www.ietdl.org

Published in IET Computers & Digital Techniques
Received on 15th October 2010
Revised on 8th June 2011
doi: 10.1049/iet-cdt.2010.0150

Selected papers from the 10th International Conference on Application of Concurrency to System Design (ACSD 2010)

ISSN 1751-8601

Routing of asynchronous Clos networks

W. Song[1] *D. Edwards*[1] *Z. Liu*[1,2] *S. Dasgupta*[1]

[1]*School of Computer Science, University of Manchester, Manchester M13 9PL, UK*
[2]*School of Information and Electronics, Beijing Institute of Technology, Beijing 100081, People's Republic of China*
E-mail: songw@cs.man.ac.uk

Abstract: Clos networks provide theoretically optimal solution to build high-radix switches. Dynamically reconfiguring a three-stage Clos network is more difficult in asynchronous circuits than in synchronous circuits. This study proposes a novel asynchronous dispatching (AD) algorithm for general three-stage Clos networks. It is compared with the classic synchronous concurrent round-robin dispatching (CRRD) algorithm in unbuffered Clos networks. The AD algorithm avoids the contention in central modules using a state feedback scheme and outperforms the throughput of CRRD in behavioural simulations. Two asynchronous Clos networks using the AD algorithm are implemented and compared with a synchronous Clos network using the CRRD algorithm. The asynchronous Clos scheduler is smaller than its synchronous counterpart. Synchronous Clos networks achieve higher throughput than asynchronous Clos networks because asynchronous Clos networks cannot hide the arbitration latency and their data paths are slow. The asynchronous Clos scheduler consumes significantly lower power than the synchronous scheduler and the asynchronous Clos network using bundled-data data switches shows the best power efficiency in all implementations.

1 Introduction

Clos networks are a class of multi-stage switching networks first proposed over 50 years ago [1]. They provide theoretically optimal solution to build high-radix switches when the requirement exceeds the capacity of a feasible crossbar. Although emerging very large-scale integration technologies intensively reduce the area of a single cross-point and enlarge the capacity of a crossbar, the insatiable desire for speed and performance always pushes router designs to the very limit. Clos networks are still used in the state-of-the-art designs.

Early telephone networks are circuit-switched networks where switches are statically configured. The later asynchronous transfer mode (ATM) networks and Internet protocol (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 using heuristic algorithms. Subsequently, numerous routing algorithms have been proposed to improve throughput [4–9]. A Clos network designed for current optical backbone networks has already achieved peta-bit throughput [6].

Clos networks also find their utilisation in intra- and inter-chip interconnection networks. Transistor scaling increases the available bandwidth of a router chip and the wire resources in on-chip networks. A router with many narrow ports is more efficient than a router with a few wide ports [10, 11]. A folded-Clos network is used in the Cray BlackWidow multiprocessor to support high-bandwidth communications [12] and Beneš networks (multi-stage Clos networks) are used in routers of an on-chip network providing delay-guaranteed services [13].

Asynchronous circuits [14] are well known for their low-power consumption and tolerance to process, voltage and temperature variation [15]. 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 implementations. No asynchronous implementation has yet been reported.

In this paper, a novel asynchronous dispatching (AD) algorithm is proposed to reconfigure unbuffered asynchronous Clos networks. It can be directly utilised to substitute the high-radix switch in asynchronous spatial division multiplexing routers [16] or asynchronous high-radix routers in on-chip networks where area constraints are important. Compared with the classic concurrent round-robin dispatching (CRRD) algorithm in unbuffered Clos networks, AD provides higher throughput in behavioural-level simulations. An asynchronous Clos network using bundled-data data switches is area and power-efficient. The remainder of this paper is organised as follows: Section 2 explains the background knowledge needed to understand this work. Section 3 describes the CRRD and the AD algorithms in detail. Section 4 then compares the performance of these two algorithms in behavioural simulations. Section 5 demonstrates the way of implementing an asynchronous Clos scheduler using the AD algorithm. Later in Section 6, three different Clos networks are implemented and compared. Finally, the paper is concluded in Section 7.

2 Clos network

Fig. 1 shows a three-stage Clos network. The terminologies used in this paper are 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 ($0 < i \leq k$).
j	index of OMs ($0 < j \leq k$).
r	index of CMs ($0 < r \leq m$).
h	index of IPs/OPs in an IM/OM ($0 < h \leq n$).
IM(i)	the (i)th IM.
OM(j)	the (j)th OM.
CM(r)	the (r)th CM.
IP(i, h)	the (h)th IP in IM(i).
OP(j, h)	the (h)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).
$\boldsymbol{C}(n, k, m)$	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 $n \times m$ crossbar. In the second stage, m CMs are statically connected with IMs and each CM is a $k \times k$ crossbar. The third stage contains k OMs, each of which is an $m \times n$ crossbar statically connected with CMs.

Switching networks can be classified into three categories [2]:

(i) *Blocking*: the switches have possible connection states such that an available input/output (I/O) pair cannot be connected because of internal blocking.
(ii) *Strict non-blocking (SNB)*: the switches ensure the connection of any available I/O pairs without altering any established connections.

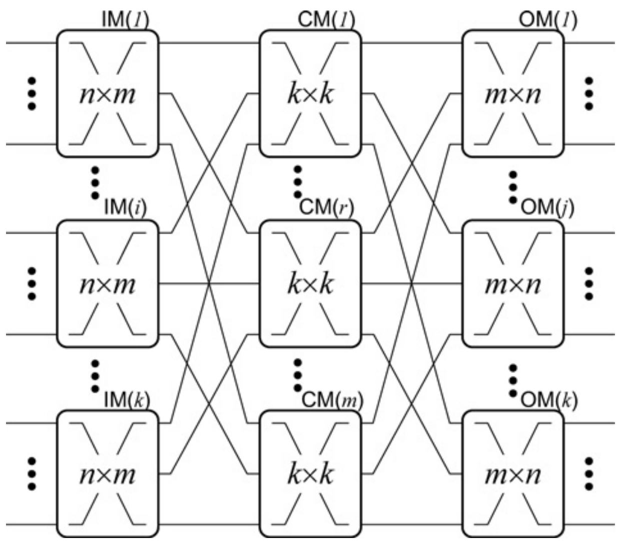


Fig. 1 *Three-stage Clos network* $\boldsymbol{C}(n, k, m)$

(iii) *Rearrangeable non-blocking (RNB)*: the switches ensure the connection of any available I/O pairs with possible modification of established connections. A three-stage Clos network with n CMs ($m = n$) is a RNB network, while it is an SNB network with more than $2n - 1$ CMs [2].

The major advantage of Clos networks over crossbars is their area efficiency. The area of a switching network is proportional to the number of internal cross-points. For a crossbar with N I/O ports, the area is proportional to the cost C.

$$C_{\text{CB}} = N^2 \tag{1}$$

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

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

$$C_{\text{Clos,RNB}} \geq (2N)^{1.5} \tag{3}$$

Fig. 2 demonstrates the area of crossbars and Clos networks with various numbers of ports. Both SNB and RNB Clos networks reduce area overhead significantly and RNB Clos networks have the minimal area. There are two classes of routing algorithms for Clos networks [5]: optimal algorithms, which provide guaranteed results for all matches but with a high complexity in time or implementation, and heuristic algorithms, which provide all or partial matches in low time complexity. Although optimal algorithms guarantee the connection of any I/O pairs, they require a global view of all modules and consume long time to reconfigure. On the other hand, heuristic algorithms are fast and spatially distributed. Most of current dynamically reconfigurable Clos networks utilise heuristic algorithms [3–9].

Buffer insertion is a usual way of improving throughput. According to the stage where buffers are inserted, a Clos network can be a space–space–space (S^3) network without any buffers, 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 (or unbuffered networks) introduce no buffer overhead but provide the worst throughput. SMS networks normally show better throughput than MSM networks because the buffers in CMs resolve the contention in CMs; however, this scheme requires a re-sequencing function in OMs because data issued to OMs are out-of-order. MSM is the most utilised scheme in ATM networks. Buffers in IMs

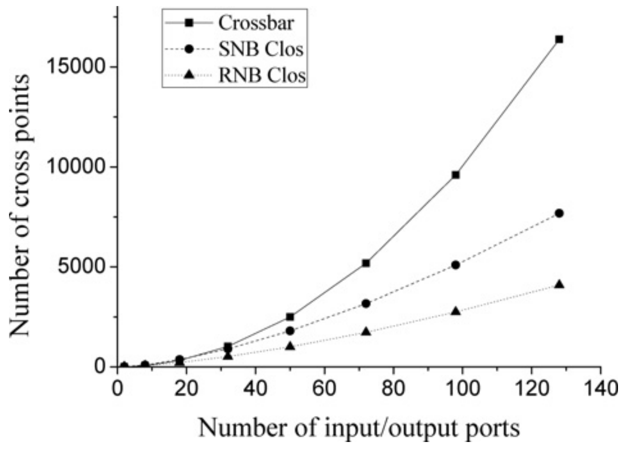


Fig. 2 *Area of crossbar and Clos network*

and OMs improve throughput without the out-of-order problem but the OMs are required to speed up m times to avoid throughput degradation. (A detailed comparison of buffer insertion schemes and memory speed-up can be found in [2, 9].)

Virtual output queuing (VOQ) [17] is an important concept in Clos networks. It is a buffer technique that solves the head-of-line (HOL) blocking problem. Instead of using a first-in-first-out (FIFO) queue for each IP which limits the throughput to 58.6% [18], breaking the queue into N logical VOQs and storing data heading to different OPs in individual VOQs achieve 100% throughput [17]. This buffer technique can be used in input-buffered switches [9, 17] or inside the IMs of MSM Clos networks [4].

In this paper, we consider only three-stage S^3 Clos networks with no VOQs in input buffers or in IMs. This limitation is introduced for two reasons:

1. The area consumption of VOQs or buffered switches is over-large for the routers in on-chip network, which is the direct application of this paper. Similar with the VOQs in input-buffered switches, routers in on-chip networks can use virtual channels (VCs) [19] to alleviate the HOL blocking with much less area overhead than VOQs. However, the analyses in [16] show that the area overhead of VCs is already significantly large in asynchronous VC routers and the synchronisation introduced by VCs compromises its throughput improvement.
2. The routing algorithms for S^3 Clos networks can be easily extended to support SMS or MSM Clos networks. The real difficulties are in the asynchronous implementations, which can schedule a Clos network complying with these algorithms. As it will be shown in Section 5, the scheduler for an S^3 Clos network is already complicated. It is better to keep the problem simple at this early research stage.

3 Dispatching algorithms

In a Clos network, CMs are shared by all I/O pairs as every I/O pair has m possible path configurations and each of them utilises a different CM. In the worst case, all the nk IPs would try to utilise the same CM ignoring that one CM is capable of setting up only k paths. Therefore an efficient routing algorithm must dispatch requests from IPs to all CMs evenly; otherwise, the throughput performance is compromised. Heuristic algorithms process a request from IP(i_1, h_1) to OP(j_2, h_2) in two stages [8]. First, module matching: reserving a path from IP(i_1, h_1) to an LO(r, j_2) which is connected with OM(j_2), and secondly port matching: connecting LO(r, j_2) and OP(j_2, h_2) in OM(j_2). As the module matching stage chooses the target CMs for all IPs, it determines the request distribution which directly affects the throughput of a routing algorithm. The sub-algorithm used in module matching, namely the dispatching algorithm, is the key research issue of Clos routing algorithms.

