An Asynchronous Routing Algorithm for Clos Networks

Wei Song and Doug Edwards
School of Computer Science, University of Manchester
Oxford Road, Manchester, M13 9PL, United Kingdom
{songw, doug}@cs.man.ac.uk

Abstract

Clos networks provide the theoretically optimal solution to build high-radix switches. This paper proposes the first-ever asynchronous routing algorithm for general three-stage Clos networks. It outperforms the synchronous concurrent round-robin dispatching algorithm in behavior level simulations. In a 32-port Clos network utilizing this algorithm, paths are reserved in 6.2 ns and released 3.9 ns.

Type of submission: regular paper
Corresponding author: Wei Song
Address:
IT 302, School of Computer Science, the University of Manchester
Oxford Road, Manchester M13 9PL, United Kingdom
Email: songw@cs.man.ac.uk
Tel: +44 (0)161 275 6292
Fax: +44 (0)161 275 6204
Category: Concurrency issues in Systems on Chips, massively parallel architectures, networks on chip, task and communication scheduling, resource, memory and power management, fault-tolerance and Quality of Service issues

An Asynchronous Routing Algorithm for Clos Networks

1 Introduction

Clos networks are a class of multi-stage switching networks proposed in last 50s [1]. It provides the theoretically optimal solution to build high-radix switches when the requirement exceeds the capacity of a feasible single-stage crossbar switch. Although the emergence of VLSI technologies intensively reduces area of a single crosspoint and enlarges the capability of a crossbar, the insatiably desire for speed and performance always pushes router designs to the very limit. Clos networks still live in cutting edge designs.

The early telephone networks are circuit switched network where switches are statically configured. The later asynchronous transfer mode (ATM) networks and IP networks achieve higher throughput using packet switching technologies [2], which require switching networks to be dynamically reconfigured. The random dispatching algorithm used in the ATLANTA chip [3] demonstrates a feasible way of dynamically reconfiguring a three-stage Clos network by heuristic algorithms. From then on, numerous routing algorithms have been proposed to improve the throughput performance [4]–[9]. A Clos network designed for current optical backbone networks has already achieved peta-bit throughput performance [10].

Clos networks also find their utilization in the field of intra- and inter-chip interconnection networks. Transistor scaling increases the pin bandwidth available for a router chip and the wire resource in on-chip networks. However, it is found out that a router with many narrow ports is more efficient than a router with a few wide ports [11], [12]. A folded-Clos network is used in the Cray BlackWidow multiprocessor to achieve high bandwidth communications [13] and a Beneš network (a case of multi-stage Clos network) has been used in a router for on-chip networks to provide delay guaranteed services [14].

Asynchronous quasi delay insensitive (QDI) circuits are well known for their low power consumption and tolerance to process, voltage and temperature variations. Considering the high power consumption of current communication fabric and the increasing process variation, it is beneficial to implement high-radix routers asynchronously. However, dynamically reconfiguring a three-stage Clos network is complicated and area consuming even for synchronous circuits. Furthermore, asynchronous arbiters are inadequate to allocate multiple resources concurrently until a recently proposed multi-resource arbiter [15]–[17].

In this paper, the first asynchronous routing algorithm for general three-stage Clos networks is proposed and implemented. The remainder of this paper is organized as follows: Section 2 introduces some fundamental concepts of Clos networks. Section 3 analyzes two classic routing algorithms used in synchronous Clos networks and proposed the asynchronous routing algorithm. Section 4 evaluates the performance of these routing algorithms on behavioral level. Section 5 demonstrates the detailed implementation of the routing algorithm and section 6 shows the hardware performance. Finally the paper is concluded in section 7.

2 Clos Network

Fig. 1 shows a three-stage Clos network. The terminology used in this paper is as follows.

IM	Input module at the first stage.
CM	Central module at the second stage.
OM	Output module at the third stage.
n	Number of input ports (IPs)/OPs in each IM/OM.
k	Number of IMs/OMs.
m	Number of CMs.
i	Index of IMs ($0 \leq i < k$).
j	Index of OMs ($0 \leq j < k$).
r	Index of CMs ($0 \leq r < m$).
h	Index of IPs/OPs in an IM/OM ($0 \leq h < n$).
IM(i)	The $(i+1)$th IM.
OM(j)	The $(j+1)$th OM.
CM(r)	The $(r+1)$th CM.
IP(i,h)	The $(h+1)$th IP in IM(i).
OP(j,h)	The $(h+1)$th OP in OM(j).
LI(i,r)	The link between IM(i) and CM(r).
LO(r,j)	The link between CM(r) and OM(j).
$\mathbf{C}(n,k,m)$	A Clos network has m CMs and k IMs/OMs with n IPs/OPs.
N	The total number of IPs/OPs ($N = nk$).

The first stage contains k IMs, each of which is an $n \times m$ reconfigurable crossbar. CMs in the second stage are statically connected with IMs and each CM is a $k \times k$ reconfigurable crossbar. The third stage contains k OMs, each of which is an $m \times n$ reconfigurable crossbar statically connected with CMs.

Switching networks could be classified into three categories [2]: *Blocking*: the switches have possible connection states such that an available input/output pair cannot be connected because of an internal blocking. *Strict Non-Blocking (SNB)*: the switches ensure the connection of any available input/output pairs without

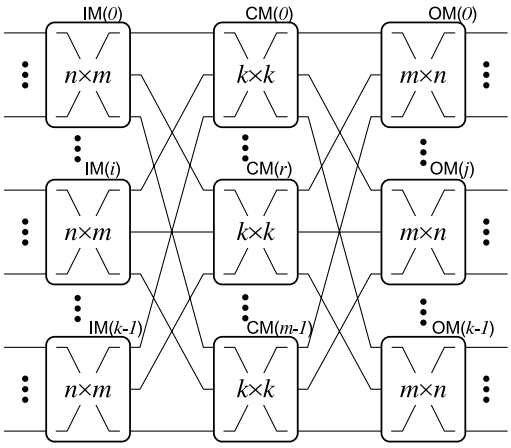


Fig. 1. A three-stage Clos network $\mathbf{C}(n, k, m)$

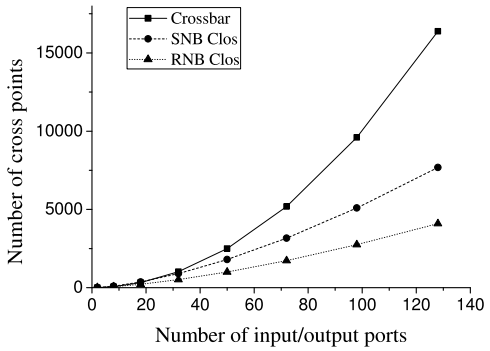


Fig. 2. Area of crossbar and Clos network

altering any established connections. *Rearrangeable Non-blocking (RNB)*: the switches ensure the connection of any available input/output pairs with possible modification of established connections. A three-stage Clos network could be an SNB network or a RNB network depending on the number of CMs. An SNB Clos network requires more than $2n-1$ CMs ($m \geq 2n-1$), while a RNB network needs only n CMs ($m = n$) [2].

