/output module, or increasing the number of central modules m. Note that m must be larger than or equal to n in order to achieve the non-blocking property in Clos networks.

Performance Evaluation

We compare the delay performance of our Distro algorithm with other related algorithms in Figure 1.14. It is clear that Distro achieves 100% throughput under uniform traffic. When load is less than 0.65, the Distro algorithm is worse than iSLIP on a reasonable scale. This is due to an increased number of contention points in the Clos-network switches. However, as the load increases, the desynchronization effect of Distro improves the delay performance. In the heavy load region, the performance of Distro closely approximates to the performance of the SRR algorithm.

The delay performance of the MSM algorithms is generally worse than other algorithms in the light load region. Since the MSM algorithms use shared-memory modules to resolve the contention for OPs, their delay performance in the heavy load region are the best when compared with other architectures. This is compensated by the high memory speedup.

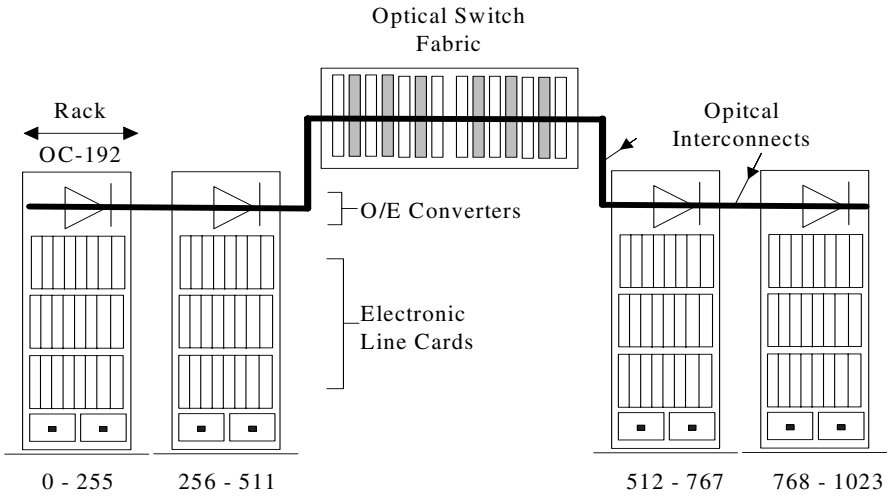


Figure 1.15. Illustration of a multi-rack switching system (1024 × 1024, 4 racks)

1.5 Optical Packet Switching

1.5.1 Multi-rack Hybrid Opto-electronic Switch Architecture

The explosion of Internet traffic has brought about an acute need for broadband communication networks. This heightens the demand for high-performance switches / routers more than ever and terabit switching systems are now receiving much attention. However, such systems with high power consumption and large numbers of ports can no longer be built in a compact, single-rack fashion [41]. Most high-capacity switches currently under development employ a multi-rack system architecture, as shown in Figure 1.15, with the switching fabric in one rack and line cards spread over several racks. Such racks are connected to each other via cables.

Besides the overall architecture there is the choice of hardware switch components. This includes:

- *Packet buffering and processing*: because of the immaturity of optical buffering technology, such as fiber delay lines (FDL) [42], packets are still buffered and processed electronically.
- *Interconnects*: cables replace the backplane in multi-rack systems. Interconnects can be copper or parallel optical cables. However, compared to copper cables, parallel optics are favored for higher bandwidth, lower latency and extended link length beyond copper capabilities [43].
- *Fabric*: The fabric can be electronic or optical. Suppose a traditional electronic fabric is used. In such a system architecture, a series of opto-electronic conversions are needed to switch packets. Line cards terminate high-capacity optical fiber links from the network, and the received packets are processed and buffered electronically. Since the switch is located in an independent rack, fiber

links are used to transfer packets between the line cards and the switch fabric. It is clear from the above description that the system introduces an additional layer of opto-electronic (O/E) conversions between the interconnect and the switch fabric, which results in high complexity and significantly higher cost. In addition, the central electronic fabric occupies valuable floor space, consumes a lot of power, and poses several reliability problems owing to the large number of high-speed active components.

An obvious evolution to this architecture is to try to replace the central electronic fabric with an optical fabric [44, 45, 46, 47, 48]. Optical switch fabrics may overcome the cost, power, space, and scalability problems that arise in sizing traditional electrical backplanes into the terabit regime.

To this end, an opto-electronic hybrid switch based on the strengths of both electronics and optics is a practical and viable approach. This architecture uses electronics for packet processing/scheduling/buffering and optics for packet transmission and switching.

