High-Performance, Low-Complexity Deadlock Avoidance for Arbitrary Topologies/Routings

DOI:
10.1145/3205289.3205307

Document Version
Accepted author manuscript

High-Performance, Low-Complexity Deadlock Avoidance for Arbitrary Topologies/Routings

Jose A. Pascual
The University of Manchester
Manchester, United Kingdom
jose.pascual@manchester.ac.uk

Javier Navaridas
The University of Manchester
Manchester, United Kingdom
javier.navaridas@manchester.ac.uk

ABSTRACT

Recently, the use of graph-based network topologies has been proposed as an alternative to traditional networks such as tori or fat-trees due to their very good topological characteristics. However they pose practical implementation challenges such as the lack of deadlock avoidance strategies. Previous proposals are either exceedingly complex, underutilise network resources or lack flexibility. We propose–and prove formally–three generic, low-complexity deadlock avoidance mechanisms that only require local information. The main strengths of our method are its topology- and routing-independence and that the virtual channel count is bounded by the length of the longest path. We evaluate our proposed mechanisms against previous proposals through an extensive simulation study to measure the impact on the performance using both synthetic and realistic traffic. First we compare against a well-known HPC mechanism for dragonfly and achieved similar performance level. Then we moved to Graph-based networks and show that our mechanisms can greatly outperform traditional, spanning-tree based mechanisms, even if these use a much larger number of virtual channels. Overall, we find that our proposal provides a simple, flexible and high performance deadlock-avoidance solution.

KEYWORDS

Deadlock avoidance; Arbitrary network topologies/routing policies; Virtual channels; Regular random graphs

ACM Reference Format:
Jose A. Pascual and Javier Navaridas. 2018. High-Performance, Low-Complexity Deadlock Avoidance for Arbitrary Topologies/Routings. In *Proceedings of ACM Intl. Conf. on Supercomputing (ICS).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Exascale computing is the next challenge for the supercomputing community aiming to design systems capable of delivering Exaflops. In order to achieve such a huge computing capability, systems will require millions of interconnected computing elements (CE) to execute massive parallel applications. For this reason new architectures and platforms are being developed, such as our novel, custom-made architecture ABCD [11]. The whole system is composed of tens of millions of low-power-consumption ARM cores to reach Exascale. These nodes are arranged by means of a unified, low-latency, lossless interconnection Network (IN) and a fully distributed storage subsystem with data spread across the nodes. In such system, the IN is crucial to ensure system performance, mainly because it needs to scale to extreme levels of parallelism with applications using tens of thousands of endpoints with any latency or bandwidth bottlenecks translating into severe penalties to execution time. In order to meet the requirements of such interconnect in ABCD we are developing our own general purpose FPGA-based router[12]. One of the requirements of our design is to be simple enough to guarantee low latency while not restricting the variety of network topologies and routing algorithms we are currently exploring. This quest for flexibility imposes on us the challenge of developing low complexity deadlock-avoidance mechanisms able to work with any topology/routing combination. Such mechanisms traditionally lack of generality being specifically designed for a given topology/routing combo, tightly coupled to the routing generation process or are based on algorithms whose complexity precludes them for being used in Exascale-sized networks with millions of endpoints.

In this work we present a collection of three topology- and routing-agnostic deadlock-avoidance mechanism called Dynamic Assignment of Virtual Channels (DAVC). DAVC imposes a negligible overhead in terms of logic as it only needs a few registers to hold local state plus, at most, two comparisons to decide upon transitions between virtual channels (VCs). In contrast to traditional topology-agnostic deadlock-avoidance proposals which require pre-calculation and assignment of paths to VCs, DAVC works on-the-fly, making decisions on each router along the path. Our approach is completely independent from the topology/routing employed and does not require to re-calculate and re-assign VCs upon changes on the architecture, including network failures. The latter is important since Exascale systems are expected to have very low mean time between failures given the sheer number of elements. Hence, extremely-complex recalculations every relatively small period of time may render the IN close to useless. DAVC seamlessly works with arbitrary routing schemes, including minimal, non-minimal and multipath routing algorithms regardless of them being algorithmic, source-routed or table-based. In addition we demonstrate here that the required number of VCs is bounded by the length of the longest path. Given that current technology allows for large-radix, low-diameter topologies – and that the community is following this very same trend [4, 17, 30] – the overheads of our proposal should be relatively small.