The major advantage of Clos networks over crossbars is area efficiency. The area of a switching network is linear with the number of crosspoints inside it. For a crossbar with N input/output ports the area is linear to the cost C.

$$C_{CB} = N^2$$

Both SNB and RNB Clos networks have the minimal cost when $k = \sqrt{2N}$.

$$C_{Clos,SNB} \geq 2(2N)^{1.5} - 4N$$

$$C_{Clos,RNB} \geq (2N)^{1.5}$$

Fig. 2 demonstrates the area of crossbars and Clos networks with various number of ports. Both SNB and RNB Clos networks reduce area overhead significantly but RNB Clos networks have the minimal area.

There are two classes of routing algorithms for Clos networks [5]: *Optimized algorithms*, which provide guaranteed results for all matches but with a higher complexity in time or implementation. *Heuristic algorithms*, which provide all or part of matches in much lower time complexity. Although optimized algorithms guarantee the connection of any IP/OP pairs, they often require the global view of all modules and use more time to reconfigure. The majority of current dynamically reconfigurable Clos networks utilize heuristic algorithms [3]–[9].

Buffer insertion is a usual way of improving performance in Clos networks. According to the stage where buffer is inserted, a Clos network could be a space-space-space (S^3) network without buffer, a memory-space-memory (MSM) network with buffer insertion in IMs and OMs, or a space-memory-space (SMS) network with buffer insertion in CMs. S^3 networks are easier to control than the other two structures but have the worst performance. Normally SMS networks show better performance than MSM networks because the buffers in CMs resolve the contention between IMs and CMs; however, this scheme requires a re-sequencing function in OMs because data stored in buffers are issued to OMs out-of-order. MSM is the most utilized scheme. Buffers in IMs and OMs improve network throughput and eliminate the out-of-order problem. However, the OMs are required to speed-up m times to avoid throughput degradation (the detailed comparison of buffer insertion schemes and memory speed-up could be found in [2], [9]).

In this paper, we are proposing the first asynchronous heuristic routing algorithm for general three-stage S^3 Clos networks. It is possible to modify the algorithm to control MSM networks in the future.

3 Routing Algorithms

All heuristic algorithms process a request from IP(i_1, h_1) to OP(j_2, h_2) in two stages [8]: module matching, selecting a CM and reserving the corresponding links LI(i_1, r) and LO(r, j_2); and port matching, connecting IP(i_1, h_1) to LI(i_1, r), LI(i_1, r) to LO(r, j_2) and LO(r, j_2) to OP(j_2, h_2). As all modules in an S^3 Clos network are crossbars, the port matching process is block-free. On the contrary, CMs are shared by all IMs and OMs. Any sub-optimized CM allocation leads to internal block states. The module matching procedure determines the overall performance of an algorithm. As a result, the algorithm used in module matching, namely dispatching algorithm, is the key research issue.

Before introducing the first asynchronous routing algorithm, we need to review the classic dispatching algorithms used in synchronous Clos networks.

3.1 Random Dispatching (RD)

The data transmitted in Clos networks are routed in units of cell, a small fraction of a packet with fixed size. Multiple cells are transmitted synchronously from IMs to OMs in one cell time. The reconfiguration of switches proceeds concurrently with cell transmission in

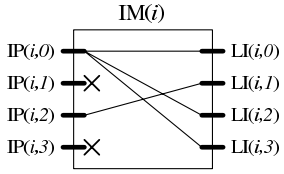


Fig. 3. Example of uneven matching in IMs

a pipelined manner. The new configuration generated in current cell time takes effect in the next cell time. A cell time could be one clock cycle or multiple cycles depending on the complexity of the routing algorithm in use.

The random dispatching (RD) algorithm utilized in the ATLANTA router chip [3] is the predecessor of many current heuristic algorithms. The Clos network in ATLANTA is an MSM network and the RD algorithm randomly dispatches packets from buffered IMs to OMs. This algorithm can be used in S^3 Clos networks. A simplified description is as follows [4]:

- Phase 1: Matching within IMs.
  - Step 1: Every non-idle IP(i, h) sends requests to all LIs in IM(i).
  - Step 2: Each LI(i, r) randomly selects an requesting IP(i, h) and up to m requests are proposed to CMs.
- Phase 2: Matching between IMs and CMs.
  - Step 1: CM(r) receives up to k requests from IMs. According to the availability of LOs, up to k requests are granted.
  - Step 2: IMs receive grants from CMs and send the corresponding cells in the next cell time. The cells without grants need to request again in the next cell time.

RD dispatches incoming cells in two phases: matching in IMs and matching between IMs and CMs. In phase 1, each LI randomly selects a non-idle IP in the same IM. Since one IM has m output ports, up to m requests are sent to CMs. The requests are statically distributed to all CMs in the same way as the Clos network itself. In phase 2, a CM receives up to k requests. If multiple requests ask for the same LO, one of them is selected randomly. Consequently up to k requests are granted. The granted IPs send cells in the next cell time while other IPs enter the next round of requesting.

The RD algorithm cannot achieve high switch throughput due to its matching within IMs. Since LIs select IPs independently, some IPs could be selected by multiple LIs while other IPs are rejected by all LIs. An example is shown in Fig. 3. Suppose all IPs in IM(i) are requesting and all LIs are available, IP($i, 0$) has been selected by LI($i, 0$), LI($i, 2$) and LI($i, 3$) leaving IP($i, 1$) and IP($i, 3$) unselected. Uneven matching is therefore generated in phase 1.

3.2 Concurrent Round-Robin Dispatching (CRRD)

The concurrent round-robin dispatching (CRRD) algorithm [4] resolves the uneven matching problem inside IMs. It is claimed that using CRRD in MSM Clos networks achieves 100% throughput under uniform traffic.

Instead of using random arbiters, CRRD uses round-robin arbiters. In each IM, every LI has an output-link round-robin arbiter and every IP has an input-port round-robin arbiter. Each CM has k round-robin arbiters on k LOs respectively. All round-robin arbiters are independent. A simplified description of CRRD is as follows:

- Phase 1: Matching within IMs.
  - First iteration
    - Step 1: Each non-idle IP sends requests to all output-link arbiters.
    - Step 2: Each output-link arbiter selects an IP using its own pointer.
    - Step 3: Each non-idle IP accepts one of the grants from all output-link arbiters using its round-robin arbiter.
  - ith iteration ($i > 1$)
    - Step 1: Each unmatched IP sends requests to all unmatched output-link arbiters.
    - Step 2 and 3: The same as the first iteration.
- Phase 2: Matching between IMs and CMs.
  - Step 1: Each matched LI sends a request to CMs. Each round-robin arbiter in CMs selects one request and sends a grant to the IM.
  - Step 2: IMs receive grants from CMs and send the corresponding cells in the next cell time. The cells without grants request again in the next cell time.