3.1 Concurrent round-robin dispatching (CRRD)

The data transmitted in synchronous Clos networks are routed in units of a 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 data transmission in a pipelined manner. The new configuration generated in the current cell time takes effect in the next cell time. The latency of generating a new configuration for the Clos network is therefore hidden. A cell time lasts one or multiple cycles depending on the complexity of the routing algorithm.

The CRRD algorithm [4] is one of the classic algorithms extensively utilised in synchronous Clos networks. It was first proposed in MSM Clos networks where VOQs are implemented in IMs. CRRD provides 100% throughput. The algorithm can be modified to schedule S^3 Clos networks, which is used in this paper to represent the classic performance of synchronous S^3 Clos networks.

As indicated by its name, the original CRRD algorithm places independent round-robin arbiters on each LI (output-link arbiter), VOQ (VOQ arbiter) and LO. In an S^3 Clos network, the VOQ arbiters are replaced with input-port arbiters on each IP. The modified description of CRRD is illustrated as follows:

1. Phase 1: Matching within IMs.

The first iteration

- *Step 1:* Non-idle IPs send requests to all output-link arbiters.
- *Step 2:* Each output-link arbiter selects an IP.
- *Step 3:* Each non-idle IP accepts one LI from the received grants.

The ith iteration ($i > 1$)

- *Step 1:* Unmatched IPs send requests to all output-link arbiters.
- *Steps 2 and 3:* Same as the first iteration.

2. Phase 2: Matching within CMs.

- *Step 1:* Matched LIs send requests to CMs. Each LO in CMs selects one request and returns a grant.
- *Step 2:* In the next cell time, the granted IPs send their cells and other IPs try again.

Fig. 3 shows an example of the iterations in CRRD. Assume that all IPs have received new packets and all LIs are available initially. As shown in Fig. 3*a*, IPs send requests to all available LIs. Then each LI grants one IP. Fig. 3*b* illustrates an uneven request distribution where multiple LI arbiters select the same IP. The step 3 of CRRD ensures all IPs to accept only one LI (Fig. 3*c*). In this way, the unmatched IPs are able to try again in the next iteration as shown in Fig. 3*d*.

CRRD ensures that requests are dispatched to different CMs evenly and one IP requests only one CM. However, the even distribution relies on the number of iterations. In

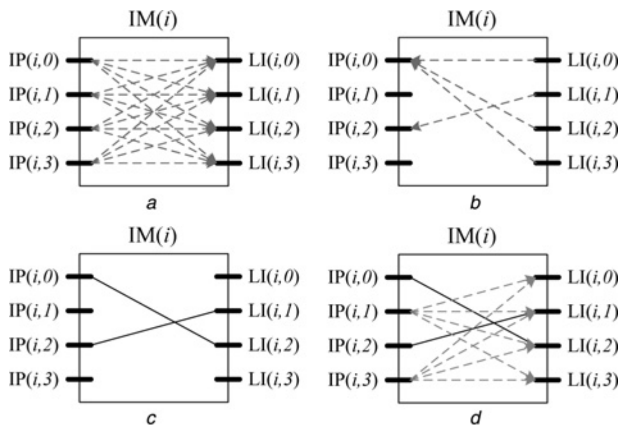


Fig. 3 *Example of iterations in CRRD*

a Step 1, request
b Step 2, grant
c Step 3, accept
d Step 1, next iteration

the worst case when only one match is made in each iteration, an IM needs n iterations to match all the n IPs. The number of iterations is limited by the cell time. As one iteration needs one clock cycle to finish, the cell time must be longer than n cycles to guarantee even request distribution.

Although requests from IPs are evenly distributed to all CMs, two requests from different IMs asking for the same OM can be distributed to the same CM competing for the same LO. CRRD produces even request distribution but this distribution is oblivious to the possible contention between requests from different IMs.

3.2 Asynchronous dispatching (AD)

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

- Incoming packets arrive asynchronously.
- An asynchronous Clos network is reconfigured for packets instead of cells.
- Modules are event-driven. The dispatching of different requests are not synchronised.

As a solution to these problems, a new AD algorithm is proposed. In this algorithm, the matching within IMs and the matching within CMs are separated in two independent sub-algorithms running concurrently. All the configuration modules are event-driven. Independent arbiters are placed on each LI (output-link arbiter), IP (input-port arbiter) and LO as the modified CRRD algorithm does, but these arbiters are MUTEX arbiters [20] or tree-arbiters [21, 22].

In the CRRD algorithm, if a request fails to reserve a path due to the contention in CMs, it automatically tries again in the next cell time. However, an asynchronous request cannot withdraw itself until an acknowledgment is received. The path in an asynchronous Clos network is reserved for a whole packet instead of a single cell. Directly adopting the CRRD algorithm in asynchronous Clos networks introduces severe arbitration latency because the contention in one CM causes at least one request to wait a whole packet time even when other CMs are available. To reduce such latency overhead, the AD algorithm introduces a state feedback scheme. Once an LO is occupied or released, the information is broadcasted to all IMs. Since IMs are informed of the availabilities of LOs in all CMs, they distribute requests only to the CMs with available LOs in the IM matching sub-algorithm. The contention in CMs is accordingly avoided. A simplified description is as follows:

1. Sub-algorithm 1: Matching within IMs.
- *Step 1:* A new packet arrives at IP(i, h).
- *Step 2:* IP(i, h) waits until at least one target LO is available.
- *Step 3:* IP(i, h) sends requests to all output-link arbiters leading to the available LOs.
- *Step 4:* Output-link arbiters return grants to IP(i, h).
- *Step 5:* IP(i, h) selects a path and withdraws requests to other output-link arbiters.

2. Sub-algorithm 2: Matching within CMs.
- *Step 1:* A request is forwarded from an IM.
- *Step 2:* The target LO returns a grant to the IM and reconfigures the CM once it is available.
- *Step 3:* The updated states are broadcasted to all IMs.

Although the algorithm description appears in a way that only one request is served at one time, the same algorithm runs concurrently in all IPs. Thus a maximal of N requests from all IPs can be served simultaneously but these requests are unsynchronised: a request may arrive at any time when another request is under processing.

Fig. 4 illustrates an example of the state feedback scheme in the AD algorithm. The initial state is shown in Fig. 4*a*. The C(3, 3, 3) network has some links occupied already, such as the links on paths IP(2, 3) to LO(1, 3) and LI(3, 3) to LO(3, 1). Assume that a new packet arrives at IP(2, 1) and requests an OP in OM(1). According to this request, the packet must occupy a CM with an available LO to OM(1). Informed by the state feedback from all CMs, the sub-algorithm running in IM(2) learns that LO(3, 1) in CM(3) is already taken. IP(2, 1) sends requests to LI(2, 1) and LI(2, 2) which lead to the CMs with the available LOs to OM(1). Since LI(2, 1) is already occupied by the path IP(2, 3) to LO(1, 3), as depicted in Fig. 4*b*, only the output-link arbiter on LI(2, 2) returns a grant to IP(2, 1). Later in Fig. 4*c*, IP(2, 1) sends a request to LO(2, 1) through LI(2, 2). The arbiter on LO(2, 1) accepts this request and broadcasts its new state to all IMs. Similar to the CRRD algorithm, the sub-algorithm running in IMs evenly distributes requests to CMs but this distribution is no longer oblivious to the contention in CMs. IMs utilise the state feedback from CMs to avoid contention, which also increases throughput.

The state feedback scheme cannot resolve the contention among the requests processed simultaneously because they use the same state feedback. In other words, the state feedback avoids contention between established paths and future requests but cannot resolve the existing contention. If contention occurs, multiple requests from different IMs are sent to the same CM competing the same LO. In this case, the arbiter on the LO grants only one request and forces others to wait until the granted request is withdrawn. The arbitration latency for the blocked requests is prolonged but they will be served eventually.

It should be noticed that the number of simultaneous requests in asynchronous Clos networks are significantly smaller than synchronous Clos networks due to the asynchronous nature. In synchronous Clos networks, all requests are synchronised; therefore the number of simultaneous requests is the total number of active requests. On the other hand, asynchronous Clos networks are not synchronised. When the network load is low, the time to establish a path is much shorter than the time to transmit a packet. The process of establishing a path can be recognised as an event. It is rare for two events to occur at exactly the same time. When the network is saturated, the number of simultaneous requests increases as many requests are blocked. Nevertheless, the number of simultaneous requests is still much smaller than the number in synchronous Clos networks as nearly half IPs are busy transmitting data (49% throughput in uniform traffic as shown in Fig. 8). Using the placed and routed implementations of the synchronous and asynchronous Clos schedulers in Section 6, we have extracted the CM contention rate (the ratio of the number of conflicted requests sent to CMs to the number of all CM requests) of the saturated Clos networks. The rates are 56.7% and 24.9% for the synchronous and asynchronous schedulers, respectively. It is shown that the state feedback scheme successfully reduces the contention significantly. As will be demonstrated in the next section, the state feedback improves throughput in saturated networks.

4 Performance of CRRD and AD

In this section, CRRD and AD algorithms are evaluated with behavioural level models written in SystemC. Schedulers for

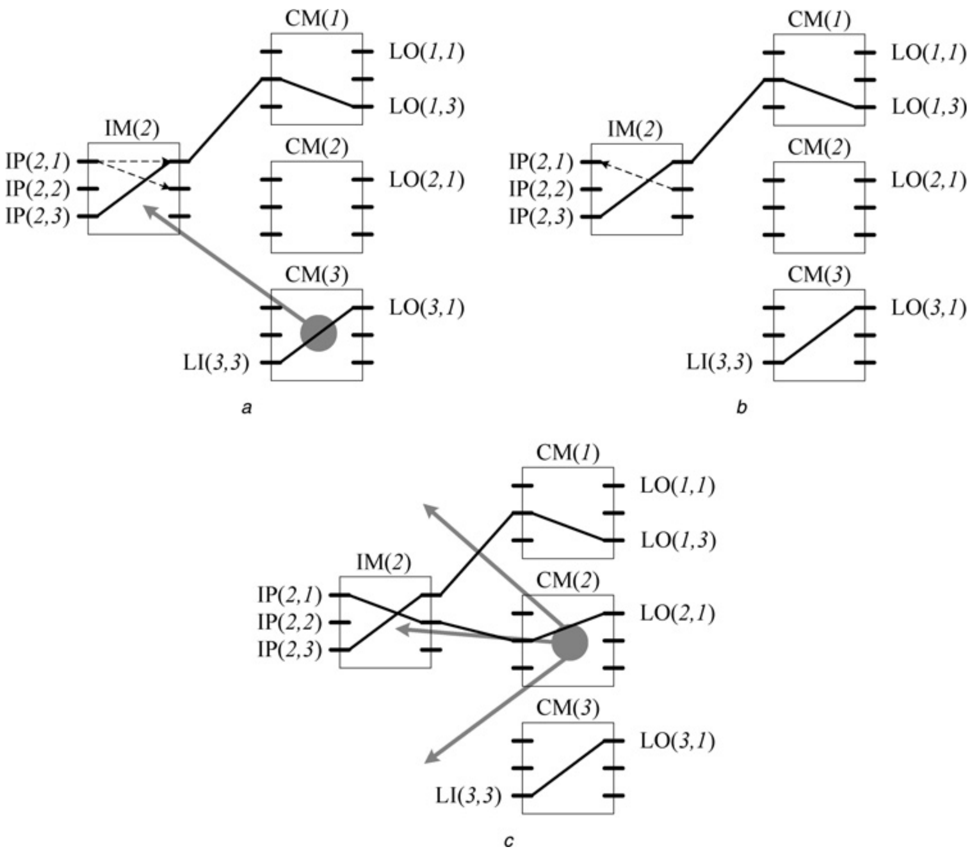


Fig. 4 *Example of AD*

a IP requests
b Grant return
c Path reconfiguration and state feedback

a $C(4, 8, 4)$ S^3 Clos network are built and injected with various traffic patterns. Some assumptions are employed to produce a fair comparison:

- Random arbiters are utilised in both models.
- Requests are withdrawn immediately after a path is configured.
- Latency is normalised in units of a cell time.