1.5.2 Optical Fabrics

The basic premise of optical switch fabrics is their high switching capacities and reduction in O/E conversions. These advantages are significant as there is no need for lots of expensive high-speed electronics. Furthermore, optical fabrics are cheaper and smaller in physical size. They may also provide potential benefits including scalability, high bit rate, and low power consumption. The technologies include optical micro-electromechanical systems (MEMS)-based switching [49][50], thermal optical switching [51], electro-optic switching, bubble switches [52], etc. Reference [53] gives a detailed comparison between these optical switching technologies.

We use optical MEMS fabric as an example. The basic switching elements of MEMS are tiny micro-actuated free-rotating mirrors as shown in Figure 1.16. The switching function is performed by reflection of the light. MEMS has several hundred of these tiny mirrors arranged in arrays and integrated on a silicon chip. The two-dimensional (2D) optical MEMS has mirrors arranged in a crossbar configuration [54]. Figure 1.17 shows a photo of a 2D $N \times N$ switch fabric made by AT&T [55]. Collimated light beams propagate parallel to the substrate plane. When a mirror is activated, it moves into the path of the beam and directs the light to one of the outputs by making a 45° angle with the beam. In general, the (i, j) mirror is raised to direct light from the ith input fiber to the jth output fiber. The mirror is no larger in diameter than a human hair. Several hundred such mirrors can be built on a chip no larger than a few centimeters square. Since MEMS create so many mirrors on a single chip, the cost per switching element is relatively low. Therefore, MEMS allow low-loss large-port-count optical switching solutions at very low cost per port.

Although there are many advantages to using optical fabrics as mentioned before, these technologies are still emerging and usually exist in sub-optimal form today. The reconfiguration times (or reconfiguration delay) of optical fabrics are much longer than those of electronic fabrics. For example, a MEMS-based optical fabric needs

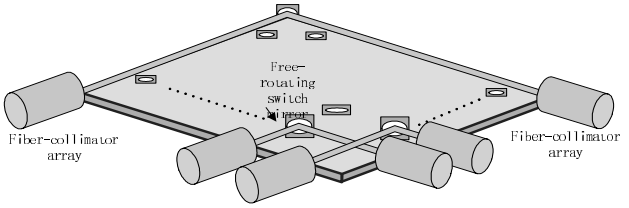


Figure 1.16. Illustration of a 2D MEMS approach to construction of the optical switch fabric

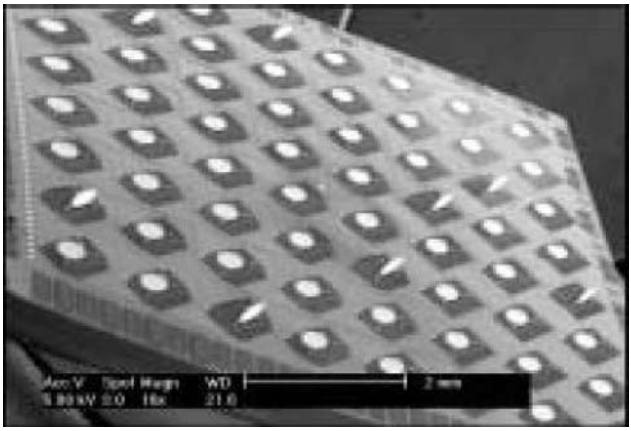


Figure 1.17. 2D $N \times N$ switch fabric demonstrated by AT&T

Table 1.1. Reconfiguration delay times for some optical switching technologies

Switching technology		Delay
Optical MEMS	Mirror/gap-closing electrostatic actuator	7 ms
	1×2 MOEMS switch based on silicon-on-insulator and polymeric waveguides	32–200 ns
Thermal optical switch	Bubble-actuated switch	1 ms
	Fully packaged polymeric four arrayed 2×2 DOS	< 5 ms
Electro-optic switch	Electroholographic (EH) optical switch (1×2)	< 10 ns
	Liquid crystal holographic optical switch (1×8)	ms
	Electronically switchable waveguide Bragg gratings switch (2×2)	10–50 ns

to adjust the angles of the fabric mirrors to set up new connections. This introduces mechanical settling, synchronization, and other time-consuming operations. These operations take times from hundreds of nanoseconds to a few milliseconds [53]. Table 1.1 samples the reconfiguration delay times for several different technologies. This is around 10 to 10^5 time slots for a system with a slot time equal to 50 ns (64 bytes at 10 Gb/s).

1.5.3 Reduced Rate Scheduling

As introduced earlier, the scheduling task in all switches is to find a one-to-one bipartite match between inputs and outputs to switch out the packets, no matter if the fabric is optical or electronic. Generally in each time slot, the electronic switch runs the scheduling, sets up a new fabric connection, and then transfers the corresponding cells.