Similar to RD, the dispatching procedure of CRRD is also divided into two phases but phase 1 contains several iterations. In each iteration, IMs try to match as many unmatched IPs as possible to unoccupied LIs in a round-robin fashion. Unlike RD, CRRD guarantees that an IP is selected by only one LI in phase 1. The phase 2 of CRRD is similar to that of RD except that round-robin arbiters are used.

Fig. 4 shows an example of one iteration in CRRD. IM(i) has the same initial states as in Fig. 3. All IPs have received new packets and all LIs are available. As shown in Fig. 4a, each IP sends requests to all available LIs. Then each LI grants one IP using its output-link round-robin arbiter. Fig. 4b shows the same grant results as the RD example in Fig. 3. However, the step 3 of CRRD forces all IPs to accept only one LI. In this way, the IPs unmatched in the current iteration can try again in the next iteration as shown in Fig. 4d.

CRRD improves switch throughput thanks to its balanced matching between IPs and LIs. The number of iterations is limited by the cell time. It is required to extend the cell time until the balanced matching is achieved by enough iterations.

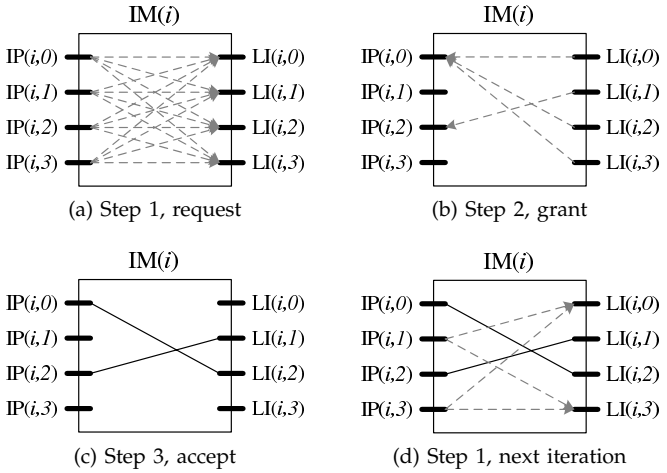


Fig. 4. Example of iterations in CRRD

3.3 Asynchronous Dispatching (AD)

Reconfiguring an asynchronous Clos network has some fundamental differences with its synchronous counterpart:

- Incoming packets arrived asynchronously and are handled asynchronously. There is no need to divide them into cells and a Clos network is reconfigured for packets instead of cells.
- Modules are event-driven without synchronization. The dispatching algorithm cannot proceed in two sequential phases but two concurrent sub-algorithms in IMs and CMs.
- The arbiters in asynchronous circuits are also event-driven. There is no need to keep a pointer in each asynchronous arbiter as the round-robin one.
- The step 3 in phase 1 of CRRD generates orphans. Grants from arbiters are normally used to generate acknowledgement. When output-link arbiters send grants to IPs, IPs are immediately acknowledged. However, when multiple grants are received by the same IP, these grants would arrive asynchronously and the first grant is selected. The unselected grants will reset the corresponding requests which is not observable to the initial request. Deferring the acknowledgement could solve this problem but, obviously, a significant control overhead is introduced.

As a solution to all these problems, a new asynchronous dispatching (AD) algorithm is proposed. In this algorithm, an IP requests only one LI to avoid orphans. CMs automatically update their states to IMs as feedback. Since IMs know the availability of CMs, IPs are always matched with the LIs connecting to idle CMs. The algorithm is divided into two independent sub-algorithms running in IMs and CMs. A simplified description is as follows:

- Sub-algorithm 1: Matching within IMs.
  - Step 1: A new packet arrives at IP(i, h).
  - Step 2: IM(i) selects an idle CM(r) according to feedback states from CMs.
  - Step 3: IM(i) sends a request to CM(r) through LI(i, r).
  - Step 4a: IM(i) successfully receives a grant from CM(r) and the packet is forwarded.
  - Step 4b: Otherwise, the state of CM(r) is updated. CM(r) is occupied by another IM; therefore, IM(i) goes back to step 1 to select a new CM.
- Sub-algorithm 2: Matching within CMs.
  - Step 1: A request is received.
  - Step 2: A grant is sent back if the target LO is available.
  - Step 3: The updated states are sent to all IMs.

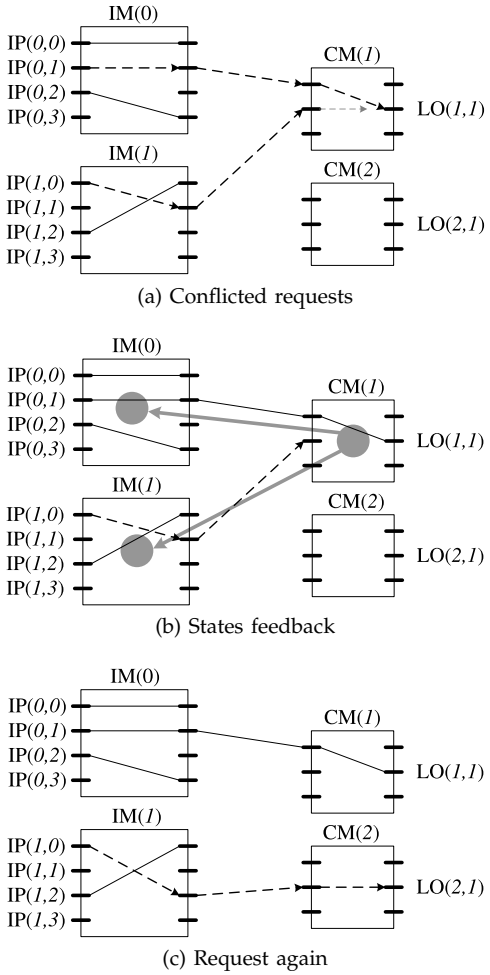
The AD algorithm is similar to the RD algorithm. IPs and LIs are randomly matched in IMs. CMs simply accept requests from IMs when the target LO is available. The AD algorithm also has iterations as the CRRD algorithm. When two requests from different IMs request the same LO, the first one is granted. Thanks to the state feedback, the IM lost arbitration will notice the state change and select another CM as described in step 4b of sub-algorithm 1.