Synchronous and asynchronous Clos networks have different hardware implementations. The simulations in this section attempt to reveal the performance differences at the behavioural level. Many hardware details are thus simplified. Both the round-robin arbiters in synchronous Clos networks and the MUTEX or the tree arbiters in asynchronous Clos networks are hardware models approximating random arbiters. Therefore pseudo-random arbiters are directly used in the behavioural models. In synchronous Clos networks, the generation of a new configuration and data transmission run in a pipelined fashion. The arbitration latency is hidden and does not compromise throughput. On the other hand, asynchronous Clos networks cannot pre-calculate a path before the path is fully available. Thus dynamic reconfiguration introduces throughput loss, which will be analysed in Section 6. For the performance comparison of the dispatching algorithms in this section, the arbitration latency is normalised by ignoring the data transmission latency and assuming that the asynchronous arbitration latency is equal to the synchronous arbitration latency – a cell time.

4.1 Non-blocking uniform traffic

In non-blocking uniform traffic, network load spreads to all output ports without any HOL blockage. Therefore the inefficiency of routing algorithms is the sole source of throughput loss. $\rho(s, d)$ is the normalised load between IP(s) and OP(d) where $1 \le s, d \le N$. 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 Poisson process which generates a load with expectation $\bar{\rho}(s) = \sum_{d=1}^{N} \bar{\rho}(s, d)$. The individual load between any IP and OP is

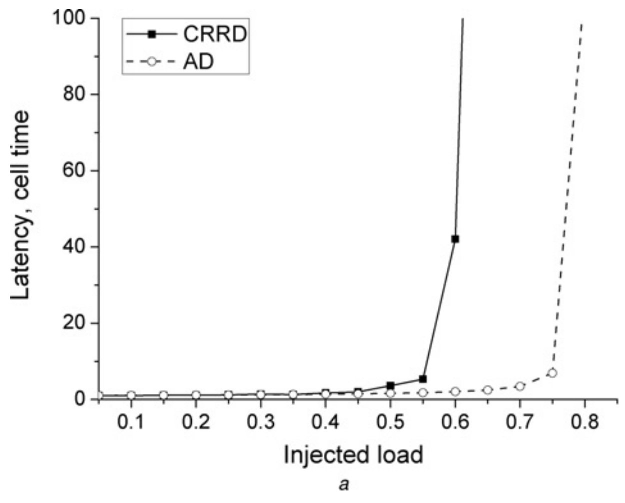
$$\bar{\rho}(s, d) = \frac{\bar{\rho}(s)}{N} \tag{4}$$

For each OP, a prior condition

$$\sum_{s=1}^{N} \rho(s, d) \le 1 \tag{5}$$

is guaranteed to avoid overloaded OPs.

Fig. 5 shows the packet latency and throughput performance using different dispatching algorithms. To achieve the top throughput, the CRRD algorithm runs with four iterations. CRRD is reported to achieve 100% throughput in MSM networks [4] where VOQs are implemented. In an S^3 Clos network, however, an IP is blocked until the blocked cell is successfully forwarded.



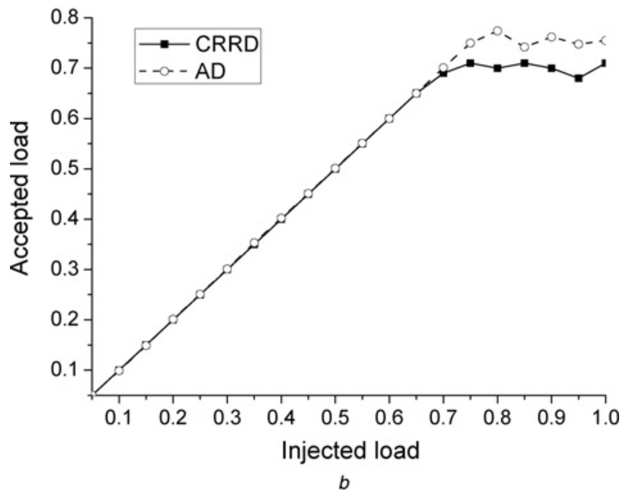


Fig. 5 *Switch performance in non-blocking uniform traffic*

a Packet latency
b Switch throughput

Although CRRD evenly dispatches requests to all CMs, the oblivious request distribution leads to contention in CMs. AD algorithm avoids such contention by utilising the state feedback from CMs in the IM matching sub-algorithm. As shown in Fig. 5*b*, the AD algorithm achieves 76% switch throughput, which is 6% higher than the throughput of the CRRD algorithm.

Heuristic algorithms cannot achieve 100% throughput in a RNB S^3 Clos network even when the traffic is non-blocking. This sub-optimal throughput has two major causes: first, most heuristic algorithms, such as the CRRD algorithm, are oblivious algorithms which cannot resolve the contention in CMs when VOQs are not implemented; secondly, established paths are not allowed to be modified in some Clos networks, including all asynchronous Clos networks. One way to improve the throughput is increasing the number of CMs because it reduces the probability that two requests from different IMs compete for the same LO in one CM [3]. When the number of CMs reaches $2n - 1$, the Clos network is SNB and new paths can be connected without modifying any established paths.

As shown in Fig. 6, both the CRRD and the AD algorithms show better performance with more CMs. The AD algorithm reaches 100% throughput when the Clos network is SNB ($m \geq 7$). Therefore the throughput of AD is solely constricted by the resources occupied by established paths. Once the Clos network has enough CMs to resolve the

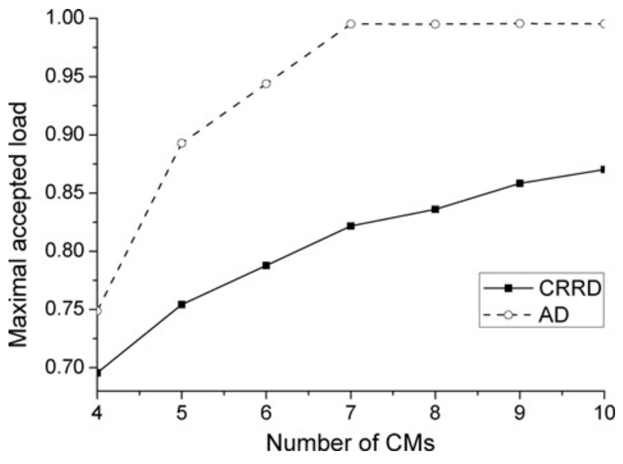


Fig. 6 *Throughput with various number of CMs*

resource bottleneck, AD can deliver all packets within a fixed amount of time. On the other hand, the CRRD algorithm cannot provide 100% throughput even with 10 CMs. Increasing the number of CMs significantly alleviates the contention in CMs but cannot resolve it.

The throughput of the CRRD algorithm is also constricted by the number of iterations in its matching within IMs. Fig. 7 shows the maximal accepted load of CRRD with various numbers of iterations. CRRD achieves the peak throughput with more than four iterations. Since an IM in a $C(4, 8, 4)$ network has four IPs, four iterations are enough to match all possible IPs. Strictly speaking, the AD algorithm also utilises iterations in its IM matching sub-algorithms. However, asynchronous modules are event-driven. Compared with the interval between two continuous requests, the latency of an internal feedback is much shorter and can be ignored. As a result, the AD algorithm always runs with sufficient iterations to provide the optimal throughput.

4.2 Blocking traffic patterns

Traffic patterns in real applications are blocking. Uniform traffic is one of the most analysed synthetic traffic patterns, which is defined as