However, this slot-by-slot approach is not practical for optical switches. For example, if the reconfiguration delay equals one time slot, one cell transmission takes two time slots. One time slot is for scheduling and connection setup (fabric reconfiguration), the second time slot is for actual transmission. In other words, only half of the fabric bandwidth is used for cell transmission (which is effective bandwidth); while the other half is wasted for reconfiguration. For a switch which transfers 64B packets at a 40 Gb/s line rate and suffers a 100 ns reconfiguration delay, the effective bandwidth is as low as 10%. If the reconfiguration delay is not addressed by a proper scheduling scheme, it can severely cripple the performance of optical switches. It is clear that the scheduling rate must be reduced to compensate for the reconfiguration delay. Each schedule holds for some time rather than changing at every time slot.

The reduced rate scheduling is not a simple extension of scheduling algorithms introduced in Section 1.2. It has been proven that the scheduling problem for optical switches with reconfiguration delay is NP-hard [48]. Currently, most of the reduced rate scheduling algorithms use the time slot assignment (TSA) approach [56, 57, 58, 48] and provide heuristic solutions.

1.5.4 Time Slot Assignment (TSA) Approach

The TSA-type of algorithm periodically accumulates incoming traffic and maps this batch to a set of switch configurations. The objective of TSA scheduling in an optical switch is to find a set of fabric configurations (or *schedules*) and their respective holding time, to switch out all of the accumulated traffic, and to maximize the fabric utilization. In other words, this is equivalent to minimizing the total *transmission makespan*, which includes the time spent in transmitting traffic and the time spent in reconfiguring the fabric. This is proved to be NP-hard under non-zero reconfiguration delay [48]. All algorithms introduced here are heuristic.

Scheduling Scheme

Using the TSA scheduling scheme, the switch works in a three-stage accumulation, scheduling, and transmission cycle. The length of the accumulating stage is set to be a predefined system constant T. Incoming traffic in these T time slots are stored in traffic matrix D. For a $N \times N$ switch, $D = (d_{ij})$ is a nonnegative integer $N \times N$ matrix. d_{ij} represents the number of cells received in input i whose destinations are output j during the accumulating stage. The scheduling stage then finds a set of N_s schedules $P_1, P_2, \ldots, P_{N_s}$ for the accumulated traffic. Each schedule will be held for $\phi_1, \phi_2, \ldots, \phi_{N_s}$ time slots respectively. Following the scheduling decision, the

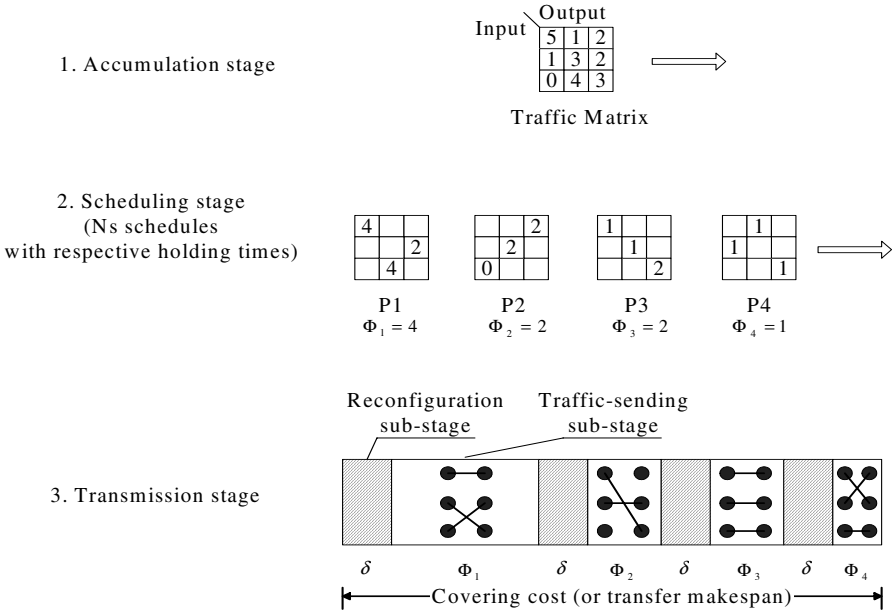


Figure 1.18. TSA scheduling example

transmission stage sets up the fabric and switches out the traffic repeatedly until all packets are sent out. The transmission makespan is equal to $\sum_{k=1}^{N_s} \phi_k + N_s\delta$. Figure 1.18 shows an example of TSA scheduling in a 3×3 switch.

The three-stage TSA operations can be further accelerated by using a pipelining framework as shown in Figure 1.19, with each stage running in T time slots. Notice that since the transmission stage suffers from a $N_s\delta$ reconfiguration delay and potentially empty time slots (that exist when a particular connection is held with no more packets to transmit), speedup S is generally needed to ensures it finishes within T time slots.

