This is a repository copy of *A Clos-Network Switch Architecture based on Partially-Buffered Crossbar Fabrics*.

White Rose Research Online URL for this paper:
http://eprints.whiterose.ac.uk/105248/

Version: Accepted Version

**Proceedings Paper:**

https://doi.org/10.1109/HOTI.2016.020

© 2016 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

**Reuse**
Unless indicated otherwise, fulltext items are protected by copyright with all rights reserved. The copyright exception in section 29 of the Copyright, Designs and Patents Act 1988 allows the making of a single copy solely for the purpose of non-commercial research or private study within the limits of fair dealing. The publisher or other rights-holder may allow further reproduction and re-use of this version - refer to the White Rose Research Online record for this item. Where records identify the publisher as the copyright holder, users can verify any specific terms of use on the publisher’s website.

**Takedown**
If you consider content in White Rose Research Online to be in breach of UK law, please notify us by emailing eprints@whiterose.ac.uk including the URL of the record and the reason for the withdrawal request.
A Clos-Network Switch Architecture based on Partially-Buffered Crossbar Fabrics

Fadoua Hassen
Lotfi Mhamdi
School of Electronic and Electrical Engineering
University of Leeds, UK
Email: {elfha, L.Mhamdi}@leeds.ac.uk

Abstract—Modern Data Center Networks (DCNs) that scale to thousands of servers require high performance switches/routers to handle high traffic loads with minimum delays. Today’s switches need be scalable, have good performance and -more importantly- be cost-effective. This paper describes a novel three-stage Clos-network switching fabric with partially-buffered crossbar modules and different scheduling algorithms. Compared to conventional fully buffered and buffer-less switches, the proposed architecture fits a nice model between both designs and takes the best of both: i) less hardware requirements which considerably reduces both the cost and the implementation complexity, ii) the existence of few internal buffers allows for simple and high-performance scheduling. Two alternative scheduling algorithms are presented. The first is scalable, it disperses the control function over multiple switching elements in the Clos-network. The second is simpler. It places some control on a central scheduler to ensure an ordered packets delivery. Simulations for various switch settings and traffic profiles have shown that the proposed architecture is scalable. It maintains high throughput, low latency performance for less hardware used.

Index Terms—Data Center Networks switching fabric, Clos-network, Partially-buffered, Packet scheduling

I. INTRODUCTION

The tremendous growth rate of Data Center Networks is key motivation for developing high performance switches/routers. DCN switching fabrics are evolving for increased scalability, higher performance and less complexity. They are using more large-capacity switches/routers to build large-scale networks. However, the scalability of the DCN fabric is closely related to the capability of its switching elements to expand and to scale to future demand. A scalable switching architecture should be able to expand with simple and repeatable design. It also needs to accommodate the increased traffic flows or new traffic patterns with no impact on the application work flows as well as the cost per port.

Single stage switches although perfectly non-blocking and simple to implement, prove to be capacity-restricted. First, scaling single-stage switches in the number of ports is not possible as the building costs become more and more prohibitive. Besides, both the complexity of the switching hardware and the scheduling algorithms depend on the square of the number of the switch ports. Moreover, packaging techniques and the current VLSI (very Large Scale Integrated circuits) technology, allow only the deployment of switches with single-stage crossbars of few Tb/s [1]. To overcome the limitations of a single-stage design, multistage switching architectures have been proposed. They are a good alternative to build large cost-effective switches. Clos-network architectures are among the most used patterns. They allow multiple crossbars and/or memory sets to be connected in a non-blocking way. A Clos switch is described according to the nature of its switching modules. While a bufferless module is referred to as Space (S), a buffered crossbar is called Memory (M).

Bufferless Clos switches appear to be cheap and attractive. However, they reach their limitations with large port counts and high data rates. This is mainly because they rely on centralized schedulers to perform a global path-allocation. The scheduler maintains records for all ports and assigns conflict-free paths to packets. Increasing the number of ports of the switch results in high unfeasible scheduling complexity and makes bufferless switches valency-sensitive. Buffered Clos-network switches solve the complexity and scalability issues that bufferless architectures encounter. Internal buffers offer some overprovisioning of the traffic admissions, allow to implement distributed scheduling and enhance performance. Central Modules (CMs) of a fully buffered Clos-network switches keep $N^2$ internal buffers ($N$ is the number of I/O links of a single CM). A major weakness of buffered multistage Clos-network architectures comes from the fact that the number of internal buffers grows quadratically with the switch ports count. This makes Memory-Memory-Memory switch (MMM) expensive and less appealing for large-scale switches/routers.