$$\bar{\rho}(s, d) = \frac{\bar{\rho}(s)}{N} \tag{6}$$

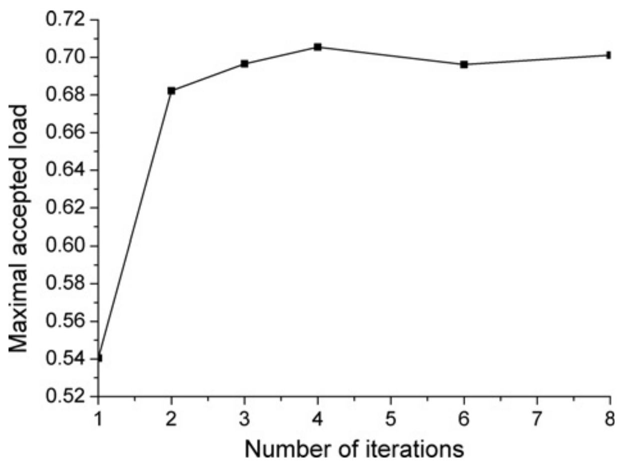


Fig. 7 *Throughput of CRRD with various number of iterations*

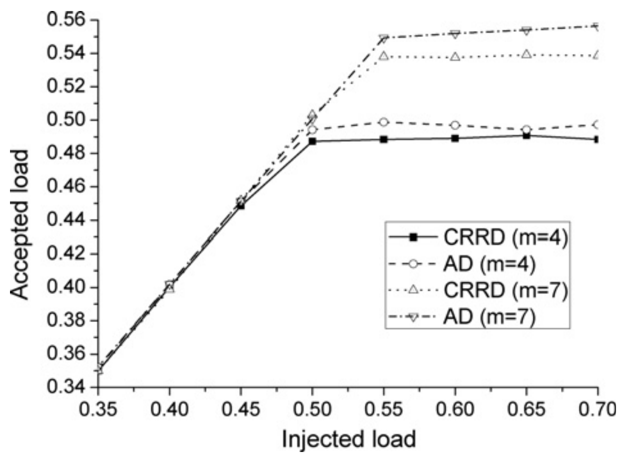


Fig. 8 *Accepted load in uniform traffic*

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

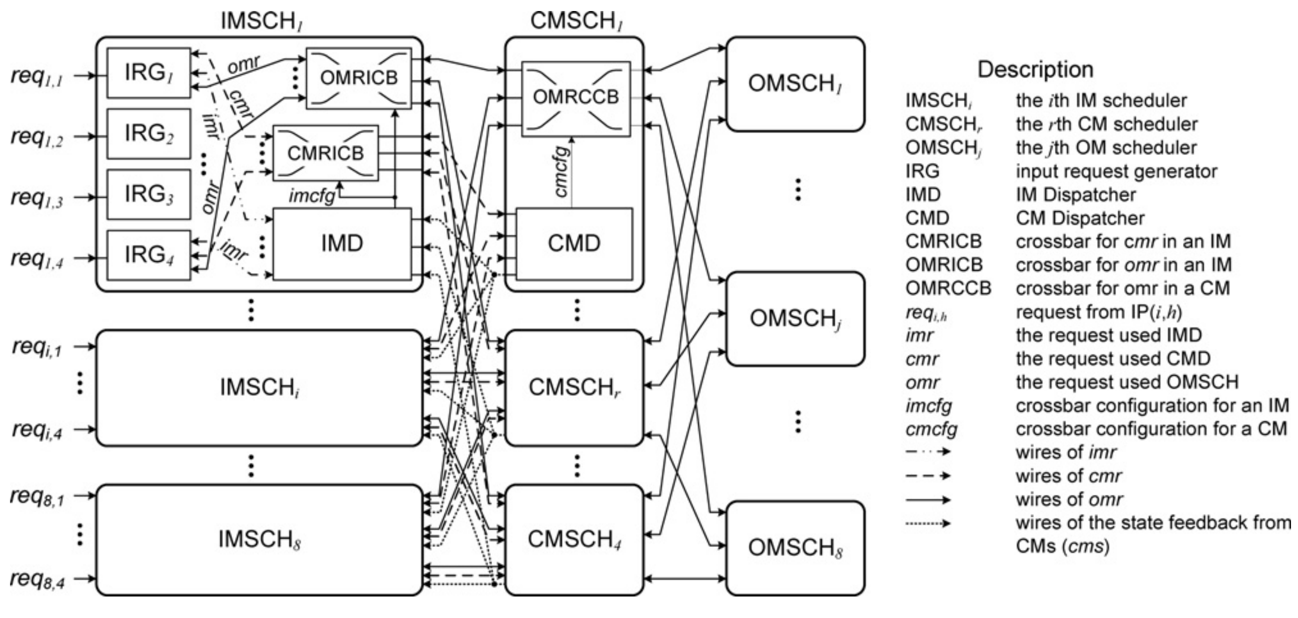
Fig. 8 shows the accepted load under uniform traffic. S^3 Clos networks are input-buffered switching networks. Every IP is connected with an infinite FIFO in simulation. This is different from the buffered IM scheme where an IP is connected to VOQs inside the IM [4]. It is known that the maximal accepted 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 much lower than 58.6%. As shown in Fig. 8, neither CRRD nor AD can provide throughput greater than 50% in RNB Clos networks. The AD algorithm achieves 49.7% throughput, which is 0.8% higher than that of the CRRD algorithm. We have also simulated both algorithms in SNB Clos networks. The maximal accepted load of the AD algorithm increases to 55.4%, which is only 3.2% lower than the optimal accepted load and 1.6% higher than that of the CRRD algorithm. If VOQs are implemented in IMs, CRRD provides 100% throughput in RNB networks because the HOL problem is solved by VOQs [4].

5 Implementation

This section reveals the hardware details of an asynchronous scheduler controlling a 32-port $C(4, 8, 4)$ S^3 Clos network using the AD algorithm. Fig. 9 depicts the overall architecture of the scheduler. Each switching module in the Clos network is reconfigured by a separate scheduler (IMSCH, CMSCH or OMSCH). For each request from an IP, a path is reserved from IM to OM in a forward direction and released from OM to IM in a backward direction. The detailed sequence control will be introduced in Section 5.1.

An IM scheduler (IMSCH) comprises n input request generators (IRGs), one IM dispatcher (IMD) and two dual-directional crossbars. Each IRG receives the request from an IP and translates the request into three different one-hot request signals: IM request (*imr*), CM request (*cmr*) and OM request (*omr*). As indicated by their names, these request signals are used in the schedulers of different switching modules. The IM crossbar is reconfigured by the IMD running the AD algorithm. The IMD receives the *imr* signals from all IRGs in the IM and the state feedback *cms* from all CMs. It selects an available CM and reserves a path in the IM crossbar through the IM configuration bus *imcfg*. This configuration is also sent to CMRICB (*cmr* forwarding crossbar in IM) and OMRICB (*omr* forwarding crossbar in IM). These two crossbars then forward *cmr* and *omr* to the CM selected by the IMD. Note that these two crossbars are dual-directional. Therefore ACK signals (*cmra* or *omra*) are sent back through the same path as configured in these crossbars when a path is reserved in CMs or OMs.

A CM scheduler (CMSCH) contains one CM dispatcher (CMD) and one dual-directional crossbar. CMD receives *cmr* from all IM schedulers. According to the target LO in each request, CMD reserves a path to the LO or block the request until the LO is available. Once a new configuration is made, the internal state of the CMD module is broadcasted to all IMs through the *cms* signals. Similar to IMSCH, the configuration bus *cmcfg* also controls the



IMSCH$_i$	the ith IM scheduler
CMSCH$_r$	the rth CM scheduler
OMSCH$_j$	the jth OM scheduler
IRG	input request generator
IMD	IM Dispatcher
CMD	CM Dispatcher
CMRICB	crossbar for *cmr* in an IM
OMRICB	crossbar for *omr* in an IM
OMRCCB	crossbar for omr in a CM
$req_{i,h}$	request from IP(i,h)
imr	the request used IMD
cmr	the request used CMD
omr	the request used OMSCH
imcfg	crossbar configuration for an IM
cmcfg	crossbar configuration for a CM
-··→	wires of *imr*
--→	wires of *cmr*
→	wires of *omr*
·······→	wires of the state feedback from CMs (*cms*)

Fig. 9 *Architecture of the Clos scheduler*

internal *omr* forwarding crossbar (OMRCCB). Thus, corresponding *omr* signals are forwarded to OMSCHs using the same configuration. OMRCCB is also dual-directional; therefore the ACK signal *omra* is sent back through the same path.

An OM scheduler (OMSCH) is simpler than IMSCH or CMSCH. It receives all *omr* signals forwarded from CMs and tries to reserve a path to the target OP for each request.

5.1 Input request generator

An IP can request one of the *nk* OPs scattered in *k* OMs. In the module matching stage, a path leading to the target OM is reserved using the dispatching algorithm. Then in the port matching stage, a path to the target OP will be reconfigured in the target OM. Therefore the request from an IP needs to be translated into two sub-requests: one for the module matching stage and the other one for the port matching stage. As IMs and CMs have separate dispatcher modules, the sub-request for module matching is further divided into two independent requests.

In our Clos scheduler, every IP is connected with an input request generator as shown in Fig. 10. The incoming request (*req*) is translated into two requests: one used in the dispatching algorithm, identifying an LO to the target OM (*lo_req*), and the other one used in the port matching stage, actually the request used in the target OM (*omr*). As required by normal asynchronous circuits, both *lo_req* and *omr* are one-hot coded. The format translator in IRG converts the coding format of *req* into one-hot. In our implementation, the *req* signals are pre-coded in one-hot and no translation is needed. Although both IMD and CMD use the same request information from *lo_req*, they have different timing requirements. *lo_req* is divided into two independent requests: *imr* and *cmr*. The ACK signals for *imr*, *cmr* and *omr* are *imra*, *cmra* and *omra*, respectively.

The signal transition graph (STG) of the input request generator is shown in Fig. 10*b*. All of *imr*, *cmr* and *omr* are fired immediately after a request is received. Although IM should be the first stage to be reconfigured, *cmr* and *omr* are automatically blocked in the two crossbars (CMRICB

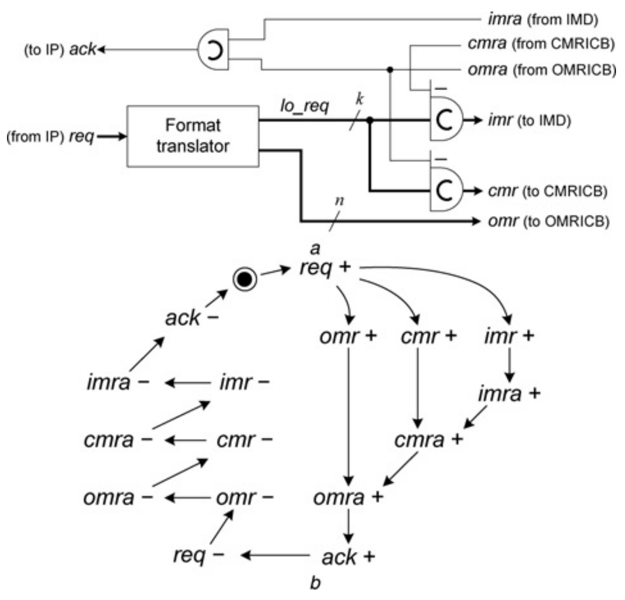


Fig. 10 *Input request generator*

a Schematic
b STG

and OMRICB) inside IM scheduler. Simultaneously firing them with *imr* introduces no side-effect and simplifies the control logic. When *imra* is driven to high by a new configuration in *imcfg*, the IMD successfully reserves a path inside IM. At the same time, *cmr* and *omr* are forwarded to the central stage through CMRICB and OMRICB, respectively. Similar to an IMD, a CMD reconfigures the CM crossbar and *omr* is forwarded to the OM scheduler through OMRCCB inside the CM scheduler. Finally when a positive edge on *omra* is detected in IRG, an acknowledgement is sent back to IP through the *ack* signal and a path is successfully reserved in the Clos network.

The release of a path is more complicated than its reservation. The release sequence must start from OMs and end in IMs. If the path in IMs or CMs is withdrawn before that the path in OMs is safely withdrawn, the release of the path in OMs would not be guarded and the next request would be misrouted. As a result, two asymmetric C-elements are added on *imr* and *cmr* to guarantee that the strict withdrawal sequence is satisfied. When a negative edge on *imra* is detected in IRG, the path is safely withdrawn and IP is acknowledged.

5.2 IM dispatcher

IMDs are the most important and complicated modules in the asynchronous Clos scheduler. An IMD receives requests (*imr*) from all IRGs in the same IM, searches an available CM for each request according to the state feedback from CMs (*cms*) and configures the IM crossbar. The structure of an IMD is shown in Fig. 11. It comprises three components: a request generate matrix, an M−N match allocator and an ACK tree.

As described in Section 3.2, an IP requests only those LIs leading to available LOs. To achieve this selective request scheme, the request-generate matrix acts as a filter where only requests with available LOs are let through. A part of its internal circuit is depicted in Fig. 12*a*. $imr_{i,h,j}$ is one bit of the *k*-bit request signal $imr_{i,h}$ from the IRG connected with IP(i, h). It is high when $imr_{i,h}$ is fired and the target output module is OM(j). There are m LOs leading to OM(j) and their availabilities are identified in the state feedback signals from $cms_{1,j}$ to $cms_{m,j}$, which come from the m CMs. Every pair of $imr_{1,h,j}$ and $cms_{r,j}$ are verified by the asymmetric C-element in Fig. 12*a*. As every bit of the total of $n \times k$ *imr* request bits is paired with m *cms* state bits, there are $n \times k \times m$ C-elements inside one request-generate matrix and the output signals of these C-elements form a three-dimensional matrix, namely *iprm* (the index i in Fig. 12*a* is constant in an IMD). Since each k-bit $imr_{i,h}$ is one-hot coded, the signal vector $\{iprm_{i,h,r,1} - iprm_{i,h,r,k}\}$ is also one-hot coded and is or-reduced into a request bit, namely $ipr_{i,h,r}$, indicating that IP(i, h) attempts to occupy LI(i, r). These *ipr* signals are grouped into n signal vectors ($ipr_{i,1}$–$ipr_{i,n}$), each of which is m bits.