Fig. 5 illustrates an example of iterations in AD. As shown in Fig. 5a, the $\mathbf{C}(4, 3, 4)$ network already has some links occupied in IM(0) and IM(1). Two concurrent packets arrived at IP$(0, 1)$ and IP$(1, 0)$ request OP$(1, 1)$ and OP$(1, 3)$ respectively. Suppose both IM schedulers in IM(0) and IM(1) have selected CM(1) for the incoming packets (both of them need LO$(1, 1)$) and the request from IM(0) is slightly faster than that from IM(1). According to sub-algorithm 2 of AD, CM(1) grants IM(0). Thus in Fig. 5b, the link between IM(0) and CM(1) is established and the updated states are sent back to all IMs including IM(1). After IM(1) has noticed its failure, it selects CM(2) to request again as depicted in Fig. 5c. The conflict between IP$(0, 1)$ and IP$(1, 0)$ is thus resolved.

4 Performance of RD, CRRD, and AD

In this section, the performance of RD, CRRD and AD algorithms is evaluated using behavioral level models written in MATLAB. A $\mathbf{C}(4, 8, 4)$ S^3 Clos network is built and injected with various traffic patterns. Some assumptions are taken to proceed a fair comparison:

- Random arbiters are utilized in all algorithms. Round-robin is a practical way to implement pseudo-random arbiters in synchronous circuits. Using random arbiters does not compromise the performance of CRRD.
- Only single cell packets are injected. Asynchronous Clos networks are reconfigured for packets instead of cells. Transferring multiple cells using the same path in synchronous Clos network is referred as bursty transmission [2] which normally compromises performance.
- Latency is estimated in units of cell time. Since packet length is one, the minimal reconfiguration time for a packet in asynchronous Clos networks

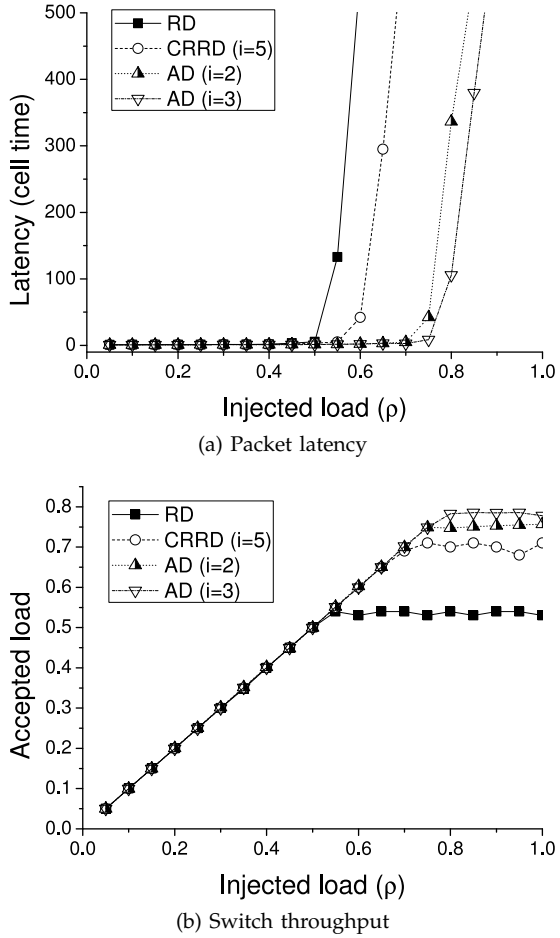


(a) Conflicted requests

(b) States feedback

(c) Request again

Fig. 5. Example of AD



(a) Packet latency

(b) Switch throughput

Fig. 6. Switch performance under non-blocking uniform traffic

is equivalent to cell time in synchronous Clos networks.

- Packets arrive synchronously. It is difficult to model a real event-driven simulation in MATLAB. The AD algorithm is still simulated in cycles but incoming packets are handled in a randomized sequence to simulate the event-driven behavior.

4.1 Non-blocking uniform traffic

Non-blocking uniform traffic spreads network load to all output ports without generating any head-of-line (HOL) blockage. Therefore, any blockage in this traffic is caused purely by routing algorithms. $\rho(s,d)$ is the normalized load between IP(s) and OP(d) where $0 \leq s,d < N$. A packet is injected in every cell time when $\rho = 1$ and no packet is injected when $\rho = 0$. The injected packet sequence of IP(s) complies with a Bernoulli process whose expectation $\bar{\rho}(s) = \sum_{d=0}^{N-1} \bar{\rho}(s,d)$. The individual load between any IP and OP is:

$$\bar{\rho}(s,d) = \frac{\bar{\rho}(s)}{N}$$

For each OP,

$$\sum_{s=0}^{N-1} \rho(s,d) \leq 1$$

is guaranteed to avoid overloaded OPs.

Fig. 6 shows packet latency and switch throughput performance using different routing algorithms. The RD algorithm has the maximal packet latency and the lowest switch throughput. The CRRD algorithm with 5 iterations increases throughput to 70% which is surpassed by the AD algorithm with 2 iterations. When it is fast enough to run 3 iterations, the AD algorithm reaches the maximal switch throughput around 76%.

The CRRD algorithm can reach 100% switch throughput in MSM networks [4] because IPs can send other cells when one is blocked. However, in an S^3 Clos network, an IP is blocked until the blocked cell is successfully forwarded.

Although CRRD resolves the uneven matching in IMs, it cannot avoid any conflicts in CMs, such as the problem shown in Fig. 5a. The AD algorithm has state feedback from CMs. Thus when any conflicts occur in CM schedulers, IMs can notice the conflicts and resolve them accordingly. As a result, the AD algorithm provides better latency and throughput performance than CRRD

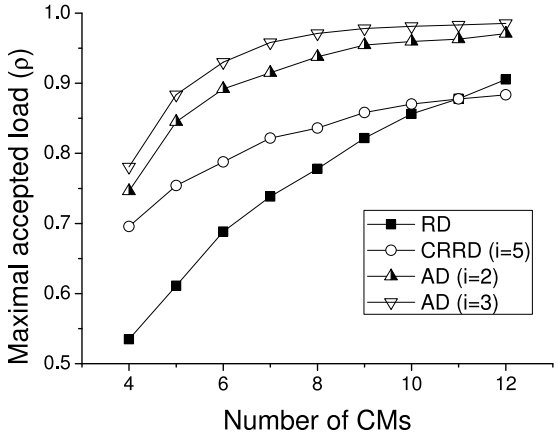


Fig. 7. Maximal accepted load with various number of CMs

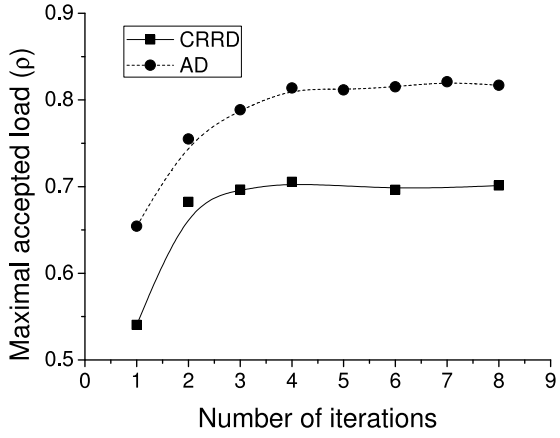


Fig. 8. Maximal accepted load with various number of iterations

and RD.