In order to substantially reduce the comparatively large number of the crosspoint buffers, the Partially Buffered Crossbar (PBC) was introduced [2]. Only a small number of internal buffers with small capacities are used rather than $N^2$ crosspoint queues. The scheduling process in PBCs is a mixture of unbuffered and buffered crossbar scheduling. PBC switches requires some extra scheduling logic (as compared to buffered switches) in order to successfully manage the limited storing space inside the fabric.

In this paper, we propose and evaluate a partially buffered Clos-network switching architecture with a small number of buffers and simple scheduling scheme. At packets arrival, requests are forwarded to the central modules. They are granted by a scheduling sub-system that manages the credits per output link and allocates space to requesting packets. The scheduling process is distributed. It can be pipelined and the scheduling phases might be run in parallel. In subsequent

Partially-buffered
Packet scheduling
in a three-stage Clos network with bufferless middle buffers at all stages to temporarily store unscheduled packets. It absorbs all contentions and offers a straightforward alternative to scale up the single-stage buffered switch architecture. An MMM packet switch is a performance, buffered SEs were added to several stages of the switching architecture. An MMM switch as described in [5] provides full output contention and the global path allocation simultaneously. The MSM switch as described in [5] provides full output contention and the global path allocation simultaneously.

In this paper, we suggest a buffered Clos-network switch with only few internal buffers in the CMs. Our goal is to design a cost-effective switch that provides performance comparable to an MMM switch and defeats the shortcomings of conventional MSM switches. Throughout the paper, we will refer to our design as the Partially Buffered Clos switch, PBClos. The proposal differs from [3] in many ways, mainly: (1) we use $B$ internal buffers per CM, where $B \ll k$, (2) we use physically separate memories inside the CM crossbar fabrics instead of costly shared memories, and (3), we simplify the scheduling process.

III. Architectural Design Challenges

A. Over-provisioning in conventional MMM

In TABLE I, we compare some of the HW requirements of different switching architectures. Conventional MMM switches like those proposed in [1] and [10], adopt fully buffered modules in the central stage of the Clos. For any couple of Input Module (IM) and Output Module (OM), there is a dedicated crosspoint buffer in a CM, which capacity may vary from one packet to several packets. Scaling up a crosspoint buffer size to several packets, boosts the switch performance especially if the traffic arrivals are irregular or bursty [7] [3]. Provided there is enough space, packets often find a room in the middle stage crossbars where they wait to exit CMs to the output stage of the switch instead of being...
Discarded [1]. Note that, holding many internal buffers in the middle-stage modules or/and increasing their capacities, is carried out to cater for the amount of packets dispatched at the input scheduling phase. However, both measures are without interest in the output scheduling phase. Actually, with no speedup used, an output scheduler serves one packet from one internal buffer among those dedicated to an LC link at a time.

Packet scheduling is a paramount task in any switching architecture. It strongly depends on the design itself. More importantly, it affects the switch performance and the implementation complexity. In a MMM switch, we tend to simplify the scheduling process at the expense of the hardware cost by distributing the arbitration function over different units. The rule of thumb says that the more intermediate buffering is available, the less a central control is needed. In bufferless MSM switch [5], a centralized scheduler must be able to find a conflict-free matching between a large set of requests and output links.

The goal of this work is to contribute towards two things:
- A less expensive switching architecture.
- A good scheduling that provides high performance.

We suggest the use of only few crosspoint buffers (much less than in common MMMs). We specify that an internal buffer can store only one packet. In addition to the straightforward distributed approach, we present a frame-based scheduling whereby we simplify the central scheduler complexity as compared to MSM switch.

