SETIT 2007
4rth International Conference: **S**ciences of **E**lectronic, **T**echnologies of **I**nformation and **T**elecommunications
March 25-29, 2007 – TUNISIA

Next Generation Internet High-Speed Switches and Routers

Feng Wang and Mounir Hamdi

Hong Kong University of Science and Technology
fwang@cse.ust.hk
hamdi@cse.ust.hk

Abstract: Shared memory architecture for packet switches was normally thought to be unsuitable for building high performance switches/routers. The main reason lies in their perceived poor scalability. In particular, shared memory architectures are typically used to build output-queued switches which are regarded as the best candidate to achieve optimal delay-throughput performance. The current trend in router/switch design in both industry and academia favors crossbar-based architectures with VOQ techniques because they provide a scalable solution. Although shared memory architectures seem to have the obvious scalability disadvantage, crossbar-based architectures have their own intrinsic limitations, such as complex scheduling algorithms and higher bandwidth allocation compared with shared memory architectures of the same capacity. In this survey, we investigate the problem of shared memory design in detail and try to find alternatives to solve the scalability bottleneck. In the end, we show that combining the crossbar and distributed shared memory architecture is the most promising method for building scalable high performance switches/routers that can provide quality-of-service support.
Key words: Shared Memory, Crossbar, Scalable, Space-Memory-Space switch architecture

1. Introduction

The basic problem in the core of networking is to resolve contentions for shared resources. Usually, contention is among distributed parties without prior coordination. Switches are the basic building blocks for interconnections. The packet switch is a multi-port device that routes incoming packets from many input ports to the proper output ports, while resolving contentions (multiple packets simultaneously desiring to exit through the same output port) by temporarily buffering all packets but one, then scheduling their departure at an appropriate later time.

For packet switches, memory strategy is very crucial affecting the performance. The very embodiment of resource contentions often happens in memories. In the shared memory architecture, if memory bandwidth is sufficiently large which means the resource provided is enough, contentions are removed naturally. But if we only have limited memory bandwidth which means the resource is limited, one possible way is to use space division to provide some more resources, which is the initial idea of the introduction of crossbar based architecture.

Looking into the development of high performance switches/routers, we can find there are three or four generations of switch architectures existed in history: [1]

1. Shared memory architecture, as illustrated in Figure 1.
2. Shared medium architecture, as illustrated in Figure 2.
3. Crossbar based switch architecture, as illustrated in Figure 3, which can be further categorized into input-queued (IQ), output-queued (OQ) and combined-input-output-queued (CIOQ) switches.

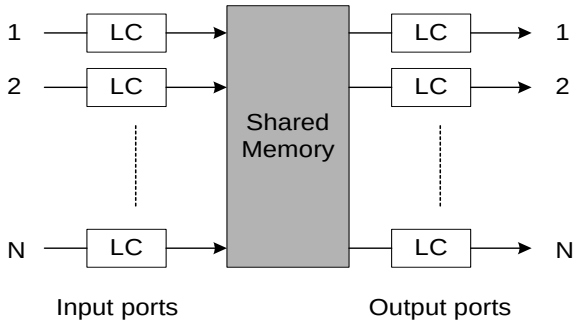

Figure 1: Shared memory switch architecture

The fourth generation is all optical switch architecture [2] which is current trend in this coming optic era and out of the scope of this survey.

Contemporary researchers mainly focus on crossbar based architectures and design many corresponding scheduling algorithms for them varying from simple to complicate. The CIOQ router is frequently referred as an abstract model for crossbar based routers: at one extreme is input queuing, at the other extreme is output queuing, and in between there's a continuum of

performance as the speedup is increased from 1 to *N* (where *N* is the number of inputs.) The most challenging task in crossbar based routers is to design efficient and fast scheduling algorithms for them. There are rich and growing theories and practical implementations for CIOQ routers, such as PIM [25], iSLIP [10], DRRM [4], FIRM [3], static round robin (SRR) [15] and so on …