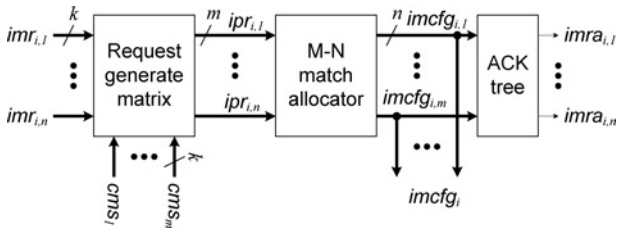


Fig. 11 *IM dispatcher*

The request-generate matrix does not ensure that the verified request reserves an LO without contention. A positive *cms* bit indicates that one LO is currently available but it may trigger multiple requests in different IMs. If these requests are sent to the same CM, contention occurs in the same way as using CRRD. The requests, which fail to reserve the LO, are blocked in the CMD and wait for the LO to be released. In this situation, a pulse is produced on the *cms* bit, but it must not withdraw the requests triggered by itself; otherwise, a false acknowledgment can be produced. The asymmetric C-elements in the request-generate matrix ensure that *cms* signals block incoming requests but the established requests are withdrawn only by themselves rather than *cms*. Avoiding false acknowledgment is the underlying reason that the state feedback cannot solve the existing contention as described previously in Section 3.2.

Fig. 12*b* demonstrates a novel M–N match allocator that matches multiple input ports to multiple output ports concurrently. It utilises the classic parallel iterative matching (PIM) algorithm [23], which is also used in the phase 1 of the CRRD algorithm. In the classic PIM algorithm, requests from input ports are synchronised. PIM uses multiple iterations to reach an even match. The key prerequisite of this algorithm is that once an output port is reserved, the matched input port must withdraw its extra requests to other output ports. In an M–N match allocator, a total number of n IPs compete for m LIs. The requests of IPs come from $ipr_{i,1}$ to $ipr_{i,n}$ and the m LIs are configured by $imcfg_{i,1} - imcfg_{i,m}$. Note that every $ipr_{i,h}$ has m bits requesting all the m LIs. The bit leading to busy LOs are filtered out by the request-generate matrix. When an IP(i, h) fires a request, $ipr_{i,h}$ requests all available LIs concurrently. If an LI is idle, the output-link arbiter on this LI grants the

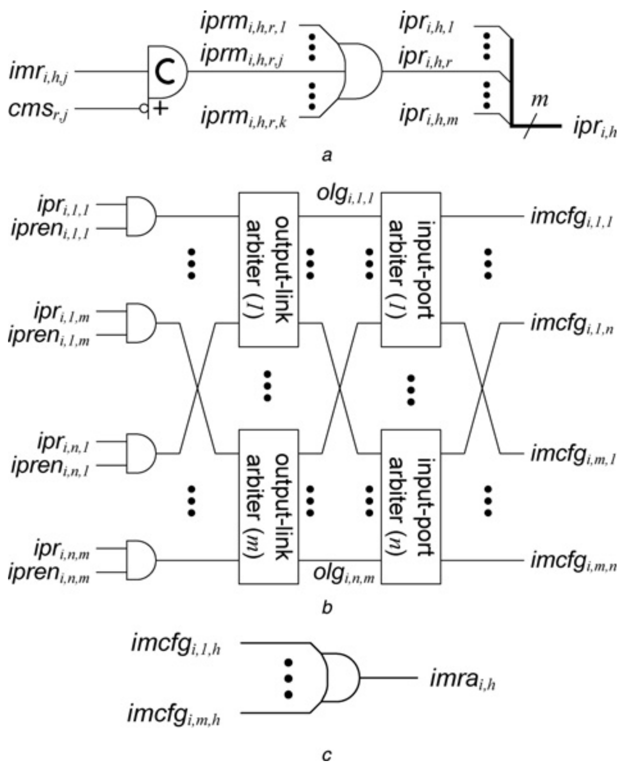


Fig. 12 *Components of IMD*

a Request generate matrix
b M–N match allocator
c Ack tree

request using the *olg* signal. As multiple output-link arbiters may grant the same IP, the input-port arbiter selects one granted LI and drives the corresponding *imcfg* bit to high. As the same as the PIM algorithm, after an LI is reserved, the extra requests to other LIs are withdrawn immediately through the request enable signals ($ipren_{i,1} - ipren_{i,n}$). The generation equation for every $ipren_{i,h,r}$ bit is expressed in (7).

$$ipren_{i,h,r} = \neg\left(\bigcup_{l=1, l\neq r}^{m} imcfg_{i,l,h}\right) \qquad (7)$$

In the literature, there are other allocators that can allocate multiple resources (output ports) to multiple clients (input ports): the forward acting $n \times m$ arbiter [24], the VC admission control presented in QNoC [20] and the multi-resource arbiter [25–27]. The VC admission control treats all clients fairly but resources are selected by an unbalanced static priority arbiter [28]. The multi-resource arbiter is a quasi-delay-insensitive (QDI) allocator that allocates resources fairly. We have utilised the multi-resource arbiter in our original scheduler design [29] but it introduces large area overhead. Both VC admission control and the multi-resource arbiter allocate resources in a serialised way that causes extra arbitration latency. The forward acting $n \times m$ arbiter is a speed independent allocator that can allocate multiple requests in parallel.

The M–N match allocator is much smaller than the multi-resource arbiter. It uses a similar structure as the forward acting $n \times m$ arbiter and it allocates requests in parallel. However, it is not speed independent or QDI because the withdrawn of requests and *olg* is unguarded for less area overhead. Fig. 13 illustrates the STG of an M–N match allocator with two requests (ipr_1 and ipr_2) and two resources. To simplify the problem, we assume the request-generate matrix blocks no requests. Thus the input request *ipr* is always duplicated to all output-link arbiters. The forward arbitration transitions are depicted in bold lines and the backward request withdrawn transitions are drawn in slim lines. The unguarded completion check is highlighted in dash lines. As an example, supposing the second resource is allocated to the first request, $imcfg_{2,1}$ is driven to high and the duplicated request sent to output-link arbiter (1) should be withdrawn. As the second request can arrive at any time, the output-link arbiter (1) may select ipr_1 ($olg_{1,1}+$) or block ipr_1 as ipr_2 arrives already. In either case, the AND gate on the duplicated request $ipr_{1,1}$ in Fig. 12*b* releases $ipr_{1,1}$ and $olg_{1,1}$. For correct operation, this withdrawn process must finish before ipr_1- otherwise $olg_{1,1}$ can produce a fake acknowledgment. Instead of using complicated completion detection circuits to enforce the speed independent requirement as shown by the dash line, we found that practical hardware implementations ensure correct operation. The timing requirement is expressed in (8).

$$t_{imcfg+\rightarrow olg-} < t_{imcfg+\rightarrow ipr-} \qquad (8)$$

In the overall transition graph in Fig. 10*b*, $imra+$ is triggered by $imcfg+$ and $ipr-$ is triggered by $imr-$. The right side of (8) is the accumulative latency of reserving and releasing a path in CMs and OMs, together with the whole data transmission delay. The left side of (8), on the other hand, is merely the accumulative latency of an OR gate tree expressed by (7), a two-input AND gate and an output-link arbiter. It will be

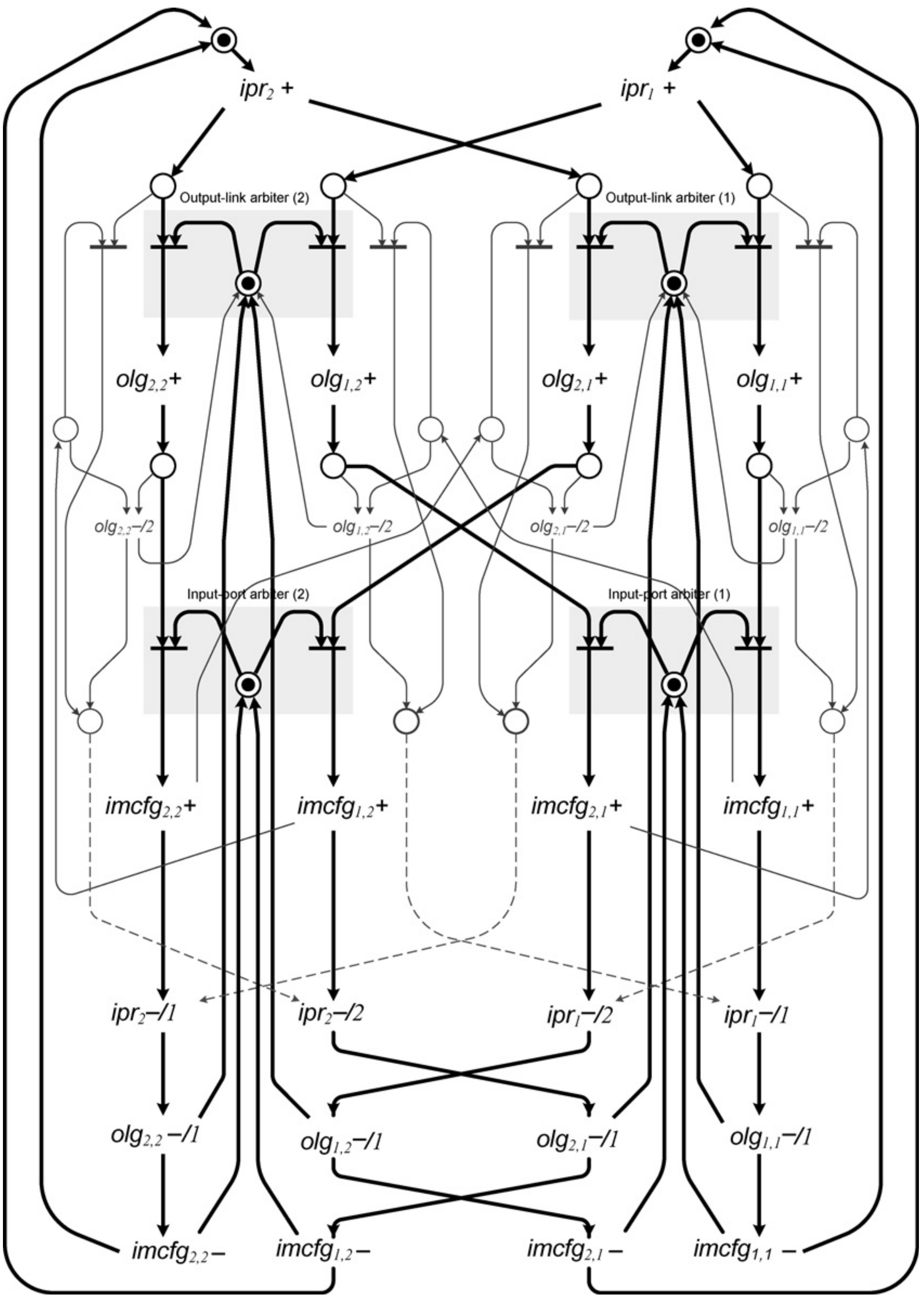


Fig. 13 *STG of a 2 × 2 M–N match allocator*

shown in Section 6.1 that the right side is far longer than the left side even without data transmission.

As shown in Fig. 12*c*, the ACK signals *imra* are generated from the configuration bus *imcfg* using the OR gate trees inside the ACK tree. These ACK signals are then sent to IRGs.

5.3 CM dispatcher

CMDs 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 granting requests from all input ports.