B. In-order packets forwarding

Buffered multistage packet switches must preserve two primordial aspects: high performance (namely the switch throughput and the overall packets latency) and its ability to deliver packets in order. A three-stage Clos interconnect provides multiple paths to route packets from the input stage modules to the middle stage switching elements, and from the central modules to the output stage buffers. Adding buffers to the central stage of the Clos-network switching architecture potentially leads to an out-of-order packets delivery. In general, cells are scheduled to cross different CMs. They experience variable queuing delays in the buffered fabric and reach to their OMs disorganized. The new design trends rise many challenges and questions. Can we provide performance comparable to fully buffered Clos switches using only few internal buffers in the CMs? Is it possible to forward packets their output ports without order disruption and with no performance degradation? Can all of the aforementioned objectives be achieved with a simple scheduling scheme? This paper tackles these issues and gives alternatives to all these objectives combined.

IV. PARTIALLY-BUFFERED CLOS SWITCHING ARCHITECTURE

A high-level abstraction of the switch architecture is depicted in Fig.1. Ingress Line Cards (ILCs) do the variable-size packets segmentation into fixed-size cells before they enter the IMs. Egress Line Cards (ELCs) reassemble received cells into packets and finally send them out of the PBClos switch.

A. High-level architecture terminology

We describe a three-stage Clos-network switch made of buffered modules that operates on fixed sized cells. Packets of variable length get segmented into fixed size cells while inside the switch and are reassembled back to packets upon their exit. There are \( k \) IMs, each has a dimension \( (n \times m) \). IMs are connected to the Central Modules (CMs) by means of \( m \) links \( \text{LI}(i, r) \) (where \( i = 0, \ldots, k-1 \) and \( r = 1, \ldots, m \)). Each IM consists of \( k \) Virtual Output Queues (VOQs) to eliminate the HoL blocking (\( N = n.k \)). Every queue stores cells coming from an input port in IM(\( i \)) to an output port OP(\( j, h \)). CM switches are Partially Buffered Crossbars (PBCs) [2]. A CM(\( r \)) has \( k \) output links, each of which is denoted as LC(\( r, j \)) that connects it to OM(\( j \)). An OM(\( j \)) has \( n \) OPs to which is associated an output buffer. An output buffer can receive at most \( m \) packets and forward one packet to the output line at every time slot.

Unlike traditional crosspoint queued crossbars, a PBC crossbar contains only a small number of internal buffers; \( B \ll k \). A total of \( B \) separate internal buffers are maintained per output link, LC. For simplicity, we consider the Clos-network’s expansion factor to be \( \frac{m}{n} = 1 \), which makes the architecture a Benes network: the lowest-cost practical non-blocking fabric. However, all the descriptions remain valid for any non-blocking Clos-network settings.

B. Scheduling in the PBClos: Distributed Approach

Packets scheduling can be either centralized or distributed. In some switching architectures, we adopt both approaches to trade-off complexity and performance. Distributed arbiters avoid the scalability limitation of a centralized approach by dispersing the scheduling function over the switching modules of the Clos-network. We first opt for a distributed scheduling. We use \( n \) Accept Schedulers (ASs) in every IM, one per input port and \( k \) Output Schedulers (OSs) in a CM; one per LC link.

Fig. 1: \( N \times N \) Three-stage PBClos switch architecture
TABLE I: Comparison between the different switching architectures

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Type of the fabric</td>
<td>bufferless</td>
<td>buffered</td>
<td>buffered</td>
<td>partially-buffered</td>
</tr>
<tr>
<td>Type of the scheduling</td>
<td>centralized</td>
<td>centralized</td>
<td>distributed</td>
<td>distributed (IV-B)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>centralized (IV-C)</td>
</tr>
<tr>
<td>Number of crosspoint buffers</td>
<td>NA</td>
<td>$k^2$</td>
<td>$k^2$</td>
<td>$k.B^*$</td>
</tr>
<tr>
<td>Size of crosspoint buffer</td>
<td>NA</td>
<td>1 packet</td>
<td>12 packets</td>
<td>1 packet</td>
</tr>
<tr>
<td>Type of internal memory</td>
<td>NA</td>
<td>separate memory banks</td>
<td>shared memory</td>
<td>separate memory banks</td>
</tr>
<tr>
<td>Size of the centralized scheduler</td>
<td>$O(m.N)^{\dagger}$</td>
<td>$O(\sqrt{k})$</td>
<td>NA</td>
<td>$O(\sqrt{k})$ $^\star$</td>
</tr>
<tr>
<td>In-sequence packets delivery</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes $^\clubsuit$</td>
</tr>
</tbody>
</table>

$^\dagger$ ($B \ll k$)

$^\ddagger$ $N = n.k$

$^\star$ The explanation is the same as done in reference [1].