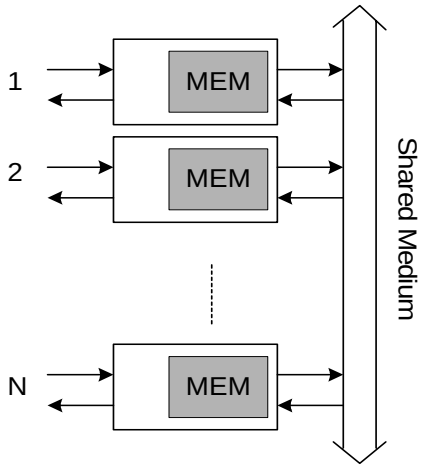

Figure 2: Shared Medium switch

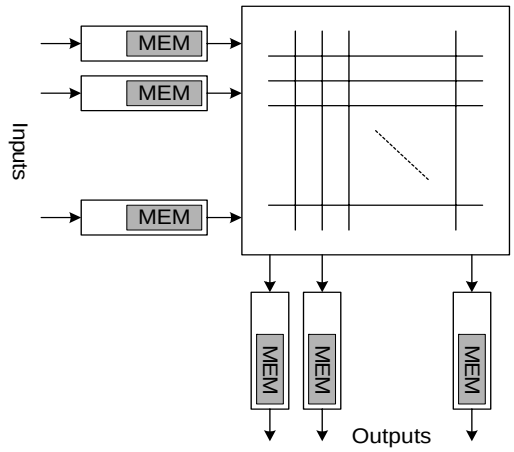

Figure 3: Crossbar based switch

While researchers in academia favor crossbar based architecture, shared memory architecture is still commonly used in industry, with both IBM Corp. and Applied Micro Circuits Corp. (AMCC) shipping since 1992 [5]. Most recently, a startup company called Terachip released its main product – a 160 *Gbps* switch fabric called TCF 16x10 based on shared memory architecture [6]. As we can see, shared memory architecture does not fade away in high performance switches; instead it always finds its stand in all the production lines from low to high ends.

So, what is the potential advantage of shared memory architecture over crossbar based switches? Switch fabric developed using a crossbar architecture requires an off-chip scheduler to control the movement of packets across the backplane [21]. In academia, we always assume we can devise some complicated scheduling algorithms for crossbar to achieve 100% throughput and small delay, while in real implementation we need hardware to accomplish these algorithms. The scheduler chip has been one of the big challenges for the crossbar approach [22]. Additionally, the scheduler requires tight synchronization between the switch and line cards. While for the shared memory architecture, the memory itself performs as a switch. In fact, scheduling in shared memories is very trivial and straightforward.

In the shared memory switch, the center of the switch fabric is a shared memory that contains queues for different output ports. Incoming packets are moved into the shared memory fabric device as quickly as possible [20]. Packets are then scheduled out of the shared memory according to the order of their departure times.

However, shared memory architecture is criticized as non-scalable. The basic reason is that, as shown in Figure 1, the bandwidth requirement of the shared memory is 2*NR* [29], which scales linearly with the number of inputs *N* and the line rate *R*. We will look into some techniques to make the shared memory scalable with the number of inputs and even scalable with the increasing line rate.

Crossbar and shared memory can both perform the switching functions. It will benefit if combining them together, taking the bandwidth and power advantages from shared memory and scalability of crossbar architecture. The most suitable candidate is the multi-stage architecture called space-memory-space (SMS) switch. The basic idea is to make all the stages of this architecture take part in the switching functions. Although there have been much literature discussing memory-space-memory (MSM) structure, one shortage of this architecture is that the memories used are not shared, thus losing the functions of switching.

The rest of this survey is organized as follows: We first survey on the popular crossbar based architecture with VOQ techniques. Then we discuss heavily on shared memory architecture and how to make it practical. In the end, we combine crossbar and shared memory to build scalable shared memory switch architecture.

2. Crossbar based architecture

With the growing traffic demand of the Internet, one challenging requirement facing the design of high performance switches/routers is that they should be scalable [24]. By saying to be *scalable*, we mean the following:

1. The physical access time of each individual component of the switches is independent on the number of input ports.
2. The complexity of scheduling algorithms (if needed) for switches does not scale with the number of input ports significantly.

It is obvious to see that the shared memory architecture as shown in Figure 1 is *not* scalable, since it requires a memory bandwidth of 2*NR* which scales

linearly with the number of input ports. One possible and most favorable solution by researchers is to use crossbar as the switch fabric, as illustrated in Figure 3, thus by dividing memories into two stages and making them non-shared, memory bandwidth requirement can be significantly reduced to 2*R*.

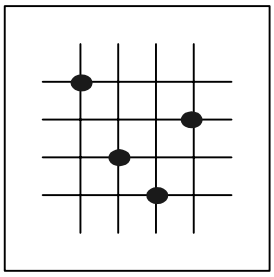

Figure 4: A configuration of a 4x4 crossbar switching fabric

Crossbar is normally made of N^2 switching points. Figure 4 shows a 4x4 crossbar switching fabric with 16 switching points. By allowing at most one switching point in each row and each column to be turned on, any permutation of the input ports can be mapped into one configuration of the crossbar. For example, Figure 4 represents four input-output pairs of (1, 1), (2, 4), (3, 2) and (4, 3).

Although resource contentions are now transferred to the crossbar switching fabric, we can distribute the contentions into N^2 switching points to make sure no components here operate faster than line rates *R*. Therefore, it is possible to make crossbar based switches *scalable*. If we regard the shared memory architecture as time division switching, crossbar based switching can be regarded as space division switching [1], alleviating time resource contentions by using more space resources.

2.1. IQ, OQ and CIOQ switches

Crossbar based switch architecture exhibits more interesting properties. In general, crossbar based switches can be categorized into input-queued (IQ) switches, output-queued (OQ) switches and combined-input-output-queued (CIOQ) switches, according to the speedups needed in the crossbar switching fabric.