Fig. 14 shows the internal structure of a CMD. As every CM has k output ports, there are k arbiters in each CMD. Each arbiter receives requests from all the k input ports and generates the configuration signal for its output port. $cmr_{r,i,j}$ is a 1-bit request forwarded from IMs indicating that an IP in IM(i) is competing for the LO(r, j) in CM(r). $cmcfg_{r,j,i}$ is set to high when the arbiter on LO(r, j) grants the request $cmr_{r,i,j}$. When a new configuration is made, an

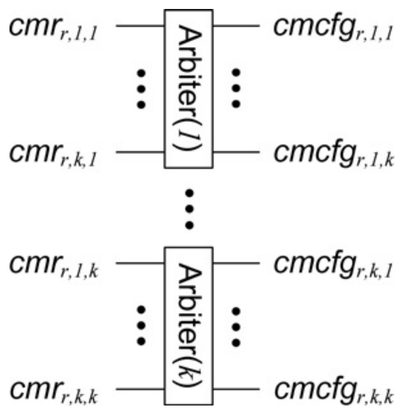


Fig. 14 *CM dispatcher*

acknowledgment is sent back to the IRG using the reserved path and the state is broadcasted to all IMs using the state feedback signal *cms*. The ACK signals *cmra* and the state feedback *cms* are generated using OR gate trees similar to the tree shown in Fig. 12*c*. Their generation equations are expressed as follows

$$cmra_{r,i} = \bigcup_{l=1}^{k} cmcfg_{r,l,i} \tag{9}$$

$$cms_{r,j} = \bigcup_{l=1}^{k} cmcfg_{r,j,l} \tag{10}$$

5.4 OM scheduler

An OM scheduler reconfigures an OM using the same arbitration structure as a CMD. Each OP has an arbiter that receives requests from all input LOs in the same OM and makes a grant when the OP is available. The requests are forwarded from the input request generators in IM schedulers through the path reserved in OMRICBs and OMRCCBs. The ACK signals are also generated using the same logic as in CMD. The generation equation is expressed below

$$omra_{j,r} = \bigcup_{l=1}^{n} omcfg_{j,l,r} \tag{11}$$

where $omra_{j,r}$ is the ACK signal from OM(j) to CM(r) and $omcfg_{j,h,r}$ is the configuration bit controlling the connection between LO(j, r) and OP(j, h).

6 Hardware performance

Three different 32-port $C(4, 8, 4)$ S^3 Clos networks have been implemented in this paper: an asynchronous Clos network with data switches using the channel-sliced pipelines [30] (A-SC), another asynchronous Clos network with data switches using bundled-data pipelines (A-BD) and a synchronous Clos network (Syn). Both asynchronous Clos networks are reconfigured by the same scheduler using the AD algorithm and the synchronous Clos network is reconfigured by a classic scheduler using the CRRD algorithm [4].

All designs are synthesised, placed and routed with commercial tools using the Faraday 0.13 μm standard cell library based on the UMC 0.13 μm technology. All basic asynchronous cell elements, such as the C-element and the 2-input MUTEX cell, are manually written in gate-level Verilog HDL using only standard cells. Accurate latency and throughput is obtained from the post-layout netlists co-simulated with test benches written in SystemC. Gate and wire latencies are extracted and back-annotated in all simulations. The power consumption of different Clos networks is obtained from the Synopsys PrimeTime^{TM} PX tool suite using the toggle rates from simulations and the accurate RC information extracted from the routed layout.

The asynchronous Clos scheduler is self-timed with a timing requirement described in (8). The channel-sliced pipeline [30] is a fast QDI pipeline style which transmits data packets through a number of unsynchronised 1-of-4 sub-channels [31]. Although the QDI data switches are tolerant to temperature, power and process variation, and dissipates extremely low power during idle states, it introduces extra power and area overhead. As an alternative, an asynchronous Clos network using bundled-data pipelines in its data switches is also implemented. The bundled-data pipeline is self-timed. Although it consumes extremely low power during idle states, it may suffer from variation and timing closure is problematic. Compared with the QDI data switches, the bundled-data data switches introduce much lower power and area overhead.

The scheduler in the synchronous Clos network is a reproduction of the CRRD algorithm described in [4]. Its structure is similar to that of the asynchronous scheduler shown in Fig. 9, but all components are synchronised with the global clock. A global state machine is added in the scheduler to control iterations. Since synchronous circuits handle binary codes easily and the sequence control problem is now resolved by the global state machine, no IRG module is needed. After detailed optimisation, the synchronous Clos network can run at as high as 300 MHz after layout. The final clock period is set to 3.5 ns (285 MHz) because running at 300 MHz introduces significant area overhead on buffers. The number of iterations is dynamically reconfigurable and the length of a cell time is

$$t_{\text{cell}} = t_{\text{clock}}(N_{\text{iteration}} + 1) \tag{12}$$

where $N_{\text{iteration}}$ is the number of iterations.

6.1 Basic implementation results

The area consumption of all Clos implementations, including the original asynchronous Clos scheduler in [29] (A-Orig) and an asynchronous crossbar using bundled-data pipelines (Crossbar-BD), are demonstrated in Table 1. The data width of each I/O port is 32 bits.

The synchronous Clos network has the smallest total area due to its smallest data switches. Asynchronous circuits introduce extra area overhead on data paths. For bundled-data pipelines, although data are transmitted in binary as the synchronous data path, extra single-rail latch control circuits are inserted as required by the self-timed handshake protocol. In the channel-sliced pipelines, every two data digits are translated into a 4-bit one-hot code word and every sub-channel delivering this 4-bit code word needs an extra ACK wire as sub-channels are not synchronised. In summary, utilising asynchronous circuits introduces area overhead in data paths due to handshake protocols and code styles.

On the other hand, the proposed asynchronous scheduler is smaller than its synchronous counterpart. Three reasons lead to this outcome: (i) the storage elements used in asynchronous

Table 1 Area consumption, μm^2

	A-CS	A-BD	A-Orig [29]	Crossbar-BD	Syn [4]
scheduler	88 057	88 191	260 740	82 344	115 262
one IMD	3862	3870	21 882	–	3186
one CMD	4879	4803	8437	–	6498
one OMSCH	985	992	1258	–	1375
one IRG	160	163	196	–	–
switch	347 276	146 804	–	251 089	97 454
total	435 333	234 995	–	333 433	212 716

circuits are C-elements which are smaller than flip-flops. (ii) As synchronous circuits are synchronised and clocked, extra storage elements are inserted where the latency of one operation, such as the iterations, is longer than one clock period. (iii) Since synchronous scheduler pre-calculates the configuration for the next cell time, the generated configuration is stored in flip-flops as an extra pipeline stage. Owing to these reasons, all asynchronous scheduler components are smaller than their synchronous counterparts except the IMD. The request-generate matrix in every IMD uses the state feedback from CMs to avoid the contention in CMs. As depicted in Fig. 11, every request-generate matrix contains an $n \times k \times m$ matrix of C-elements, which causes the large area consumption of IMDs.

It is also shown in Table 1 that the asynchronous scheduler implemented in this paper achieves a significant area reduction of 66% from its original design. The original design utilises the multi-resource arbiter which sequentially allocates multiple input ports to multiple output ports. As every I/O pair has m different paths through the m CMs, each IMD has m multi-resource arbiters running in parallel. These parallel arbiters lead to the major area overhead. We have also optimised the state feedback logic and timing constraints for the new implementation.

The asynchronous crossbar using bundled-data pipelines is scheduled by 32 tree arbiters, one per individual output port. The bundled-data Clos switch demonstrates 41% area reduction but the Clos scheduler is 7% larger than tree arbiters. It is normal to produce large Clos schedulers as the scheduling problem of Clos networks is more complicated than that of crossbars. The area reduction in data switches compensates the area overhead of schedulers and the overall area is reduced.

The detailed latency is labelled in the simplified STG shown in Fig. 15. The transitions from *req*+ to *ack*+ denote the path reservation procedure and the transitions from *req*− to *ack*− denote the path release procedure. During the interval between *ack*+ and *req*−, data are being transmitted through the path reserved in the Clos network. The latencies labelled are averaged from different IPs requiring various OPs in an idle Clos network. When the Clos network is busy, these latencies are related to the load as some requests are blocked. The asynchronous Clos network can reserve a path in 4.44 ns and release it in 5.16 ns. The minimal allocation period is 9.6 ns, which is 4.9% shorter than the 10.1 ns period of the original design [29]. Apropos of the timing assumption (8) of the M−N match allocator, the left side $t_{imcfg+\rightarrow olg-}$ is around 0.69 ns, which is far shorter than the 6.73 ns latency of the right side without data transmission.

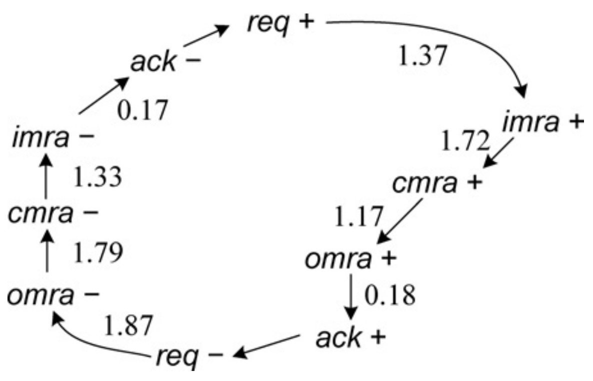


Fig. 15 *Speed performance, ns*

For the data switches, the channel-sliced Clos network is slightly slower than the bundled-data one. In the channel-sliced Clos network, the average period for one data transmission is 4.9 ns whereas it is 4.6 ns in the bundled-data Clos network. However, both asynchronous switch implementations are much slower than the synchronous Clos network which can easily run at more than 400 MHz (less than 2.5 ns). The reason for this low speed is straightforward. Both asynchronous Clos networks use four-phase handshake protocols [14] that require four transitions in one cycle whereas the synchronous pipeline requires only one. Some techniques can be used to reduce the period, such as the two-phase handshake protocols [14, 32], the lookahead pipeline [33], the GasP pipeline [34] or inserting pipeline stages inside the Clos network. As the key research issue in this paper is the routing algorithm, we will not exploit these techniques.

It is possible to evaluate the consistency between the hardware implementation and the behaviour models used in Section 4. The SystemC model in Section 4 is now back-annotated with the latencies shown in Fig. 15. The same SystemC test bench loads both the SystemC model and the post-layout netlist with the uniform traffic model described in Section 4.2. As shown in Fig. 16, the SystemC model accurately matches the post-layout netlist.

6.2 Comparison among implementations

Fig. 17 reveals the packet latency in uniform traffic. According to (12), the cell time for the synchronous Clos network is set to five clock cycles to guarantee an even request distribution. Using the same assumption in Section 4, we assume a packet contains only one cell and every cell comprises 20 bytes of data (32 bits/cycle for five cycles). The packets delivered in asynchronous Clos networks have the same length as in synchronous Clos networks.

When network load is low, asynchronous Clos networks show shorter packet latency thanks to their faster scheduler. As described in Section 6.1, the asynchronous scheduler can reserve a path in 4.44 ns. The synchronous Clos scheduler needs a cell time to calculate a new configuration.