$^\clubsuit$ If centralized frame-based scheduling is used.

Fig. 2: PBClos switch with distributed scheduling. Grants are forwarded from CMs to active VOQs. A VOQ that receives more than one grant from a CM (i.e. The CM has room to house a packet to the corresponding OP), accepts one grant in a RR fashion.

Fig. 3: Scheduling in the central stage PBC switches.

Fig. 2 depicts the succession of events in one time slot. Up to $n$ new packets arrive to an IM and get stored to their corresponding queues. As in [10], the path allocation is made in the upstream direction. An AS selects the first non-empty VOQ that appears in its RR selector and marks it as active. The distributed scheduling provides no visibility on the status of the internal buffers in the middle stage fabric. Hence, an active VOQ sends requests to all CMs. At the starting of the scheduling cycle, the source/destination tuple of the HoL packets are copied to the Request Matrix of a LI link (event 1 from active VOQs to the Request Matrix). A Request Matrix of LI($i, r$) serves to map all active requests that come to a CM($r$) through LI($i, r$). The mapping action does not include any cells transfer. It just sends requests to trigger the PBC modules’ schedulers in the next pipeline stage of the scheduling. The output schedulers send grants to requests (event 2, from the CMs to VOQs) and ultimately, accept schedulers select, each, one grant (event 3).

Grant and Accept scheduling:

The internal buffers run at the external line rate. The choice for isolated buffers is to make sure that we use low bandwidth and to avoid the need for high-throughput shared memories used in [2], [3]. Fig. 3 shows the grant mechanism in a central module. Each internal buffer can house at maximum one packet. Output schedulers hold Credit Queues ($CQs$) of size $B$ to record the number of empty internal buffers per LC link. Credits are decremented at each time a grant is sent to a mapped request in the Request Matrix (event 3 in Fig.3) and they get incremented at the end of the scheduling cycle; when packets exit the fabric. At the end of the grant phase, grants are directly forwarded to active VOQs in the IMs (event 4). If an active VOQ($i, j$) receives more than one grant from different CMs, the associated AS($i$) accepts the one that appears in the RR priority and drops unaccepted grants and the packet is transferred to its corresponding CM (event 5). Note that, a matched LI link is marked as reserved to avoid clashes. It automatically gets excluded from the arbitration set of the remaining active VOQs in other IMs.

Mapping requests, grant phase and the output arbitration are all independent. They can be pipelined and run in a parallel way. While a scheduling cycle ends and cells leave the PBC modules, the buffers reservation of freshly mapped requests takes over. Packets that arrive to the output stage of the Clos-network find their ways to their OPs. Although simple to implement and practical, the distributed control end up routing packets of the same flow across several CMs. Packets remain in the internal buffers until it is their turn to exit the fabric.
switch with fully buffered CMs, dedicated Crosspoint Queues scheduling in a CMs. The selected VOMQ broadcasts packets to Ties are broken in a RR fashion and only one input queue is full input module, the controller marks as eligible, VOMQs with controller to track the buffers availability in all CMs. For every shared between all LI links (i.e. IMs). We use a central LC link (i.e. OMs) is very limited. A buffer can accommodate only one packet. Besides, the buffering space is logically shared between all IMs. Moreover, we impose no restriction on how to locate the CQs in the different CMs. Instead of entrusting the output scheduling to the central controller, we use simple and distributed output schedulers as we described in IV-B. Packets of the same batch arrive to internal buffers at the same time slot. To maintain in-sequence packets delivery, we employ a First-Come-First-Served (FCFS) output scheduling policy.