It is known that increasing the number of CMs improves throughput [3]. As shown in Fig. 7, all routing algorithms have better performance with more CMs. According to the theory of Clos networks, when the number of CMs is more than 7 the Clos network ($n = 4$) is an SNB network. However, none of these routing algorithms reach 100% throughput even with more than 7 CMs. The basic difference between heuristic algorithms and optimized algorithms is that heuristic algorithms provide sub-optimized routing results with less latency overhead. Since none of these routing algorithms can resolve all conflicts, the accepted load is always less than 100%.

It is shown in Fig. 7 that CRRD does not benefit as much as other algorithms. It achieves lower throughput than RD when there are 12 CMs because of conflicts in CMs. Once a conflict occurs, at least one IP needs to request again in the next cell time. Compared with it, RD allows an IP to be matched with multiple LIs. As long as one of these LIs is successfully granted, the requesting IP is successfully acknowledged. Therefore the multi-match issue of RD reduces conflict rate, although it is also the cause of the uneven matching in IMs. Iterations in AD resolve both the uneven matching in IMs and conflicts. It has the best performance.

Fig. 8 shows the maximal accepted load of CRRD and AD with various number of iterations. CRRD achieves peak throughput with more than four iterations. Since an IM in a $\mathbf{C}(4, 8, 4)$ network has four IPs, four iterations are enough to match all possible IPs. AD always provides higher throughput than CRRD because the AD algorithm can resolve conflicts in CMs. As shown in Fig. 8, the peak switch throughput of AD in a $\mathbf{C}(4, 8, 4)$ Clos network is around 82%.

4.2 Blocking traffic patterns

The traffic patterns generated by real applications are blocking. Uniform traffic is one of the most analyzed synthetic traffic pattern, which is defined as

$$\bar{\rho}(s, d) = \frac{\bar{\rho}(s)}{N}$$

Uniform traffic has the same load description as the non-blocking uniform traffic but without the extra non-blocking constraint; therefore, in some occasions, an OP could be loaded with traffic exceeding its actual bandwidth and some IPs are thus blocked.

Fig. 9 shows the accepted load under uniform traffic for all routing algorithms. S^3 Clos networks are input buffered switching networks. (Every IP is connected with an input buffer in simulation. This is different with the buffered IM scheme where an IP is connected to multiple virtual output queues inside the IM.) It is known that the maximal accepted load for an input buffered switching network is 58.6% [18]. When the number of CMs is 4 ($m = 4$), the Clos network is a RNB network where connection capability is restricted. As a result, the maximal accepted load of all routing algorithms is less than 58.6%. As shown in Fig. 9, the AD algorithm with 3 iterations has the best throughput performance of 50%, which is 1.3% higher than that of CRRD but 8.6% lower than the optimal throughput. We have simulated the performance of AD with 3 iterations in an SNB Clos network ($m = 7$). The maximal accepted load increases to 57.1%, which is only 1.5% lower than the optimal one.

Unbalanced traffic is used to evaluate the throughput performance under different level of conflicts. The traffic pattern is defined as

$$\bar{\rho}(s, d) = \begin{cases} \bar{\rho}(s) \cdot (w + \dfrac{1-w}{N}), & \text{if } s = d, \\ \bar{\rho}(s) \cdot \dfrac{1-w}{N}, & \text{otherwise.} \end{cases}$$

where w is the unbalanced factor. When $w = 0$, the injected traffic is uniform. On the other hand, when $w = 1$, the traffic is totally unbalanced and all traffic loaded on IP(s) is heading for OP(d), where $s = d$. In this

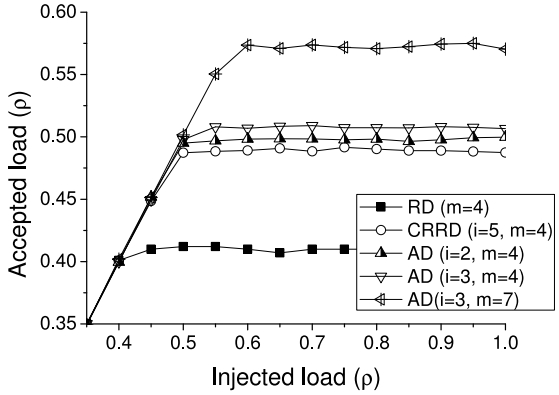


Fig. 9. Accepted load under uniform traffic

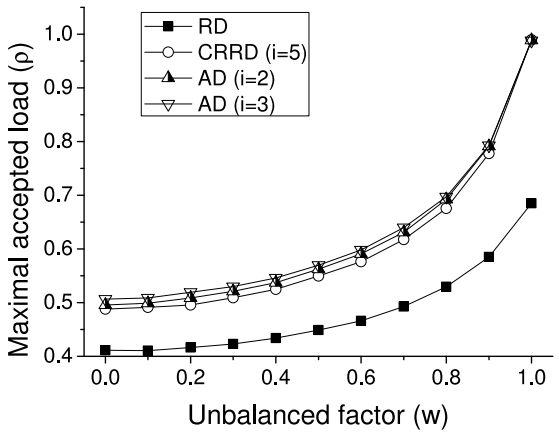


Fig. 10. Accepted load under unbalanced traffic

case, the conflicts in CMs are eliminated by the traffic pattern itself.

Fig. 10 shows the maximal accepted load of all routing algorithms. Due to the uneven matching in IMs, RD has the lowest throughput in all algorithms. Both CRRD and AD algorithms can resolve the uneven matching in IMs. Conflicts in CMs are gradually alleviated along with the increasing unbalanced factor. Both CRRD and AD demonstrate 100% throughput performance when $w = 1$.

5 Implementation

In this section, the AD algorithm is implemented into a hardware scheduler to control a 32-port $\mathbf{C}(4, 8, 4)$ S^3 Clos network.

Fig. 11 depicts the internal architecture of the Clos scheduler. As described in the beginning of section 3, the routing procedure is divided into two stages: module matching and port matching. The AD algorithm is used in module matching (IM_Dis and CM_Dis) to reconfigure IMs and CMs. Once module matching is finished, the request information for port matching (*OMr*) is forwarded to OMs using the configured RIMs and RCMs. After an OM scheduler (OM_Sch) has successfully allocated an OP for the incoming request, the grant signal travels backward through the Clos network and finally an IP is acknowledged.