TSA algorithms are favored because of their good features:

- **Stability:** because the traffic batch gathered in the accumulation stage is always sent out in the transmission stage, TSA scheduling is stable under any admissible traffic patterns.
- **Bounded cell delay:** the pipeline diagram indicates that a cell will go through the switch within $3T$ slot times. This bounded worst cell delay ($3T$) makes it possible to provide QoS guarantees.

For the required buffer size, it is observed that traffic from at most three different batches may appear in a particular input buffer at the same time. Assume B is the number of bits sent to one input per time slot. A buffer of size $3TB$ is enough for each input and $2TB$ for each output. If all ports are considered, the switch needs at most $5TBN$ bits buffer size.

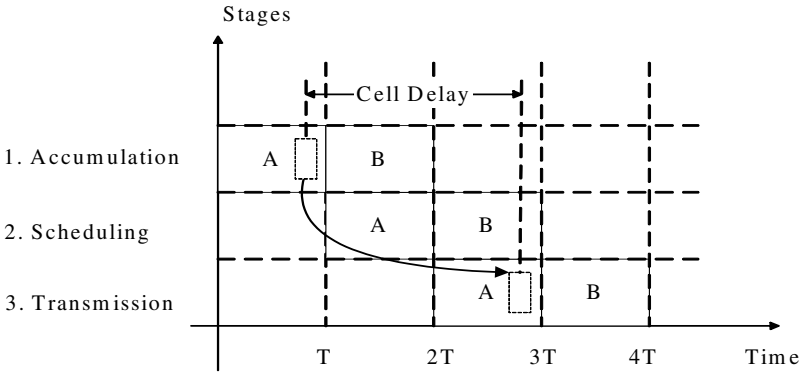


Figure 1.19. Pipelined TSA execution with two traffic batches

1.5.5 DOUBLE Algorithm

Early TSA algorithms assume the reconfiguration delay to be zero [56] or infinite [57] for simplicity. However, the extreme assumptions on reconfiguration delay are no longer practical in optical switching systems. A recent DOUBLE algorithm [58] considers scheduling under moderate reconfiguration delays. It performs better than algorithms with extreme assumptions under a large spectrum of reconfiguration delay values [58].

DOUBLE works by separating the traffic matrix D into *coarse* and *fine* matrices and devotes N configurations to each. Coarse matrix A and fine matrix B are generated in a way that ensures $D \leq \lceil T/N \rceil A + B$. The algorithm first generates the coarse matrix A by dividing the elements of D by T/N and taking the floor. The rows and columns of A sum to at most N (because of the admissible traffic assumption), thus the corresponding bipartite multigraph can be edge-colored in N colors. Each subset of edges assigned to a particular color forms a match, which is weighted by $\lceil T/N \rceil$. The fine matrix B for D does not need to be explicitly computed because its elements are guaranteed to be less than $\lceil T/N \rceil$. Thus, any N configurations that collectively represent every entry of B, each weighted by $\lceil T/N \rceil$, can be used to cover the fine portion. An example of DOUBLE execution is shown in Figure 1.20.

In summary, DOUBLE generates $2N$ schedules, each with holding length $\lceil T/N \rceil$. The total transmission makespan is $2N \times \lceil T/N \rceil + 2N \times \delta = 2T + 2N\delta$. DOUBLE has a time complexity of $O(N^2 \log N)$, which is mainly determined by the edge-coloring algorithm [59].

1.5.6 ADJUST Algorithm

Although the DOUBLE algorithm greatly enhances scheduling performance, its scheduling decision is rigid and does not change according to different system parameters. For example, if there are two switching systems with the same switch port number and accumulation time, but with different reconfiguration delay, DOUBLE