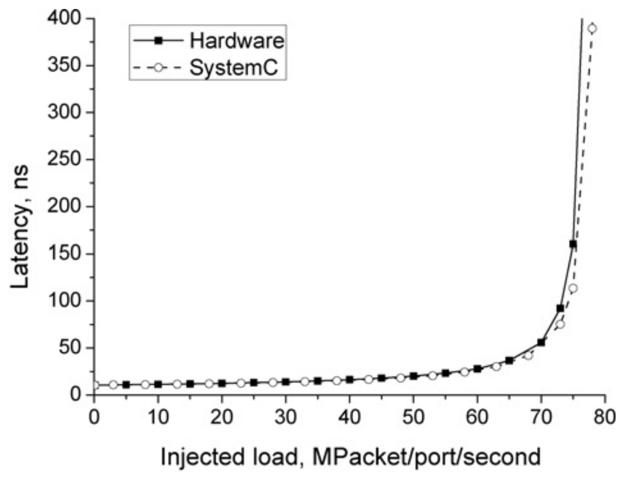


Fig. 16 *Consistency between hardware and behaviour simulation*

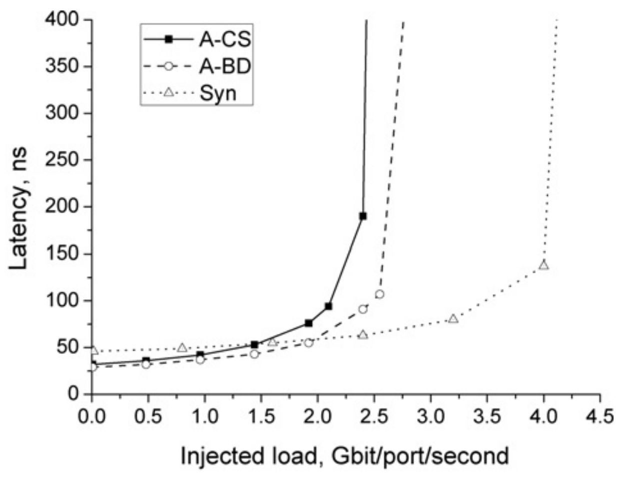


Fig. 17 *Packet latency with four iterations*

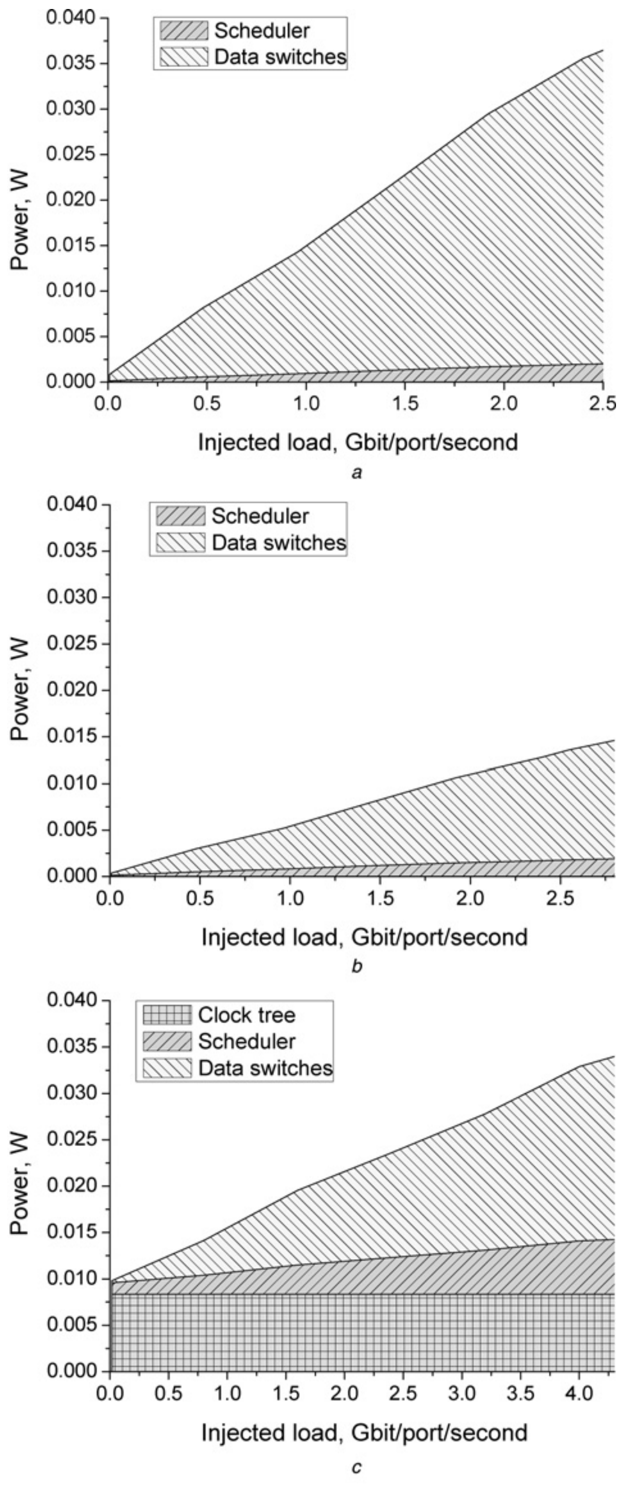


Fig. 18 *Power consumption with four iterations*

a A-CS
b A-BD
c Syn

In this test case, a cell time is 17.5 ns (12), which is 2.9 times longer than the 4.44 ns in asynchronous Clos networks. However, when networks are heavily loaded, the throughput performance is affected by data transmission latency rather than the arbitration latency. As the periods for data in asynchronous Clos networks are much longer than the clock period in synchronous Clos networks and the arbitration latency is not hidden, the maximal accepted load of asynchronous Clos networks is significantly smaller than synchronous Clos networks.

Fig. 18 shows the power consumption of all Clos networks. The power of all Clos networks increases with the injected load. When networks are idle, both asynchronous Clos networks demonstrate nearly zero power dissipation but the synchronous Clos network consumes 9.8 mW of which 85% is dissipated on the clock tree. When networks are heavily loaded, the bundled-data Clos network shows the best power efficiency whereas the channel-sliced Clos network consumes the most power. Specifically, when the network load is 2.4 Gbit/port/second, the channel-sliced, the bundled-data and the synchronous Clos networks consume 35.6, 12.8 and 23.8 mW, respectively.

The significant power consumption of the channel-sliced Clos network is related to the QDI handshake protocol and the 1-of-4 code style. In synchronous data paths, an average of 50% of the 32 data wires turn over every cycle, which is 16 toggles per cycle. In the bundled-data pipelines, data wires have the same toggle rate but the single-rail latch control logic transit four times every cycle period. When the data width is large, the extra toggle rate on latch control logic can be ignored; therefore the bundled-data and the synchronous Clos networks have similar power consumption on data switches. When the QDI 1-of-4 pipelines are utilised, every two data bits are translated into a 4-bit code word. In each cycle, one wire in this 4-bit code word transits twice, which causes 32 toggles per cycle period. Every ACK wire also transits twice every cycle. As the channel-sliced pipelines utilise unsynchronised sub-channels, the overall toggle rate is around 64 toggles per cycle. The high toggle rate of the channel-sliced Clos network leads to its high power consumption. Using traditional 1-of-4 pipelines with a common ACK wire can reduce the toggle rate to 34 toggles per cycle. However, the completion detection tree on every pipeline stage consumes extra power and compromises the speed performance significantly. The area and speed of traditional and channel-sliced pipelines has been compared in [30].

The asynchronous Clos scheduler consumes low power. As shown in Fig. 18, when the network load is 2.4 Gbit/port/second, the asynchronous scheduler consumes 1.7 mW, whereas the synchronous scheduler consumes 12.3 mW (including the power of the clock tree as data switches contain no flip-flops). Thanks to this small power consumption, the bundled-data Clos network shows the best power efficiency.

The length of a cell time is an important design parameter. In synchronous Clos networks, it determines the maximal number of iterations in the CRRD algorithm (12). If the number of iterations is less than n, the CDDR algorithm cannot guarantee an even request distribution. In asynchronous Clos networks, a cell is equivalent to a packet; therefore the length of a cell controls the amount of data being transmitted in one packet and the highest frequency that a path is reconfigured.

Fig. 19 shows the throughput of all Clos networks with various packet lengths (cell time). The throughput of synchronous Clos networks is stable when the cell time is more than four clock cycles. As described in the original paper of the PIM algorithm [23], the average number of iterations C required to match an $N \times N$ crossbar is

$$E[C] \leq \log_2 N + \frac{4}{3} \tag{13}$$

As the PIM algorithm is used in the CRRD algorithm and in (13), $N = n$ (the number of IPs in one IM), $E[C] = 3.33$. On average, a cell time larger than 4.3 is enough to reach an even request distribution, which is demonstrated in Fig. 19.

For asynchronous Clos networks, throughput increases monotonically with packet length. Unlike the synchronous Clos network where new configuration is pre-calculated, a path must be strictly idle before it can be re-allocated. As a result, the bandwidth of certain data paths is wasted during the arbitration process. When the length of packets increases, the frequency of reconfiguration decreases and throughput increases.

The power consumption of Clos networks with various packet lengths (cell time) is revealed in Fig. 20. The power of both the bundled-data and the synchronous Clos networks decreases along with the packet length because of the low reconfiguration frequency with long packets. On the