Speedup is a common term used in crossbar switches. By saying a speedup of *S*, we mean that the crossbar can move up to *S* packets from each input and deliver up to *S* packets to each output within a time slot. [32]

If the crossbar operates at a speedup of 1 which means at most one packet can reach one output port in one time slot, packets can only queue in the input side, we call this situation IQ switches. If the crossbar can operate at a speedup of *N* which means the crossbar can move up to *N* packets to one output port in one time slot, packets can only queue in the output side, and we call this situation OQ switches. In between the two extreme situations, if the crossbar's speedup is great than 1 but less than *N*, we need buffers in both input and output sides, and we call this situation CIOQ switches.

2.2. OQ switches emulation

OQ switches have the optimal delay-throughput performance [16]. In addition, nearly all the QoS algorithms and AQM schemes assume an OQ switch [33]. However, implementing a straightforward OQ switch is very challenging since the crossbar needs to run *N* times faster than the line rate which is nearly the same situation as in shared memory architecture, thus making crossbar based OQ switches only suitable for edge routers with limited input ports and low line rates. In order to make it scalable, the basic requirement is that the speed of the crossbar should be independent on the number of input ports.

On the other hand, research shows that we can use CIOQ switches to emulate OQ switches. By saying emulating, we means that by feeding the same traffic pattern to the ideal OQ switch and CIOQ switch with a speedup of *k* (less than *N* and independent on *N* hopefully), the output packet sequences of the two switches are identical [12].

The well-known theorem in [18] shows that:

A speedup of 2-1/N is sufficient and necessary for a CIOQ to emulate an OQ switch.

2.3. Crossbar architecture with VOQ technology

Although the theorem above is exciting and elegant, it is impractical in hardware implementation. First, it assumes a Push-In-Arbitrary-Out (PIAO) queue which is very difficult to implement. Second, the scheduling algorithm employs a stable-marriage-problem algorithm [28] which has rather high complexity. Normally, the more practical way in industry is that we always prefer some simple First-In-First-Out (FIFO) queues and scheduling algorithms with less time complexity.

However, simple memory strategy such as FIFO will limit the throughput to around 58% due to the notorious HoL blocking problem [7]. To resolve this problem, virtual-output-queue (VOQ) concept is proposed as a common practice, as illustrated in Figure 5. Rather than maintaining a single FIFO queue for all cells, each input maintains separate queues for the cells directed to different outputs [15]. VOQ technique with proper scheduling algorithm can resolve the HoL blocking problem thoroughly. Crossbar architecture together with VOQs can abstract the switching problem into the problem of bipartite matching in graph theory. For example, the switching situation in Figure 5 (a) can be represented using a bipartite graph in Figure 5 (b). The scheduling algorithm is required to find the maximal matching of the bipartite graph for each round of packet transmission. For example, the thick lines in Figure 5 (b) represent the maximal match for the current traffic request.

Although VOQ technique solves the HoL blocking problem, finding the maximal matching of the bipartite graph is not a trivial task. From graph theory, the most efficient centralized algorithm we know till now

has the time complexity of $O(N^{2.5})$ [8], where N is the number of linecards. In other words, the centralized scheduling algorithms for crossbar based switches with VOQs are not scalable, thus becoming the bottleneck of performance. In practice, researchers tend to devise distributed algorithms to find the maximal matching quickly. Most of their algorithms are heuristic algorithms which are hoped to converge to the maximal matching quickly, such as PIM, iSLIP, DRRM, FIRM, SRR and so on…

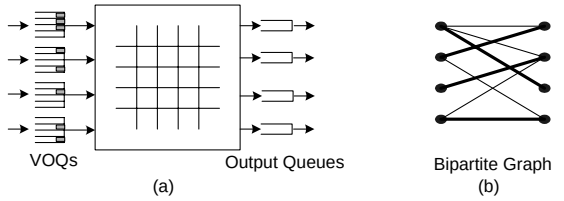

Figure 5: Bipartite graph representation of a traffic request.

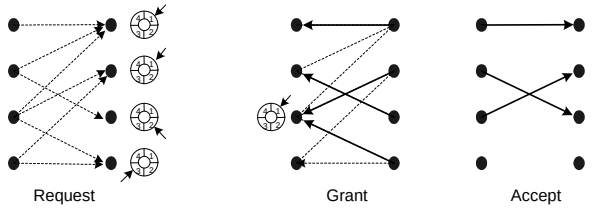

Figure 6: First iteration of the RGA algorithms

The basic idea of all these heuristic algorithms is that they try to behave in a distributed manner. We call all these kinds of algorithms RGA (Request-Grant-Accept) algorithms. A typical RGA algorithm runs as shown in Figure 6.

- *Request*: First, Each input sends a request to every output for which it has a queued cell.
- *Grant*: Secondly, 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 scheduler is updated using a scheme which differs slightly in various algorithms.
- *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 updated according to various algorithms.