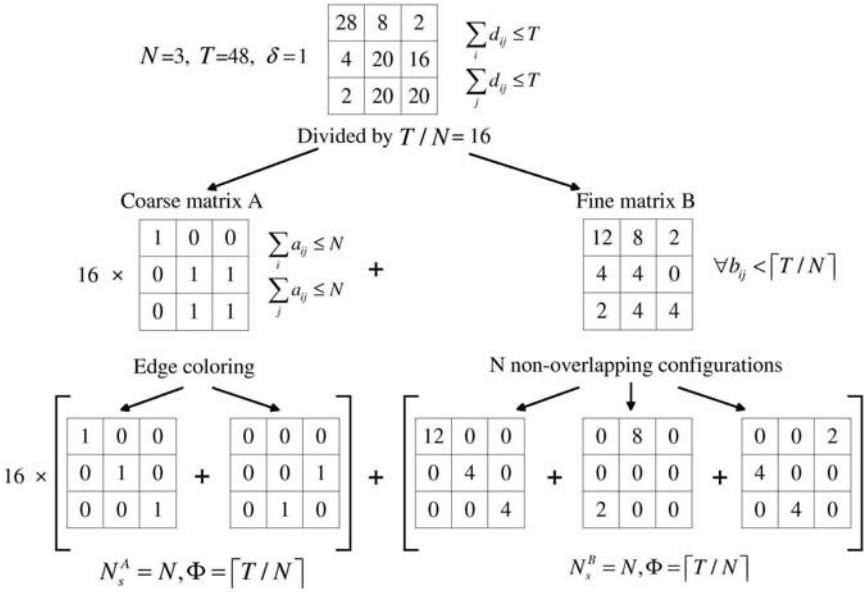


Figure 1.20. Example of the DOUBLE algorithm

will generate identical scheduling. However, it is natural that when the reconfiguration delay increases, the number of schedules should decrease to avoid incurring large overhead. Furthermore, simulation shows that a fixed choice of matrix division factor (T/N) may cause a large fine matrix with high cost, especially when the reconfiguration delay is relatively small compared to the accumulation length. All the above greatly influence the flexibility and performance of the DOUBLE algorithm. An enhancement ADJUST algorithm is therefore proposed. By introducing a regulating factor $\lambda = \sqrt{T/\delta N}$, it is able to self-adjust with different systems.

Similar to DOUBLE, ADJUST works by first separating the traffic D into a *quotient* matrix and a *residue* matrix and assigning schedules to each. The matrices are generated by dividing the elements in D by $T/\lambda N$. Under such a division, quotient and residue matrices can be covered by λN and N configurations, respectively. All configurations hold for $\lceil T/\lambda N \rceil$ time slots. The total transmission makespan is then $(T + \delta N) + (\delta\lambda N + \frac{T}{\lambda})$ (refer to [48] for details). The makespan is minimized when $\lambda = \sqrt{T/\delta N}$. An example of ADJUST execution is shown in Figure 1.21. For $N = 3$, $T = 48$ and $\delta = 1$, the regulating factor $\lambda = \sqrt{T/\delta N} = 4$.

It is obvious that the ADJUST algorithm always outperforms DOUBLE in the sense of makespan minimization. In fact, the DOUBLE algorithm can be viewed as a special case with λ set to 1.

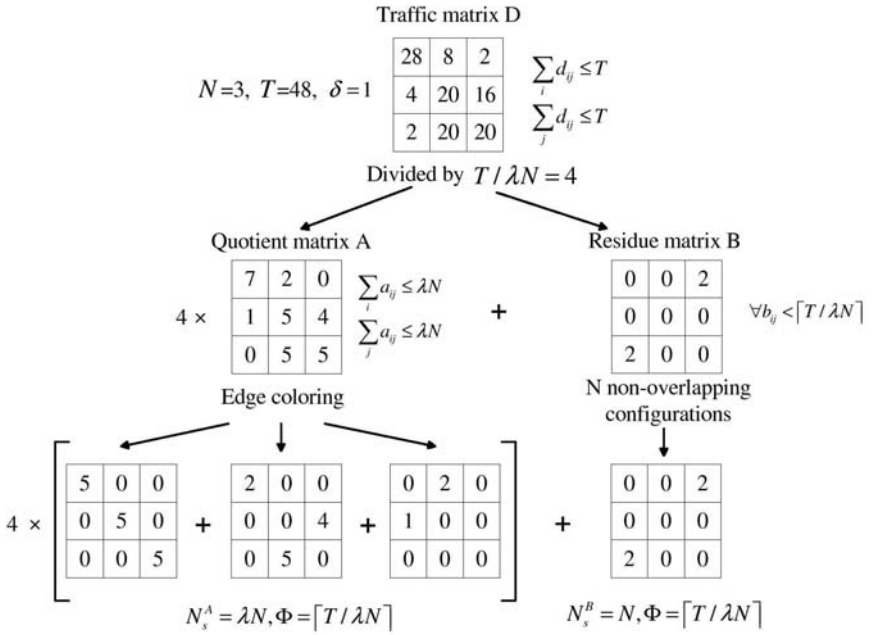


Figure 1.21. Example of the ADJUST algorithm

1.6 Conclusion

In this chapter, we have presented and discussed different router and ATM switch designs. We have discussed the bufferless switch architecture, the buffered switch architecture, the Clos-network architecture and the optical switching architecture.