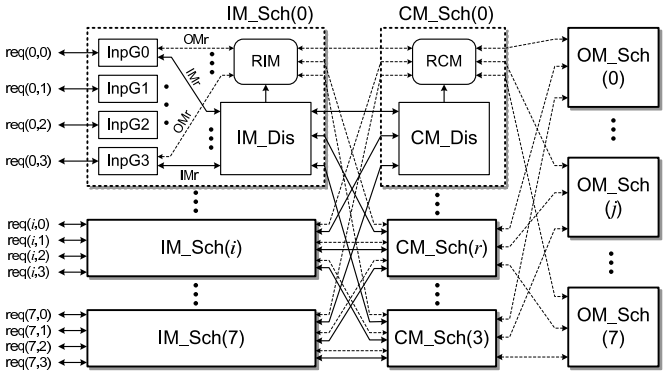


Fig. 11. Architecture of the Clos Scheduler

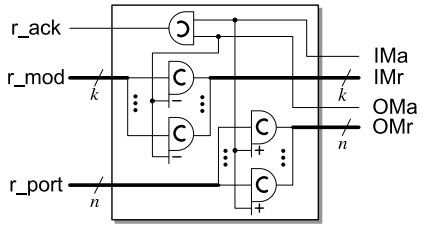


Fig. 12. Input generator

5.1 Input generator (InpG)

An IP can requests for up to 32 OPs in a $\mathbf{C}(4, 8, 4)$ network. In module matching, an IM containing a requesting IP is required to find out the OM containing the target OP from a total of k OMs and reserve a path to it. Then the target OP, one of the n OPs in the OM, is allocated in port matching. Therefore, the request from an IP is divided into two sub-requests: *r_mod* that denotes the target OM, and *r_port* that denotes the target OP in that OM. In this paper, we suppose the incoming request is already divided into sub-requests and one-hot coded.

An input generator module is added on each IP to feed requests to IMs and OMs separatively, and generate the final acknowledgement to the IP. The internal structure of an InpG is shown in Fig. 12. Since a path from IMs to OMs is reserved before port matching, the request (*IMr*) used in module matching is generated immediately after the incoming request and held until the reserved OM is released, which is indicated by the drop of *OMa*. The requests to OMs (*OMr*) are generated after a path to the target OM is reserved, indicated by *IMa* +, and released before *IMr*.

5.2 IM dispatcher (IM_Dis)

IM dispatchers are the most important and complicated modules of AD. They receive requests (*IMr*) from IPs, match IPs to LIs according to the state feedback of CMs (*CMs*), and configure IM switches.

Fig. 13 demonstrates an IM dispatcher for a $\mathbf{C}(4, 8, 4)$ Clos network. It recieves four requests, $IMr(0 \sim 3)$, from its four IPs and each bit of *IMr* denotes a target OM

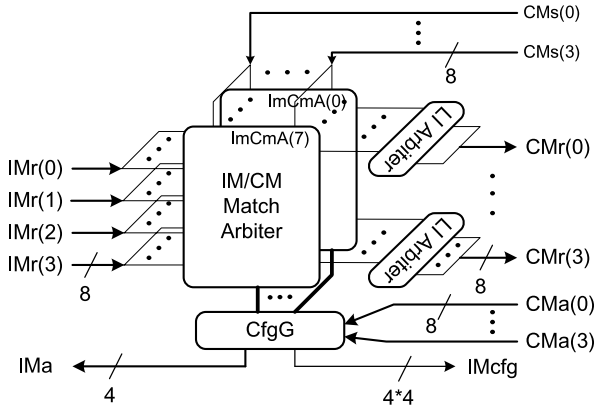
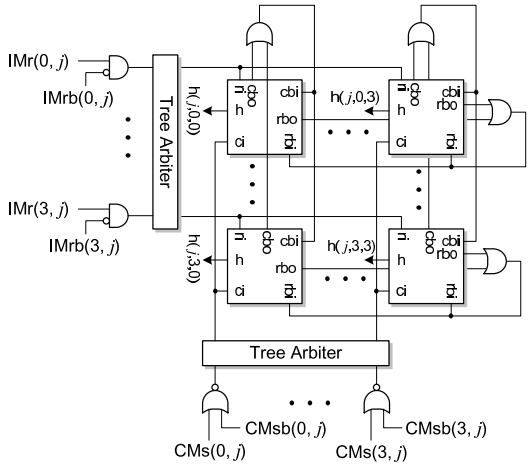
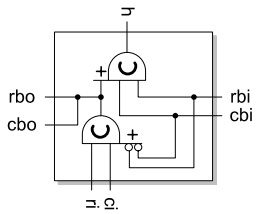


Fig. 13. IM dispatcher



(a) Tile matrix



(b) A tile

Fig. 14. IM/CM arbiter

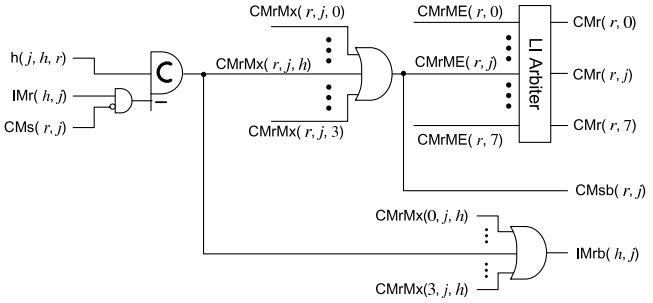


Fig. 15. Connection between IM/CM match arbiters and LI arbiters

in the eight OMs. Accordingly, the four states feedback signals, $CMs(0 \sim 3)$, from the four CMs denotes the availability for all OMs in each CM. One IM/CM match arbiter is a 4×4 multi-resource arbiter [15]–[17] which can match up to four IPs requesting the same OM to four LIs sequentially. Since there are eight OMs in $\mathbf{C}(4, 8, 4)$, an IM dispatcher has eight IM/CM match arbiters to handle requests to the eight OMs concurrently. However, a LI could be selected by more than one IM/CM match arbiters simultaneously. A LI arbiter is thus added to select one active request of them. The internal states of all IM/CM match arbiters are forwarded to the configuration generator (CfgG), which uses the acknowledge signals from CMs (*CMa*) to generate the IM configuration (*IMcfg*) and the acknowledgement to InpGs (*IMa*).

Fig. 14 shows the internal structure of the IM/CM match arbiter for OM(j). $IMr(0, j)$∼$IMr(3, j)$ are active high requests targeting OM(j). $CMs(0, j)$∼$CMs(7, j)$ are state feedback signals from CMs. When $CMs(r, j)$ is low, the LO(r, j) connecting CM(r) to OM(j) is available. Multiple requests could arrive simultaneously and multiple CMs could be available at the same time as well. Therefore, two tree arbiters [19], [20] are added to ensure only one IP and CM pair is matched at one time. The tile-based matrix are QDI circuits ensuring the corresponding match result $h(j, h, r)$ are safely and explicitly notified. Since only one pair of IP and LI is matched at one time, the match result is first captured by the following circuits ($CMrMx(r, j, h)$ in Fig. 15) and the request $IMr(h, j)$ and the state signal $CMs(r, j)$ are consequently blocked by $IMrb(h, j)$ and $CMsb(r, j)$ to allow other pairs to be matched. The detailed timing analysis and circuit implementation of the tile-based multi-resource arbiter is described in [16], [17].