Distribution always causes contention, especially in the core of networking where all traffic aggregates. All the algorithms stated above, such as RRM, iSLIP, DRRM, SRR and FIRM, differ only in the schemes of updating the pointers. The original RRM algorithm suffers from pointer synchronization. So, its successors try alternative pointer updating schemes to avoid contentions as much as possible. From the simulation results, we know that FIRM has the best throughput performance under normal traffic conditions [3].

3. Making shared memory architecture practical

Shared memory architecture is common in the early days of routers when the memory bandwidth is much larger than the aggregate line rates [20]. The basic structure of a shared memory switch is shown in Figure 1. In shared memory architecture, scheduling is simple because the memory performs switching naturally. When a packet arrives, it is put into the shared memory immediately. When it is time to leave, it departs from the memory in time [29]. If the bandwidth is large enough, nearly no scheduler is needed here.

No scheduler or less complex scheduler is the most attracting point of shared memory architecture. However, with the introduction of optical fibers, memory bandwidth can not catch up with the increasing line rates any more. As we have seen in the previous section, a straightforward implementation of shared memory architecture requires a bandwidth of $2NR$ which scales *linearly* with the number of input ports. We need to use some techniques to make the shared memory architecture scalable. We first survey on the techniques about cutting down the memory bandwidth requirement to line rate R, then on techniques that make memory even more scalable with the ever increasing line rates.

3.1. Parallel memories

In order to make shared memory architecture scalable, we should make the memory bandwidth requirement independent on the number of input ports. The technique is to use parallelism. Actually, it is a natural idea. If one piece of memory can not meet the bandwidth requirements, we use multiple of them. The basic idea is shown in Figure 7.

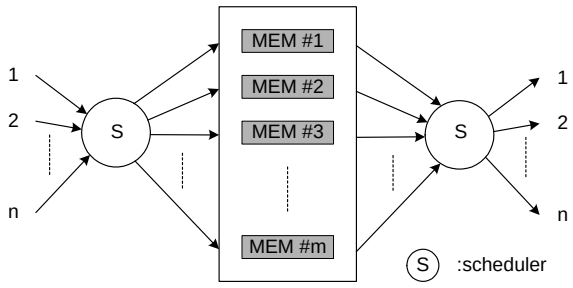

Figure 7: Parallel memories shared by all the inputs

In the Figure above, every single memory in the central stage runs at the line rate, which means that in one time slot there's at most one read and on write operation performed on one memory [18]. We use m memories in parallel to distribute the memory bandwidth requirements of $2NR$. We defer the details of the scheduler **S** to the next section. Now, we focus on the memory requirement in this parallelism strategy.

When a packet comes in the input side scheduler, it will find a *free* memory to be written in. By saying *free*, we mean the following [13]:

- The memory is not written by other packets, which are up to N-1 packets.

- The memory does not contain packets which will depart at the same time as this incoming packet, which are up to N-1 packets.

From the *pigeon-hole* principle, we can see that the minimum number of memories required is

$(N\text{-}1) + (N\text{-}1) + 1$

$= 2N\text{-}1.$

This number is a little interesting. We have seen that in the original straightforward shared memory implementations, the number of queues (memories) we need to maintain is N (the number of output ports). So, we can say that if we use parallelism to distribute the memory bandwidth requirements, we need a space extension ratio of $(2N\text{-}1)/N = 2\text{-}1/N$, which is exactly the same as the speedup needed for a CIOQ switch to emulate an OQ switch. Note that a shared memory switch is naturally an OQ switch.

Another advantage of this parallel memory technology is that we can distribute all the memories into individual line cards [9]. This will make the switch system maintenance more handy and flexible. We remind here that although memories are distributed into line cards, they are still shared by all the inputs and outputs. That is to say, every memory in individual line cards is accessible by all the inputs and outputs.

Although parallel memories provide us a possible way to scale the shared memory switches, the benefit does not come for free. It is at the cost of a scheduling algorithm. We need an additional scheduling algorithm in the input side to dispatch packets into the *free* memories when they arrive at the input ports. We will discuss the algorithm in the next section.

The output side scheduler is somewhat easy to devise. It just selects the packets which are in their departure times and lets them go. Since packets have resolved all of their contentions in the input side already, no scheduling algorithms are needed in the output side.

3.2. Memory techniques scalable with line rates

As the line rate keeps increasing, buffers needed in routers have to be sufficiently large, according to the *rule-of-thumb* equation: $B = R \times RTT$ [30], where R is the line rate, RTT stands for round-trip-time and B is the buffer needed in the line card. In other words, we need large and fast memories to keep up with the buffering and switching requirements of high performance routers. Current memory technologies are mainly SRAM and DRAM. DRAM offers large capacity to hold many packets, but its random access time is too slow. On the other hand, SRAM is fast and might be able to keep up with line rates, but are too small to be economically viable for large packet buffers.