The IQ crossbar switch is of interest due to its low hardware requirements, scalability and popularity. An IQ switch adopting a FIFO queue structure suffers from the HoL problem, which reduces the throughput of the switch to nearly half its available bandwidth. The speedup-based solution relies on the availability of advanced technology in terms of fast memory. However, this is totally impractical for high-performance routers. Very high speed is either uneconomical or simply impossible to achieve. The well-known efficient VOQ architecture is used to overcome the blocking problem and is of interest because of its ability to improve the switching capacity by several orders of magnitude. Despite its popularity, the VOQ solution has its own problems. A centralized scheduler is required to configure the switch 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 order to alleviate the centralized scheduler's task, the VOQ/BCS was proposed. The VOQ/BCS architecture is based on the strength of the VOQ architecture at the input side coupled with the internally buffered crossbar. This architecture has many advantages compared to the IQ switch. The internal buffering element enables

totally distributed VOQs arbitration and the scheduler is not centralized anymore. The distributed nature of the arbitration improves the switching performance by several orders of magnitude. By means of computer simulation, we showed the improvement of the VOQ/BCS over the IQ switch. Furthermore, the VOQ/BCS doesn't need any additional cost such as speedup.

In an attempt at scalability, single-stage switching techniques are considered to be inherently limited by their quadratic complexity. The Clos-network architecture is widely recognized as a very scalable architecture for high-speed switching systems. So far, only limited success has been reported in the design of practical distributed scheduling schemes for the Clos-network. The ATLANTA switch with MSM architecture is an example of a commercially successful Clos-network switch. However, it necessitates internal bandwidth speedup. Recently, the CRRD and CMSD algorithms have been proposed in order to overcome the throughput limitation and implementation complexity problem. Based on the static round-robin technique, SRRD has been proposed to reduce the delay performance and deduce the hardware complexity. However, due to the memory speedup up problem in the shared memory modules, the MSM arrangement of the Clos-network switches hinder the scalability to very large port size. A bufferless Clos-network architecture has been proposed in which the shared memory modules are replaced by bufferless crossbar switching components. Although it requires a greater communication overhead among distributed modules, the bufferless Clos-network architecture is an efficient way to solve the scalability problem in switching systems.

One attractive direction in building scalable switches and routers is to adopt optical switching techniques. However, a main challenge remains in efficiently queuing the packets in the optical domain. The techniques (*i.e.* fiber delay lines) are not yet mature enough to support such functionalities. A viable solution is to have a hybrid switch architecture, with electronic buffers and optical fabrics. Having optical fabrics provides more scalability and higher switching speed over the electronic counterparts.

We are still facing many challenging requirements pertaining to the design of scalable and high-performance routers. In summary, the new generation of switches and router should first be highly scalable in port number and interface speed. Moreover, routers have to offer many services to meet today's customer needs such as guaranteed delay, bounded delay variation, minimal packet loss and controlled access. Finally, a router has to be cost effective.

References

1. M. Karol, M. Hluchyj, and S. Morgan. Input versus output queuing on a space division switch. *IEEE Transaction on Communications*, 35(12):1347–1356, 1987.
2. N. McKeown. islip: A scheduling algorithm for input-queued switches. *IEEE/ACM Transaction On Networking*, 7(2):188–201, Apr 1999.
3. M. Nabeshima. Performance evaluation of combined input-and crosspoint-queued switch. *IEICE Transaction on Communicaitons*, E83-B(3), Mar 2000.
4. S. Nojima, E. Tsutsui, H. Fukuda, and M. Hashimmoto. Integrated packet network using bus matrix. *IEEE Journal on Selected Areas in Communications*, 5(8):1284–1291, Oct 1987.
5. L. Mhamdi, M. Hamdi, C. Kachris, S. Wong, and S. Vassiliadis. High-performance switching based on buffered crossbar fabrics. To appear in *Computer Networks Journal*.
6. T. Cheney, J.A. Fingerhurt, M. Flucke, and J.S. Turner. Design of a gigabit atm switch. *INFOCOM*, pages 2–11, Apr 1997.
7. F.M. Chiussi, J.G. Kneuer, and V.P. Kumar. Low-cost scalable switching solutions for broadband networking: the atlanta architecture and chipset. *IEEE Communication Magazine*, 35(3):44–53, Dec 1997.
8. A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queuing algorithm. *Inernetworking: Research and Experience*, 1(1):3–26, Sep 1990.
9. H. Zhang. Service disciplines for guaranteed performance service in packet-switching networks. *Proceeding of the IEEE*, 83(10):1374–96, Oct 1995.
10. S. T. Chuang, A. Goel, N. Mckeown, and B. Prabhakar. Matching output queueing with a combined input output queued switch. *INFOCOM*, pages 1169–1178, Apr 1999.
11. S. Y. Li. Theory of periodic contention and its application to packet switching. *INFOCOM*, pages 320–325, 1998.
12. N. McKeown, V. Anantharam, and J. Walrand. Achieving 100% throughput in an input-queued switch. *INFOCOM*, pages 296–302, Mar 1996.
13. A. Mekkittikul and N. McKeown. A starvation-free algorithm for achieving 100% throughput in an input-queued switch. *ICCCN*, pages 226–231, Oct 1996.
14. A. Mekkittikul and N. McKeown. A practical scheduling algorithm to achieve 100% throughput in input-queued switches. *INFOCOM*, pages 792–799, Apr 1998.
15. T. Anderson, S. Owicki, J. Saxe, and C. Thacker. High speed switch scheduling for local area networks. *ACM Transaction on Computer Systems*, 11(4):319–352, Nov 1993.
16. D. N. Serpanos and P. I. Antoniadis. FIRM: A class of distributed scheduling algorithms for high-speed atm switches with multiple input queues. *INFOCOM*, pages 548–555, Mar 2000.
17. L. Tassiulas. Linear complexity algorithms for maximum throughput in radio networks and input queued switches. *INFOCOM*, 2:533–539, Mar 1998.
18. P. Giaccone, B. Prabhakar, and D. Shah. Towards simple, high-performance schedulers for high-aggregate bandwidth switches. *INFOCOM*, pages 1160–1169, Jun 2002.
19. D. Gale and L. S. Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 69(9):9–14, 1962.
20. I. Stoica and H. Zhang. Exact emulation of an output queueing switch by a combined input output queueing switch. *IEEE/IFIP IWQoS*, pages 218–224, May 1998.
21. P. Krishna, N. S. Patel, A. Charny, and R. J. Simcoe. On the speedup required for work-conserving crossbar switches. *IEEE Journal on Selected Areas in Communications*, 17(6):1057–1066, Jun 1999.
22. R. Bakka and M. Dieudonne. Switching circuits for digital packet switching network. United States Patent 4,314,367, Feb 1982.