Fig. 15 shows the connection between IM/CM match arbiters and LI arbiters. The match result $h(j, h, r)$ is captured by $CMrMx(r, j, h)$, a three-dimension matrix. After the C2N elements in Fig. 15 has captured the value, corresponding block signals $IMrb(h, j)$ and $CMsb(r, j)$ are fired to withdraw $h(j, h, r)$. $CMrMx(r, j, h)$ would be released under two situations: the request from IP(h) has been successfully served (denoted by $IMr(h, j)$–), or the target OM(j) has been allocated to other IMs (indicated by $CMs(r, j)$+). As mentioned earlier, multiple IM/CM arbiters could select the same LI. The LI arbiter on $CMrME(r)$ selects one active request and $CMr(r)$ is therefore one-hot coded. For those requests blocked by LI arbiters, they will be reset when feedback signals *CMs* are driven by the chosen requests.

As shown in Fig. 16, the configuration generator utilizes the captured match results (*CMrMx*), the requests sent to CMs (*CMr*) and the grant signals from CMs (*CMa*) to generate the configuration signals *IMcfg* and acknowledgement *IMa*. $CMr(r, j)$ specifies that the active match $CMrMx(r, j, h)$ is selected by the LI arbiter. When a positive grant on $CMa(r, j)$ is detected, the corresponding bit in the configuration matrix *cfgMx* is notified. The switch configuration signal $IMcfg(r, h)$ is synthesized from *cfgMx* using an OR gate. Using the same structure, the acknowledgement signal $IMa(h)$ is generated from

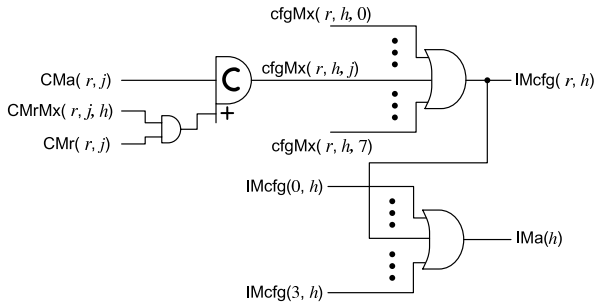


Fig. 16. Configuration generator

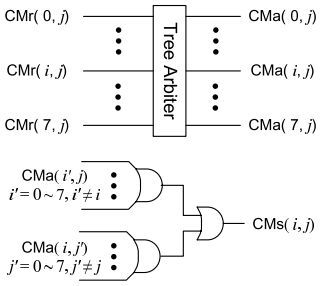


Fig. 17. CM dispatcher

IMcfg.

5.3 CM dispatcher (CM_Dis)

CM dispatchers reconfigure the central stage of a Clos network using the AD algorithm. They are similar to the arbiters of crossbars where each output port has an independent arbiter to grant requests from all input ports.

Fig. 17 shows a part of the CM dispatcher in CM(r). Since a CM in $\mathbf{C}(4, 8, 4)$ is a 8×8 switch, there is a 8-input arbiter on each output port, such as the one for LO(r, j) shown in Fig. 17. *CMa*(i, j) and *CMs*(i, j) is a pair of complementary acknowledgement for *CMr*(i, j) where *CMa*(i, j) denotes a successful grant and *CMs*(i, j) indicates the resource is occupied by other requests. *CMs*(i, j) is generated by an OR gate tree. When *CMs*(i, j) is notified by the upper half of the tree, the input link LI(i, r) is occupied by another request from the same IM; when *CMs*(i, j) is notified by the lower half of the tree, the output link LO(r, j) is occupied by other requests from other IMs. In either case *CMr*(r, j) should be withdrawn.

5.4 Other parts of the Clos scheduler

RIMs and RCMs are crossbars controlled by IM_Dises and CM_Dises respectively. They transmit *OMr* from InpGs to OM_Sches once the first two stages are reconfigured by the AD algorithm. OM_Sches are normal crossbar arbiters which have the same structure as the CM_Dis as shown in Fig. 17 but without the OR gate trees. The grant signal from OM_Sches are sent back to InpGs using the same path reserved in RIMs and RCMs.

TABLE 1
Area Breakdown

	area (um^2)	gate count (NAND2X1)	percent
InpG	6.26K	1.6K	2.5%
IM_Dis	175.06K	43.8K	67.2%
CM_Dis	33.75K	8.4K	12.9%
OM_Sch	10.06K	2.5K	3.8%
RIM & RCM	13.46K	3.4K	5.2%
other	22.15K	5.5K	8.4%
Total	260.74K	65.2K	

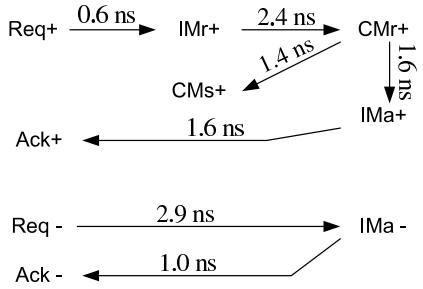


Fig. 18. Speed performance

6 Hardware Performance

The Clos scheduler is implemented using the Faraday 0.13 μm standard cell based on the UMC 0.13 μm technology. All modules including the basic C elements and MUTEX gates are manually written by Verilog HDL using standard cells only. The design is synthesized by Design Compiler and post-synthesis simulations are back-annotated with cell latency.

Table 1 shows the area consumption of the Clos scheduler. IM dispatchers are the most area consuming modules that use 67.2% of the total area. Since an IM_Dis contains k IM/CM match arbiters ($n \times m$) and two $k \times m \times n$ signal matrices (*CMrMx* and *cfgMx*), the area of a single IM_Dis is proportional to $k \times m \times n$ and, therefore, the area of all IM_Dis modules is proportional to k^2mn. When the Clos network is a RNB network, $m = n$ and $k^2mn = N^2$. The area of IM_Dis modules is proportional to N^2 in a RNB Clos network.

The speed performance of the Clos scheduler is demonstrated in Fig. 18. When no conflicts occur in the Clos network, a path is allocated in 6.2 ns and released in 3.9 ns. The AD algorithm requires 4.0 ns to reconfigure the first two stages. We also evaluate the latency of the iterations between IMs and CMs. When a request is sent to a CM, the state feedback *CMs* is sent to other IMs in 1.4 ns. Therefore, the latency of an iteration from *IM+* to *CMs+* is 3.8 ns. When the request from an IP is withdrawn immediately after the path is reserved, the request procedure reaches the minimal period of 10.1 ns, which contains 2.66 iterations. As a result, the iteration limit of the AD algorithm in this implementation is 2.66.