C. Centralized control

In the context of buffered switching architectures, a distributed scheduling is a straightforward option. The scheduling phases might be pipelined for faster execution. More importantly, the distributed approach do not bound the scalability of the switch as a dispersed arbitration decouples the architecture from the control plane and proves more effective for large high-performance switch design. All the same the distributed scheduling do not assure in-order packets delivery. Instead of adding re-ordering buffers to the first or last stage of the Clos-network [7] [9], we alter the organization of the input buffers in IMs and we proceed to a frame scheduling rather than a packet-per-packet scheduling.

IMs are organized into Virtual Output Module Queues (VOMQs), where packets are enqueued according to their destination OMs. VOMQs are shared memories that can receive packets from up to $n$ input ports at a time and send up to $m$ packets to the different CMs. The choice for VOMQ queuing is done in accordance with the frame scheduling itself. The idea is as follows: We always forward a flow of packets through the same path to avoid any resequencing logic within the flow. The approach is similar to hash-based routing solutions, where several hash functions are used to assign routes to the different packet flows. However, the ultimate purpose of the two schemes are different (load-balancing for the hash-based routing versus ordered packets delivery for the frame scheduling). The arbitration in the PBClos switch is made of two phases: The input scheduling which consists on sending packets from the VOMQs to the internal buffers in the PBC modules and the output scheduling that concerns with forwarding packets from CMs to their OPs.

1) Input scheduling: Fig.4 gives an example for the frame scheduling in a $9 \times 9$ PBClos switch. $m$ consecutive packets in a VOMQ make a frame. Unlike previous proposals [8] [1] [3], where dedicated crosspoint buffers with variable capacity are available, the current switch design imposes several constraints: In a PBC module, the number of internal buffers per LC link (i.e. OMs) is very limited. A buffer can accommodate only one packet. Besides, the buffering space is logically shared between all LI links (i.e. IMs). We use a central controller to track the buffers availability in all CMs. For every input module, the controller marks as eligible, VOMQs with full frames for which there is enough space in the PBC fabric. Ties are broken in a RR fashion and only one input queue is elected at a time. The selected VOMQ broadcasts packets to CMs.

2) Output scheduling: In [1], authors described a MMM switch with fully buffered CMs, dedicated Crosspoint Queues (CQs) and a centralized batch scheduling. In addition to the VOMQs selection, the scheduler processes the output scheduling. It chooses $k$ non-empty CQ one from every set of internal buffers of an output link LC. The scheduler applies the same configuration to all CMs and packets are dequeued from the CQs in the different CMs. In this way, a batch is reconstructed again at the corresponding OM. We use physically separate CQs in the PBC fabric to keep the memory requirements low. Subsequently, we can embed the output schedulers and get more compact CMs. We call column of CQs, the set of internal buffers that belong to an LC. CQs might be located anywhere in the column since the access to the buffering space is logically shared between all IMs. Moreover, we impose no restriction on how to locate the CQs in the different CMs. Instead of entrusting the output scheduling to the central controller, we use simple and distributed output schedulers as we described in IV-B. Packets of the same batch arrive to internal buffers at the same time slot. To maintain in-sequence packets delivery, we employ a First-Come-First-Served (FCFS) output scheduling policy.

V. Performance Analysis

In this section we evaluate the throughput and delay performance of the PBClos switch under different scenarios and we compare them with MSM switch and different MMM architectures that have been described in [1], [7] and [8]. All simulations are implemented in an event-driven switch simulator. The switch size is 256, if not explicitly mentioned and the simulation time is $10^6$ time slots. Statistics start to be collected from the $10^5$ time slots to ignore any transient behaviour of the simulation. The current switch is interesting in the way it provides more degrees of design freedom. Varying the number of internal buffers helps monitoring the cost, complexity and performance of the switch. Considering full number of crosspoint buffers in the CMs and two different scheduling schemes, maps the switch to two previous proposals: Three-Stage Buffered Clos-network Switch (TSBCS) [1] (if the frame-based scheduling is used) and Scheduled
Fabric (SF) [3] [10] with crosspoints size $b = 1$ (if distributed scheduling is adopted). We use the notation PBClos-centr for the switch with a frame-based scheduling and PBClos-dist for distributed scheduling. MMM is the switching architecture suggested in [8] and xbuff is the size of its crosspoint buffers.