23. M. Katevenis. Fast switching and fair control of congested flow in broad-band networks. *IEEE Journal on Selected Areas in Communications*, 5(8):1315–1326, Oct 1987.
24. A. K. Gupta, L. O. Barbosa, and N. D. Gorganas. 16×16 limited intermediate buffer switch module for atm networks for b-isdn. *GLOBECOM*, pages 939–943, Dec 1991.
25. M. Lin and N. Mckeown. The throughput of a buffered crossbar switch. *IEEE Communications Letters*, 9(5):465–467, May 2005.
26. S. Chuang, S. Iyer, and N. McKeown. Practical algorithms for performance guarantees in buffered crossbars. *INFOCOM*, pages 981–991, Mar 2005.
27. F. Abel, C. Minkenberg, R. Luijten, M. Gusat, and I. Iliadis. A four-terabit packet switch supporting long round-trip times. *IEEE Micro Magazine*, 23(1):10–24, Jan/Feb 2003.
28. R. Rojas-Cessa, E. Oki, Z. Jing, and H. J. Chao. CIXB-1: Combined input one-cell-crosspoint buffered switch. *HPSR*, pages 324–329, May 2001.
29. R. Rojas-Cessa, E. Oki, Z. Jing, and H. J. Chao. CIXB-K: Combined input-crosspoint-output buffered packet switch. *GLOBECOM*, pages 2654–2660, Nov 2001.
30. D. Stephann and H. Zhang. Implementing distributed packet fair queuing in a scalable switch architecture. *INFOCOM*, pages 282–290, Apr 1998.
31. M. Katevenis and G. Passas. Variable-size multipacket segments in buffered crossbar (CICQ) architectures. *ICC*, pages 999–1004, May 2005.
32. L. Mhamdi and M. Hamdi. Output queued switch emulation by a one-cell-internally buffered crossbar switch. *GLOBECOM*, 7:3688–3693, Dec 2003.
33. B. Magill, C. Rohrs, and R. Stevenson. Output-queued switch emulation by fabrics with limited memory. *IEEE Journal on Selected Areas in Communications*, May:606–615, 2003.
34. A. Mekkittikul. *Scheduling Non-Uniform Traffic In High Speed Packet Switches And Routers*. PhD thesis, Stanford University, Nov 1998.
35. L. Mhamdi and M. Hamdi. Practical scheduling algorithms for high-performance packet switches. *ICC*, pages 1659–1663, May 2003.
36. L. Mhamdi and M. Hamdi. MCBF: A high-performance scheduling algorithm for internally buffered crossbar switches. *IEEE Communications Letters*, 7(9):451–453, Sep 2003.
37. K. Pun and M. Hamdi. Distro: A distributed static round-robin scheduling algorithm for bufferless clos-network switches. *GLOBECOM*, Nov 2002.
38. F.M. Chiussi and A. Francini. A distributed scheduling architecture for scalable packet switches. *IEEE Journal on Selected Areas in Communications*, 18(12):2665–2683, Dec 2000.
39. E. Oki, Z. Jing, R. Rojas-Cessa, and J. Chao. Concurrent round-robin dispatching scheme in a clos-network switch. *ICC*, 1:107–111, Jun 2001.
40. K. Pun and M. Hamdi. Static round-robin dispatching schemes for clos-network switches. *HPSR*, pages 329–333, May 2002.
41. C. Minkenberg, R. P. Luijten, F. Abel, W. Denzel, and M. Gusat. Current issues in packet switch design. *ACM SIGCOMM Comput. Commun. Review*, 33:119–124, 2003.
42. F. Masetti, P. Gavignet-Morin, D. Chiaroni, and G. Da Loura. Fiber delay lines optical buffer for ATM photonic switching applications. In *Proc. of INFOCOM*, pages 935–942, April 1993.
43. L. A. Buckman, K. S. Giboney, J. Straznicky, J. Simon, S. W. Corzine, X. J. Zhang, A. J. Schmit, and D. W. Dolfi. Parallel optical interconnects. In *Proc. Conf. Lasers and Electro-Optics (CLEO)*, pages 535–536, San Francisco, CA, May 2000.
44. S. Bregni, A. Pattavina, and G. Vegetti. Architectures and performance of AWG-based optical switching nodes for IP networks. *IEEE J. Select. Areas Commun.*, 21:1113–1121, September 2003.