It is possible to evaluate the consistency between the hardware implementation and the behavior models used in section 4. The post-synthesis netlist is injected with

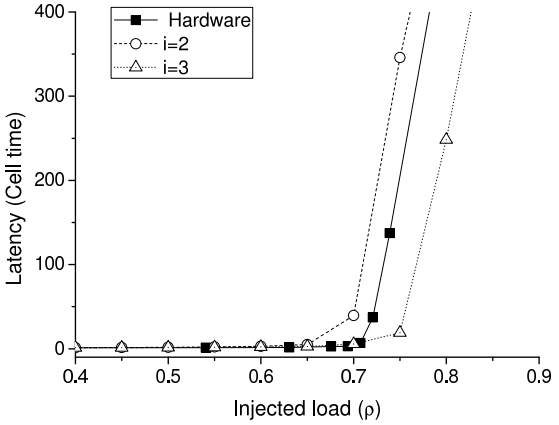


Fig. 19. Consistency between hardware and behavior simulation

the non-blocking uniform traffic pattern described in section 4.1. The injected load is normalized by converting the minimal period of 10.1 ns to a cell time. As shown in Fig. 19, the latency performance of the hardware implementation is in the between of the two behavior models. Therefore, the behavior level estimations in section 4 is consistent with the practical hardware performance.

7 Conclusion

In this paper, the first asynchronous routing algorithm for three-stage Clos networks is proposed and implemented.

The first asynchronous dispatching algorithm for the first two stages of a Clos network, namely the asynchronous dispatching (AD) algorithm, is proposed and evaluated. We have analyzed two classic dispatching algorithms used in synchronous Clos networks: random dispatching (RD) and concurrent round-robin dispatching (CRRD). Both of them are heuristic algorithms that provide fast configurations. However, they cannot be directly utilized to control an asynchronous Clos network. The proposed AD algorithm is capable of routing a Clos network asynchronously by using state feedback from CMs and dividing the sequential algorithm, such as RD and CRRD, into two independent sub-algorithms running in IMs and CMs. All algorithms have been evaluated on behavior level. The results show: the AD algorithm outperforms both RD and CRRD under all traffic patterns; without buffering in IMs, none of the algorithms reaches 100% load but the AD algorithm shows only 1.5% load loss compared with the optimal 58.6% load performance under uniform traffic.

The new routing algorithm has been implemented into a Clos scheduler to control a 32-port $\mathbf{C}(4, 8, 4)$ S^3 Clos network using the Faraday 0.13 μm cell library. Only standard cells have been used with no custom asynchronous cells. The Clos scheduler consumes 65.2K gates. Post-synthesis simulations with back-annotated cell latency show that the Clos scheduler can reserve a path in 6.2 ns and release it in 3.9 ns. The loop latency of an iteration is 3.8 ns and the minimal configuration period of 10.1 ns contains 2.66 iterations.

References

[1] C. Clos, "A study of nonblocking switching networks," *Bell System Technical Journal*, vol. 32, no. 5, pp. 406–424, March 1953.

[2] H. J. Chao, C. H. Lam, and E. Oki, *Broadband Packet Switching Technologies: A Practical Guide to ATM Switches and IP Routers*. John Wiley & Sons, Inc., 2001.

[3] F. M. Chiussi, J. G. Kneuer, and V. P. Kumar, "Low-cost scalable switching solutions for broadband networking: the ATLANTA architecture and chipset," *IEEE Communications Magazine*, vol. 35, no. 12, pp. 44–53, December 1997.

[4] E. Oki, Z. Jing, R. Rojas-Cessa, and H. J. Chao, "Concurrent round-robin-based dispatching schemes for Clos-network switches," *IEEE/ACM Transactions on Networking*, vol. 10, no. 6, pp. 830–844, 2002.

[5] H. J. Chao, Z. Jing, and S. Y. Liew, "Matching algorithms for three-stage bufferless Clos network switches," *IEEE Communications Magazine*, vol. 41, no. 10, pp. 46–54, October 2003.

[6] H. J. Chao, K.-L. Deng, and Z. Jing, "Petastar: a petabit photonic packet switch," *IEEE journal on Selected Areas in Communications*, vol. 21, no. 7, pp. 1096–1112, September 2003.

[7] J. Cheyns, C. Develder, and et al, "Clos lives on in optical packet switching," *IEEE Communications Magazine*, vol. 42, no. 2, pp. 114–121, 2004.

[8] R. Rojas-Cessa and C.-B. Lin, "Scalable two-stage Clos-network switch and module-first matching," in *Proc. of Workshop on High Performance Switching and Routing*, 2006, pp. 303–308.

[9] E. Oki, N. Kitsuwan, and R. Rojas-Cessa, "Analysis of space-space-space clos-network packet switch," in *Proc. of Internatonal Conference on Computer Communications and Networks*, August 2009, pp. 1–6.

[10] H. J. Chao, K.-L. Deng, and Z. Jing, "Petastar: a petabit photonic packet switch," *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 7, pp. 1096–1112, September 2003.

[11] J. Kim, W. J. Dally, B. Towles, and A. K. Gupta, "Microarchitecture of a high radix router," in *Proc. of ISCA*, June 2005, pp. 420–431.

[12] C. Gómez, M. E. Gómez, P. López, and J. Duato, "Exploiting wiring resources on interconnection network: increasing path diversity," in *Proc. of Euromicro Conference on Parallel, Distributed and Network-Based Processing*, February 2008, pp. 20–29.

[13] S. Scott, D. Abts, J. Kim, and W. J. Dally, "The blackwidow high-radix clos network," *SIGARCH Comput. Archit. News*, vol. 34, no. 2, pp. 16–28, 2006.

[14] A. Leroy, D. Milojevic, D. Verkest, F. Robert, and F. Catthoor, "Concepts and implementation of spatial division multiplexing for guaranteed throughput in networks-on-chip," *IEEE Transactions on Computers*, vol. 57, no. 9, pp. 1182–1195, September 2008.

[15] S. Golubcovs, D. Shang, F. Xia, A. Mokhov, and A. Yakovlev, "Modular approach to multi-resource arbiter design," in *Proc. of ASYNC*, 2009, pp. 107–116.

[16] S. Golubcovs, D. Shang, and et al, "Multi-resource arbiter decomposition," Newcastle University, Tech. Rep., 2009.

[17] D. Shang, F. Xia, S. Golubcovs, and A. Yakovlev, "The magic rule of tiles: virtual delay insensitivity," in *Proc. of PATMOS*, 2009.

[18] M. J. Karol, M. G. Hluchyj, and S. P. Morgan, "Input versus output queueing on a space-division packet switch," *IEEE Transactions on Communications*, vol. 35, no. 12, pp. 1347–1356, December 1987.

[19] M. B. Josephs and J. T. Yantchev, "CMOS design of the tree arbiter element," *IEEE Transactions on VLSI*, vol. 4, December 1996.

[20] D. J. Kinniment, *Synchronization and Arbitration in Digital Systems*. John Wiley & Sons Inc., 2007.