3.3. Combined SRAM and DRAM technologies

One possible and natural means is to use combined SRAM and DRAM [11]. The main idea is to build a memory hierarchy, where the memory bandwidth is increased by reading (writing) multiple cells from (to) DRAM memory in parallel. When packets arrive to the switch they are stored temporarily in an SRAM, waiting their turn to be written into DRAM. At the appropriate time (determined by a memory management algorithm), multiple packets are written into the DRAMs at the same time [11]. We can think of the memory hierarchy as a large DRAM containing a set of FIFOs; the head and tail of each FIFO is cached in a (possibly on-chip) SRAM as shown in Figure 8. The SRAM is sized so that whenever the arbiter requests a packet, it is always ready in the SRAMs so that it can depart in time, regardless of the sequence of requests.

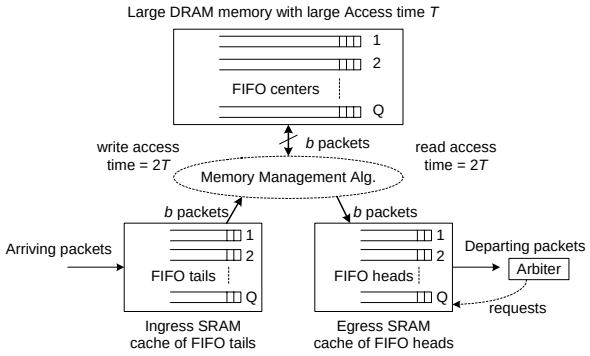

Figure 8: Combined DRAM and SRAM

In particular, we have to determine how large the SRAM needs to be, and find algorithms for deciding when to replenish the SRAM cache so as to minimize its size, or to minimize latency. In [34], the authors proved that:

- An SRAM cache of size must be at least $Q(b\text{-}1)(2+\ln Q)$.

With the Most Deficit Queue First memory management algorithm, an SRAM cache of size $Qb(2+\ln Q)$ is sufficient, where Q and b are defined as in Figure 8.

3.4. Parallel memories working with packet stripping

Wavelength division multiplexing (WDM) is making available long-haul fiber-optic links with very high capacity. When the line rate keeps increasing and exceeds the bandwidth of single memory, we need other techniques to overcome this insufficiency. Still, the philosophy of parallelism is used here. It is natural to consider using a parallel packet switch (PPS) [12] architecture comprised of multiple identical lower-speed packet switches to form a high performance router.

The incoming stream of packets is spread, packet-by-packet, by a de-multiplexer across the slower packet-switches as shown in Figure 9, then recombined by a multiplexer at the output. As seen by the arriving packets, a parallel packet switch (PPS) is a single-stage packet switch; all of the buffering is contained in the slower packet switches in the central stage [12].

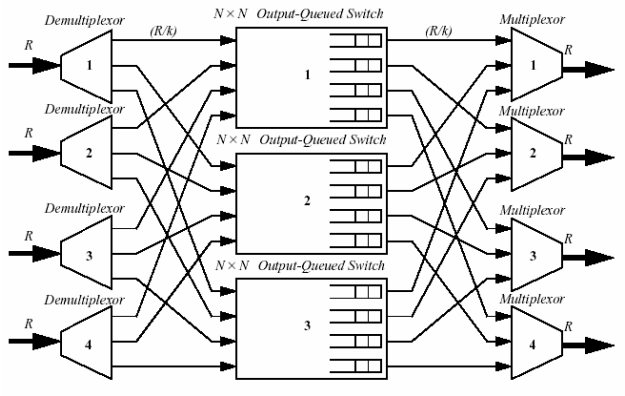

Figure 9: Parallel Packet Switch

In Figure 9, there are *k* low-speed packet switches in the central stage. Each low-speed packet switch operates at a fraction of the line rate *R*, for example, *R/k*. If we use output-queued switch in the central stage, it is desirable to make the whole system emulate an OQ switch. Results in [31] show that it is theoretically possible for a PPS to emulate a FCFS output-queued packet switch if each OQ switch operates at a rate of approximately 2*R/k*. This simple result is analogous to Clos's theorem [19] for a three stage circuit switch to be strictly non-blocking and also the ratio of 2-1/*N* for parallel memories to emulate a shared memory switch.

4. Scalable shared memory architecture

Shared memory architecture is the basic and straightforward form to represent and resolve the contentions in networks. The original simple implementation of shared memory switch, as illustrated in Figure 1, is the most ideal switch and naturally an OQ switch which does not need any scheduler. However, it is intrinsically non-scalable. In philosophy, it aggregates network resource contentions all in the memories and by providing sufficient memory (bandwidth) resources it performs switching smoothly. When memory bandwidth becomes a bottle neck, we need to distribute the contention resolutions to other components.

With current technology, packet switching is accomplished by two major methods. One is to use shared memory, and the other is to use crossbar or some other space-division switching technologies [31]. However, most work in academia investigates in either direction deeply and does not take both advantages of shared memory and crossbar. For example, in most shared memory architectures, they do not use crossbar, and in most crossbar architectures, memories are separated in the input and output sides [10] and not shared, thus losing its intrinsic role of switching.