45. H. J. Chao, K. L. Deng, and Z. Jing. Petastar: a petabit photonic packet switch. *IEEE J. Select. Areas Commun.*, 21:1096– 1112, September 2003.
46. L. Dittmann, C. Develder, D. Chiaroni, F. Neri, F. Callegati, W. Koerber, A. Stavdas, M. Renaud, A. Rafel, J. Sole-Pareta, Cerroni, N. Leligou, L. Dembeck, B. Mortensen, M. Pickavet, N. Le Sauze, M. Mahony, B. Berde, and G. Eilenberger. The European IST project DAVID: a viable approach toward optical packet switching. *IEEE J. Select. Areas Commun.*, 21:1026–1040, September 2003.
47. K. Kar, D. Stiliadis, T. V. Lakshman, and L. Tassiulas. Scheduling algorithms for optical packet fabrics. *IEEE J. Select. Areas Commun.*, 21:1143–1155, September 2003.
48. X. Li and M. Hamdi. On scheduling optical packet switches with reconfiguration delay. *IEEE J. Select. Areas Commun.*, 21:1156–1164, September 2003.
49. P. B. Chu, S. S. Lee, and S. Park. MEMS: the path to large optical crossconnects. *IEEE Commun. Mag.*, 40:80–87, 2002.
50. Tze-Wei Yeow, K. L. E. Law, and A. Goldenberg. MEMS optical switches. *IEEE Commun. Mag.*, pages 158–163, November 2001.
51. M. Hoffman, P. Kopka, and E. Voges. Thermooptical digital switch arrays in silica on silicon with defined zero voltage state. *J. Lightwave Technol.*, pages 395–400, March 1998.
52. J. E. Fouquet, S. Venkatesh, M. Troll, D. Chen, H. F. Wong, , and P. W. Barth. A compact, scalable cross-connect switch using total internal reflection due to thermally-generated bubbles. In *Proc. IEEE/LEOS Annu. Meeting*, pages 169–170, Orlando, FL, May 1988.
53. X. H. Ma and G. H. Kuo. Optical switching technology comparison: optical MEMS vs. other technologies. *IEEE Commun. Mag.*, 41:S16–S23, 2003.
54. P. D. Dobbelaere, K. Falta, S. Gloeckner, and S. Patra. Digital MEMS for optical switching. *IEEE Commun. Mag.*, 40:88–95, 2002.
55. L. Y. Lin, E. L. Goldstein, and R. W. Tkach. Free-space micromachined optical switches for optical networking. *IEEE J. Select. Topics Quantum Electron.*, 5:4–9, 1999.
56. T. Inukai. An efficient SS/TDMA time slot assignment algorithm. *IEEE Trans. Commun.*, 27:1449–1455, October 1979.
57. I. S. Gopal and C. K. Wong. Minimizing the number of switchings in an SS/TDMA system. *IEEE Trans. Commun.*, 33:497–501, June 1985.
58. B. Towles and W. J. Dally. Guaranteed scheduling for switches with configuration overhead. *IEEE/ACM Trans. Networking*, 11:835–847, October 2003.
59. R. Cole and J. Hopcroft. On edge coloring bipartite graphs. *SIAM J. on Comput.*, 11:540–546, August 1982.