The first set of simulations is performed for uniform traffic: Bernoulli arrivals with evenly distributed destinations and uniform arrivals of bursts with a default burst size of 10 packets. Bursty traffic can be modeled as an on-off process. The burst of packets that comes to a switch input port during the on period is in destination to the same output port. Destinations among bursts are uniformly distributed among output ports. The delay performance presents the sojourn time of packets through the switch fabric. It is estimated by averaging over all queueing delays measured during the simulation. Fig.5(a) shows results for Bernoulli $i.i.d$ arrivals. PBClos has delay performance that is in between what MSM and MMM provide. The switch performs poorly under light loads, but deals better with medium to high loads. With distributed scheduling, the switch offers lower latency than when a frame-based scheme is used. The reason is that VOMQs need wait for full frames before they are eligible for scheduling. Fig.5(b) is a clearer view of the delay-throughput curves shown in Fig.5(a). Increasing the number of crosspoint buffers contributes towards higher throughput. For $256 \times 256$ switch, setting $B = \frac{k}{2} = 8$ queues/LC link and using a distributed arbiters, make PBClos achieve $90\%$ throughput versus $95\%$ for the SF. On the other hand, we save half of the buffering amount used in the middle stage of the SF architecture.

Using the same number of internal buffers per output link in CMs and a frame-based scheduling, PBClos provides full throughput and slightly higher delay than TSBCS. Fig.6(a)
shows that under uniformly destined bursty traffic, PBClos outperforms MSM and MMM [8]. Note that a frame-based scheduling with as few as 4 internal buffers per LC link achieves full throughput as presented in Fig.6(b). On the whole, we observe that the variation of the average latency of the PBClos switch is smooth. It is not affected by the switch valency while that of MSM is much influenced.

The uniform arrival patterns do not reflect a real traffic. They just help trace the switching architecture’s response assuming the best case scenario. For the next set of evaluations, we consider non-uniform traffic: hot-spot, skewed [2] and diagonal traffic. Fig.7(a) depicts the latency of switches under hot-spot traffic. The bufferless architecture do not adopt well with non-uniform traffic. CRRD with 4 iterations achieves as low throughput as 70%. The experimental results show that holding $k/2$ buffers/LC in the central-stage modules of the PBClos switch, still provides high throughput performance.

In Fig.7(b), we compare the performance of the PBClos varying the switch valency under hot-spot packet arrivals. Unlike a distributed scheduling, the frame-based scheme is not as efficient under light to medium loads as it is under heavy loads. It results in an initial waiting time that is proportional to the switch size (the central scheduler waits for the construction of larger frames). The distributed scheduling is insensitive to the switch size, but its throughput saturates at 90% for a 64 $\times$ 64 switch and 97% for a 256-ports switch. In Fig8(a) and 8(b), we plot the throughput of switches respectively under unbalanced and diagonal traffic. The throughput of PBClos increases as we increase the number of crosspoint buffers in the central modules. We observe that the throughput is marginally affected when a distributed packets scheduling is adopted. Under diagonal traffic, both scheduling approaches
make the switch achieve equally high throughput even when \( B = k/2 \).

VI. CONCLUSION

In this paper, we proposed a practical and cost-effective multistage switching architecture (PBClos) that relies on partially-buffered crossbars with capacity-limited queues. PB-Clos comes in between bufferless and fully-buffered architectures, and takes the best of both designs. We evaluated the switch performance using multiple, independent single-resource schedulers that work in a pipeline. In spite of its good performance, the distributed scheme do not guarantee an ordered packets delivery. Thus, we suggest a frame-based scheduling that distributes packets of a flow over all CMs of the Clos and builds back a frame at the OMs avoiding the need for costly resequencing buffers. This discipline cumulates the delay when the switch is light-loaded. However it gives good performance and proves to be scalable under heavy loads. Work in progress includes research on how to lower the initial delay of the frame-based scheduling. We also investigate alternatives to shared memory in the input stage to simplify the hardware structure of the switch.

REFERENCES