To build high performance switches/routers with large number of input ports, scalability is among the most important concerns [23]. In the previous section, we surveyed techniques on how to make the memories scalable. We left the scheduler dispatching packets into the memories yet to be discussed. We survey the schedulers to make shared memory architecture fully scalable in this section.

4.1. Space-Memory-Space architecture

We can see that the scheduler **S** for shared memory architecture in Figure 7 performs an *N*x(2*N*-1) mapping basically. It is natural to use a *N*x(2*N*-1) crossbar to achieve this task. We know that for crossbar, since it is scalable, every component will run at the line rate.

However, as stated in the previous section, we need an additional algorithm to dispatch the incoming packets into individual memories according to the two *free* constraints we stated above. We encounter the bipartite graph matching problem again. However, the situation here is a little different from the one in the VOQ scheduling in crossbar switches.

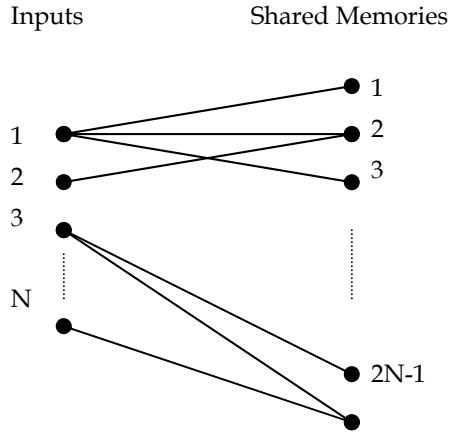

Figure 10: bipartite graph in the parallel shared memory

Figure 10 is the bipartite graph representation of the memory assignment needed here. A link between input *i* and memory *j* means that memory *j* is compatible for *i* with regards to the second *free* constraint, which is to say, memory *j* does not contain packets bearing the same departure time as packet *i*. We compare it with the bipartite graph representation in VOQ scheduling in Figure 5 (b) to find the essential differences.

- This graph is not symmetric, with *N* nodes in the input side and 2*N*-1 node in the memories side.
- The degree of every input node *i* is at least *N*.
- The number of maximal matching is always equal to *N*.

These three properties make efficient algorithms for memory assignments a little easier to design. Even for the heuristic distributed algorithms, they are proved to be faster converging to the maximal matching. The most valuable work was done by Amit et al. [13].

First, they show that the matching can be computed in $O(log^2 N)$ time using a parallel computer on a CREW PRAM. [13]

Second, they also follow the same three stage paradigm of Request-Grant-Accept (RGA) to design

some randomized and scalable algorithms. Furthermore, by pipelining, they proved that their algorithm will converge in $O(log^{*}N)$ rounds with each round running at a constant time, which is extremely fast and, as they claimed, near optimal[14].

Shared memory sandwiched by two stages of crossbar is usually referred as Space-Memory-Space (SMS) architecture. It has the most distinctive performance since all of its parts take part in the role of switching. While in some other multi-stage architectures, such as Memory-Space-Memory (MSM), the memories employed by them are not shared, thus losing the function of switching.

4.2. Load-balanced packet switches

SMS architecture has many variants, among which there is a recent famous architecture called load-balanced birkhoff-von Neumann switch [17], as illustrated in Figure 11. It uses N VOQ memories in the central stage rather than $2N$-1 FIFO queues, and most creatively, they do not employ any real scheduler in both crossbar sides. They just let them act in a predefined iteration way. For example, every input just sends packets to the central shared memory in a round-robin rotating fashion. The memories they use and the packet scheduling they employ are obviously *scalable*. Most surprisingly, they prove that this architecture can achieve 100% throughput in most traffic conditions. Although they start with an initial motivation of pre-process the traffic with a load balancing scheme, it is really a variant of scalable shared memory architecture.

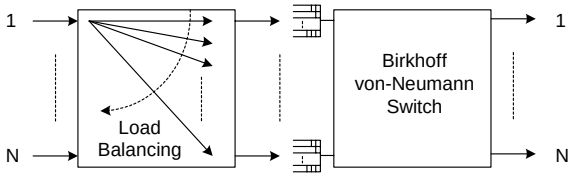

Figure 11: Load-balancing Birkhoff-von Neumann switch

4.3. Central Buffered Clos-network

Crossbar switching technology is *scalable*, but the cost of crossbar is very expensive. Normally, it is at the order of $O(N^2)$. This will become even more costly when N is extremely large, for example, in the network Point-of-Presences (POP). Some work has been led to deal with the economic scalability of the first and the third stage of the SMS architecture stated above. It employed ideas from Clos-network [19], using modular design philosophy.

In Figure 12, we show a Central Buffered Clos-network (CBC) switch [16]. This architecture resembles traditional Clos-network except that the central-stage switches are split into two identical copies with memories linking each port pair. Note that the memories here are fully shared by all the inputs and outputs.