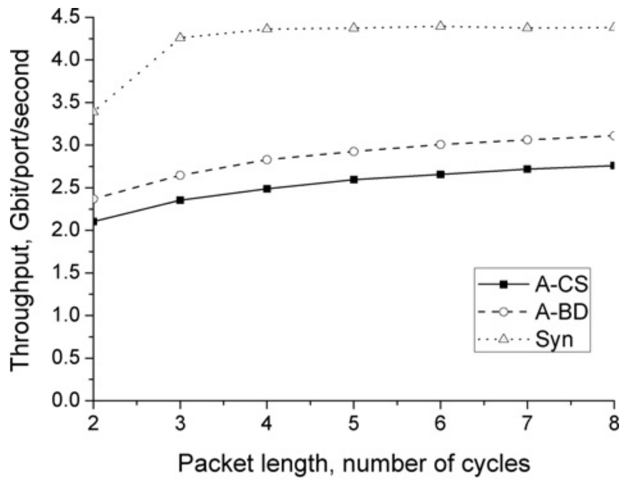


Fig. 19 *Throughput with various packet lengths*

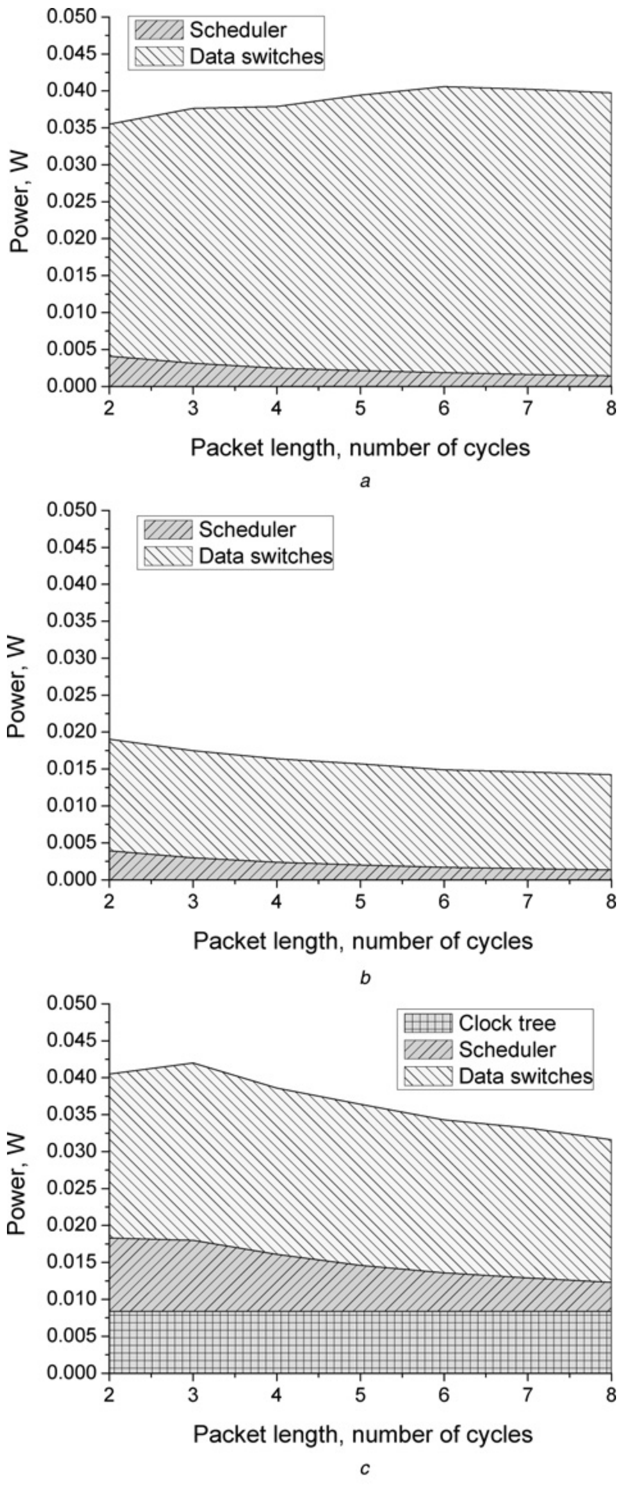


Fig. 20 *Power consumption with various packet lengths*

a A-CS
b A-BD
c Syn

contrary, the power of the channel-sliced Clos network slightly increases. Although the power of the asynchronous scheduler decreases as expected, the power of data switches increases as the maximal accepted load increases in the meanwhile. As described in the previous simulation (Fig. 18), QDI data switches consume a significant amount of power. Since the power saved from the scheduler cannot compensate the extra power consumed by the increased throughput, the total power consumption rises.

7 Conclusions

In this paper, the first asynchronous dispatching (AD) algorithm for general unbuffered three-stage Clos networks is proposed and implemented. Behavioural-level simulations have been made to compare the performance of the AD algorithm against the classic CRRD algorithm in S^3 Clos networks. The CRRD algorithm dispatches requests evenly to all CMs but this distribution is oblivious and leads to the contention in CMs. The AD algorithm introduces a state feedback scheme. By utilising the state information from CMs, AD avoids the contention in CMs. Behavioural simulation results show that the AD algorithm outperforms the CRRD algorithm in all traffic patterns. When the traffic is non-blocking, the AD algorithm achieves 100% throughput in SNB Clos networks. When the traffic is blocking, the AD algorithm shows only 3.2% throughput loss compared with the theoretically optimal throughput.

Three 32-port $\boldsymbol{C}(4, 8, 4)$ S^3 Clos networks have been implemented using the Faraday 0.13 μm cell library: two asynchronous Clos networks using channel-sliced and bundled-data data switches and a synchronous Clos network. The AD and CRRD algorithms are utilised in asynchronous and synchronous Clos networks, respectively. Post-layout simulations show that the asynchronous Clos scheduler can reserve a path in 4.44 ns and release it in 5.16 ns.

Different Clos implementations demonstrate their own advantages. The synchronous Clos network supports the highest throughput but it is power consuming and not tolerant to variation. Both asynchronous Clos networks consume little power and show shorter packet latency than synchronous Clos networks when the network is not heavily loaded. The bundled-data Clos network shows the best power efficiency in all Clos implementations and outperforms the channel-sliced Clos network in throughput. If we consider schedulers only, the asynchronous scheduler is more power and area-efficient than the synchronous scheduler and is tolerant to variation.

The authors are currently integrating the asynchronous Clos network into the asynchronous on-chip networks. Compared with traditional five ports routers in mesh networks, using high-radix routers reduces communication latency and improves throughput. However, the size of the internal crossbar increases quadratically with the port number. Clos networks can be used to replace these crossbars and reduce the area overhead.

8 Acknowledgments

Part of this paper has been published in the 10th International Conference on Application of Concurrency to System Design (ACSD 2010) [29]. The authors appreciate the helpful review comments from all the anonymous reviewers. This work is supported by EPSRC EP/E06065X/1.

9 References

1 Clos, C.: ‘A study of nonblocking switching networks’, *Bell Syst. Tech. J.*, 1953, **32**, (5), pp. 406–424

2 Chao, H.J., Lam, C.H., Oki, E.: ‘Broadband packet switching technologies: a practical guide to ATM switches and IP routers’ (Wiley, 2001)

3 Chiussi, F.M., Kneuer, J.G., Kumar, V.P.: ‘Low-cost scalable switching solutions for broadband networking: the ATLANTA architecture and chipset’, *IEEE Commun. Mag.*, 1997, **35**, (12), pp. 44–53

4 Oki, E., Jing, Z., Rojas-Cessa, R., Chao, H.J.: ‘Concurrent round-robin-based dispatching schemes for Clos-network switches’, *IEEE/ACM Trans. Netw.*, 2002, **10**, (6), pp. 830–844

5 Chao, H.J., Jing, Z., Liew, S.Y.: ‘Matching algorithms for three-stage bufferless Clos network switches’, *IEEE Commun. Mag.*, 2003, **41**, (10), pp. 46–54

6 Chao, H.J., Deng, K.L., Jing, Z.: ‘PetaStar: a petabit photonic packet switch’, *IEEE J. Sel. Areas Commun.*, 2003, **21**, (7), pp. 1096–1112

7 Cheyns, J., Develder, C., Breusegem, E.V., *et al.*: ‘Clos lives on in optical packet switching’, *IEEE Commun. Mag.*, 2004, **42**, (2), pp. 114–121

8 Rojas-Cessa, R., Lin, C.B.: ‘Scalable two-stage Clos-network switch and module-first matching’. Proc. Workshop on High Performance Switching and Routing, 2006, pp. 303–308

9 Oki, E., Kitsuwan, N., Rojas-Cessa, R.: ‘Analysis of space–space–space Clos-network packet switch’. Proc. Int. Conf. Computer Communications and Networks, 2009, pp. 1–6

10 Kim, J., Dally, W.J., Towles, B., Gupta, A.K.: ‘Microarchitecture of a high radix router’. Proc. Int. Symp. on Computer Architecture, 2005, pp. 420–431

11 Gómez, C., Gómez, M.E., López, P., Duato, J.: ‘Exploiting wiring resources on interconnection network: increasing path diversity’. Proc. Euromicro Conf. Parallel, Distributed and Network-Based Processing, 2008, pp. 20–29

12 Scott, S., Abts, D., Kim, J., Dally, W.J.: ‘The BlackWidow high-radix Clos network’, *SIGARCH Comput. Archit. News*, 2006, **34**, (2), pp. 16–28

13 Leroy, A., Milojevic, D., Verkest, D., Robert, F., Catthoor, F.: ‘Concepts and implementation of spatial division multiplexing for guaranteed throughput in networks-on-chip’, *IEEE Trans. Comput.*, 2008, **57**, (9), pp. 1182–1195

14 Sparsø, J., Furber, S.: ‘Principles of asynchronous circuit design – a systems perspective’ (Kluwer Academic Publishers, 2001)

15 Krstić, K., Grass, E., Gürkaynak, F.K., Vivet, P.: ‘Globally asynchronous, locally synchronous circuits: overview and outlook’, *IEEE Des. Test Comput.*, 2007, **24**, (5), pp. 430–441

16 Song, W., Edwards, D.: ‘Asynchronous spatial division multiplexing router’, *Microprocess. Microsyst.*, 2011, **35**, (2), pp. 85–97

17 McKeown, N., Mekkittikul, A., Anantharam, V., Walrand, J.: ‘Achieving 100% throughput in an input-queued switch’, *IEEE Trans. Commun.*, 1999, **47**, (8), pp. 1260–1267

18 Karol, M.J., Hluchyj, M.G., Morgan, S.P.: ‘Input versus output queuing on a space-division packet switch’, *IEEE Trans. Commun.*, 1987, **35**, (12), pp. 1347–1356

19 Dally, W.: ‘Virtual-channel flow control’, *IEEE Trans. Parallel Distrib. Syst.*, 1992, **3**, (2), pp. 194–205

20 Dobkin, R.R., Ginosar, R., Kolodny, A.: ‘QNoC asynchronous router’, *Integr. VLSI J.*, 2009, **42**, (2), pp. 103–115

21 Josephs, M.B., Yantchev, J.T.: ‘CMOS design of the tree arbiter element’, *IEEE Trans. VLSI*, 1996, **4**, (4), pp. 472–476

22 Kinniment, D.J.: ‘Synchronization and arbitration in digital systems’ (Wiley, 2007)

23 Anderson, T.E., Owicki, S.S., Saxe, J.B., Thacker, C.P.: ‘High-speed switch scheduling for local-area networks’, *ACM Trans. Comput. Syst.*, 1993, **11**, (4), pp. 319–352

24 Patil, S.S.: ‘Forward acting $n \times m$ arbiter’ (Computation Structures Group, Massachusetts Institute of Technology, Memo 67, 1972)

25 Golubcovs, S., Shang, D., Xia, F., Mokhov, A., Yakovlev, A.: ‘Modular approach to multi-resource arbiter design’. Proc. IEEE Int. Symp. Asynchronous Circuits and Systems, 2009, pp. 107–116

26 Golubcovs, S., Shang, D., Xia, F., Mokhov, A., Yakovlev, A.: ‘Multi-resource arbiter decomposition’, Newcastle University, Technical Report, 2009, NCL-EECE-MSD-TR-2009-143

27 Shang, D., Xia, F., Golubcovs, S., Yakovlev, A.: ‘The magic rule of tiles: virtual delay insensitivity’. Proc. Int. Workshop on Power and Timing Modelling, Optimization and Simulation, 2009, pp. 286–296

28 Bystrov, A., Kinniment, D., Yakovlev, A.: ‘Priority arbiters’. Proc. IEEE Int. Symp. Asynchronous Circuits and Systems, 2000, pp. 128–137

29 Song, W., Edwards, D.: 'An asynchronous routing algorithm for Clos networks'. Proc. Int. Conf. Application of Concurrency to System Design, 2010, pp. 67–76

30 Song, W., Edwards, D.: 'A low latency wormhole router for asynchronous on-chip networks'. Proc. Asia and South Pacific Design Automation Conf., 2010, pp. 437–443

31 Bainbridge, J., Furber, S.: 'Chain: a delay-insensitive chip area interconnect', *IEEE Micro*, 2002, **22**, pp. 16–23

32 Sutherland, I.E.: 'Micropipelines', *Commun. ACM*, 1989, **32**, (6), pp. 720–738

33 Singh, M., Nowick, S.M.: 'The design of high-performance dynamic asynchronous pipelines: lookahead style', *IEEE Trans. VLSI*, 2007, **15**, (11), pp. 1256–1269

34 Sutherland, I., Fairbanks, S.: 'GasP: a minimal FIFO control'. Proc. Int. Symp. Asynchronous Circuits and Systems, 2001, pp. 46–53