When a packet arrives in one of the incoming ports, it is dispatched into one compatible memory immediately. By saying to be compatible, we mean exactly the same *free* constraints as stated in the previous parallel memory architecture.

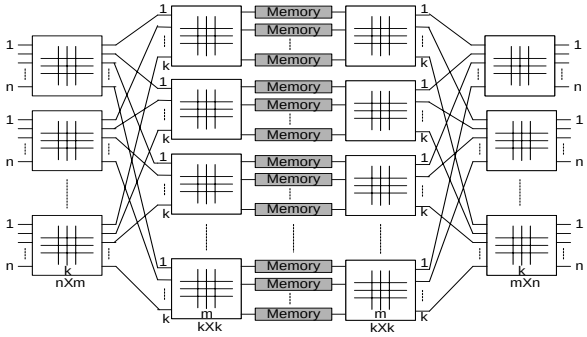

Figure 12: Central Buffered Clos-network switch

Mapping this CBC switch architecture into the SMS architecture, we can see that it uses a modular Clos-network as the first and third space-division switching part, thus make it cost effective and more scalable. Using the same pigeon-hole principle, the authors analyzed that the space extension ratio of $m = 2 - 1/k$ is sufficient for CBC to emulate an FCFS output-queued switch, where k is the number of the central modules needed in CBC architecture and the number of total input ports $N = mxk$.

The number $2 - 1/k$ is another interesting Figure we should pay attention to since it is slightly smaller than $2 - 1/N$. The smaller extension ratio is achieved by using a multi-stage architecture, compared with the single stage of switching we surveyed in previous sections.

5. Conclusions and future work

Shared memory architecture for packet switches was common in the early days of networking. With the increasing of carrier's line rates, researchers changed their focus to many space-division switching technologies, such as crossbar based architecture, since shared memory architecture was likely to be non-scalable. However, crossbar based architecture is far from perfect for packet switching. Scalable scheduling algorithms achieving 100% throughput and less delay for crossbar are very challenging to obtain. Usually, we can only afford some heuristic algorithms which are only suitable in some benign traffic conditions.

In this survey, crossbar based architecture and its corresponding scheduling algorithms are first investigated. After that, we turn to shared memory architecture and explored various techniques to make shared memory architecture practical and scalable. In the end, we identified current switch design trends that combined space and memory strategy for switching is the most suitable for high performance switching in the near future.

Three techniques appear very commonly in the design and analysis of scalable high performance switches/routers.

1. Parallelism

2. RGA algorithms
3. Pigeon-hole principle

When memory bandwidth becomes a bottle neck, try to use them in parallel. But the costs we pay for the benefits gained may be that sometimes we are involved in a resource expansion and an additional scheduling algorithm which is complicate in most situations.

RGA algorithms are mainly used in finding maximal matching in the bipartite graph, whether it represents the input-output matching pair or it represents the input-memory matching pair.

Pigeon-hole principle is very useful in analyzing resource requirements by distributed many parties. We rely heavily on pigeon-hole principle in the memory requirement analysis in this survey. Even in pure space switching such as traditional Clos-network for circuit switching, pigeon-hole principle is a good tool [19] to use to find the necessary and sufficient resource requirements.

Although we pointed out the most possible architecture for future scalable high performance switches/routers, there are still many challenging problems remained, one of which is the scheduling algorithms for SMS packet switching architecture. These algorithms are hoped to have less complexity than those for scheduling packets in VOQ based crossbar architectures.

Another possible direction is that for even higher speed switches which leave very limited time for the scheduler to find a maximal matching, we can only employ some heuristic algorithms. And maybe we need to modify the strict SMS architecture a little by adding some small buffers in the first and third stage. In this case, scheduling algorithms will become more interesting and challenging, concerning the role of the small buffers.

REFERENCES

[1] F. A. Tobagi, *Fast packet switch architectures for broadband integrated services digital networks*, in proceedings of IEEE, vol. 78, issue 1, Jan. 1990

[2] X. Li and M. Hamdi, *On Scheduling Optical Switches with Reconfiguration Delay*, IEEE Journal on Selected Areas in Communications (JSAC), vol. 21, Issue 7, 1156-1164, Sep 2003.

[3] D. N. Serpanos and P. I. Antoniadis, *FIRM: a class of distributed scheduling algorithms for high speed ATM switches with multiple input queues*, in Proc. IEEE INFOCOM, 2000, pp. 548-555.

[4] H. J. Chao, J. S. Park, *Centralized contention resolution schemes for a large-capacity optical ATM switch*, in Proc. of the IEEE ATM workshop, 1998.

[5] URL: *http://www.lightreading.com/document.asp?doc_id=25989*

[6] URL: *http://www.commsdesign.com/story/OEG20030203S0062*

[7] M. Karol, M. Hluchyj, and S. Morgan, *Input versus output queuing on a space division switch*, IEEE Trans. Commun., vol. 35, pp. 1347-1356, 1988

[8] J. E. Hopcroft and R. M. Karp. *An $O(N^{2.5})$ Algorithms for Maximum Matching in Bipartite Graphs*, Society for Industrial and Applied Mathematics Journal of Computation, vol. 2 pp. 225-31, 1973

[9] S. Iyer, R. Zhang, and N. McKeown, *Routers with single stage of buffering*, in proceedings of SIGCOMM, 2002

[10] N. McKeown, *The iSLIP scheduling algorithm for input queued switches*, IEEE/ACM Transactions On Networking, vol. 7, no. 2, April, 1999

[11] S. Iyer and N. McKeown, *Techniques for fast shared memory switches*, Stanford HPNG technical report TR01-HPNG-081501

[12] S. Iyer, A. Awadallah, and N. McKeown, *Analysis of a packet switch with memories running slower than the line-rate*, in proceedings of IEEE Infocom, 2000

[13] A. Prakash, S. Sharif and A. Aziz, *An $O(\log^2 N)$ parallel algorithm for output queuing*, in proceedings of IEEE INFOCOM, 2002

[14] A. Aziz, A. Prakash and V. Ramachandran, *A near optimal scheduler for switch-memory-switch Routers*, ACM Symposium on Parallelism in Algorithms and Architectures, 2003

[15] Y. Jiang and M. Hamdi, *A fully desynchronized round-robin matching scheduler for a VOQ packet switch architecture*, in proceedings of IEEE Workshop on High Performance Switching and Routing, 2001, pp. 407-411

[16] F. Wang and M. Hamdi, *Analysis on the central-stage buffered Clos-network for packet switching*, to appear in proceedings of IEEE ICC, 2005

[17] C. S. Chang, D. S. Lee and Y. S. Jou, *Load balanced Birkhoff-von Neumann switches, part I: one-stage buffering*, Computer Communications 25 (2002) 611-622

[18] S. T. Chuang, A. Goel, N. McKeown and B. Prabhakar, *Matching output queuing with a combined Input/Output-queued switch*, IEEE Journal on Selected Areas in Communications, vol. 17, no. 6, June 1999

[19] C. Clos, *A study of non-blocking switching networks*, Bell Systems Technical Journal, pp. 406-424, March 1953

[20] A. Singhal and R. Jain, *Terabit switching: a survey of techniques and current products*, Computer Communications 25 (2002) 547-556

[21] H. J. Chao, C. H. Lam and X. Guo, *Fast ping-pong arbitration for input-output queued packet switches*, Int. J. Commun. Syst. 2001, 14: 663-678

[22] K. Lee, S. J. Lee, and H. J. Yoo, *A Distributed On-Chip Crossbar Switch Scheduler for On-Chip Network*, Custom Integrated Circuits Conference (CICC), September, 2003

[23] H. J. Chao, *Next generation routers*, in proceedings of the IEEE, vol. 90, pp. 1518-1558, 2002.

[24] I. Keslassy, S. T. Chuang, K. Yu, D. Miller, M. Horowitz, O. Solgaard, and N. McKeown, *Scaling Internet routers using optics*, in proceedings of SIGCOMM 2003

[25] T. Anderson, S. Owicki, J. Saxie and C. Thacker, *High speed switch scheduling for local area networks*, ACM Trans. Comput. Syst., vol. 11, no. 4, pp. 319-352, Nov. 1993

[26] B. Prabhakar and N. McKeown, *On the speedup required for combined input and output queued switching*, Automatica, vol. 35, 1999

[27] I. Stoica and H. Zhang, *Exact emulation of an output queuing switch by a combined input and output queuing switch*, in proceedings of IEEE/IFIP IWQoS, 1998, pp. 218-224

[28] D. Gale and L. S. Shapley, *College admissions and the stability of marriage*, American Mathematical Monthly, vol. 69, pp. 9-15, 1962

[29] H. J. Chao, C. H. Lam and E. Oki, *Broadband Packet Switching Technologies: A practical guide to ATM switches and IP routers*, Chapter 4, published by Wiley

[30] G. Appenzeller, I. Keslassy and N. McKeown, *Sizing router buffers*, in proceedings of SIGCOMM 2004

[31] S. Iyer and N. McKeown, *Making parallel packet switches practical*, in proceedings of IEEE INFOCOM 2001

[32] M. Yang and S. Q. Zheng, *An efficient scheduling algorithms for CIOQ switches with space-division multiplexing expansion*, in proceedings of IEEE INFOCOM 2003

[33] S. Floyd and V. Jacobson, *Random Early Detection gateways for congestion avoidance*, IEEE/ACM Transactions on Networking, August 1993

[34] S. Iyer, R. R. Kompella and N. McKeown, *Analysis of a Memory Architecture for Fast Packet Buffers*, IEEE Workshop on High Performance Switching and Routing, May 2001