First, we prove theoretically that DAVC guarantees deadlock-freedom for any topology/routing combination. We start by formally defining the deadlock routing problem. Then we show that the channel allocation induced by our strategies follows a strict order and thus, it induces and acyclic utilization of channels, which ensures deadlock freedom. Afterwards, we proceed to evaluate the

ICS, 2018, Beijing
2018. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

performance of our approach. Given that our focus is on large-scale interconnects, we rely on simulation to carry out our analysis. We start by assessing DAVC performance against an existing high-performance network-specific algorithm for the Dragonfly topology [17]. This mechanism has versions for both Minimal and Valiant [33] routings. Given DAVC flexibility, we are able to implement other generic routings which are not supported by the standard algorithm – in particular, Shortest Path (SP), Equal Cost Multiple Paths (ECMP) and AllPath (AP). Our second set of experiments uses the Jellyfish topology [30]–a regular random graph (RRG)–for which no efficient deadlock-avoidance mechanism exists [4]. For this reason, and given that typical solutions for irregular networks rely on spanning trees [24, 34] we compare DAVC with a multi-spanning-tree solution similar to the one proposed in [22] in which a configurable number of spanning trees (one per VC) are selected. Results show that DAVC delivers similar performance as the standard deadlock-avoidance mechanism for Dragonflies while allowing the use of other generic routing policies. For RRGs, DAVC avoids deadlock but delivers much higher performance than spanning-tree-based solutions.

In summary, the contributions of our paper are the following:

- We propose a novel, flexible, high-performance, low-overhead deadlock-avoidance mechanisms capable of supporting arbitrary network topologies and routing functions.
- We demonstrate formally that our approach guarantees deadlock freedom.
- We discuss implementation details and highlight the simplicity of its design.
- We evaluate DAVC against a HPC implementation for Dragonfly topologies and find out that it can provide comparable performance levels than topology-specific approaches.
- We extend this evaluation to irregular topologies (Jellyfish) where we compare it with a topology-agnostic spanning-tree based algorithm. DAVC can provide huge benefits in terms of performance and simplicity without all the limitations and overheads of algorithms that rely on global information.

2 BACKGROUND AND MOTIVATION

Deadlock avoidance has been an active research topic since the very beginning of HPC INs. There exist basically two types of routing algorithms to create deadlock-free paths: those that avoid the creation of cycles in the channel dependency graph (CDG) and those that break the cycles in the CDG using VCs. The most prominent examples of the first group are the Spanning Tree protocol, defined in the IEEE 802.1D Standard [1], and Up*/Down* routing [26], the standard in HPC networks such as Infiniband. Indeed, Spanning trees are a specific instance of Up*/Down* routing. Up*/Down* forbids the use of an *up* link after a *down* link has been used. This kind of routing, mainly used in multi-stage networks (i.e. fat-trees), is deadlock-free and can be easily implemented without using VCs. However this approach has many limitations when applied to general topologies: (i) deciding which links are considered 'up' or 'down' is far from trivial, (ii) can leave many resources underutilized (iii) can not ensure minimal-paths, (iv) routes are not balanced efficiently. For this reason other alternatives such as A-2 and MA-2 routing [29], L-turn routing [19] and Multiple Up/Down routing [10] have been proposed to improve the performance of the network by either increasing the proportion of shortest paths or balancing the use of the resources. However, they are still essentially simple variations over the spanning tree concept and so are inherently very restrictive in terms of routing and load-balancing. What is worse, they require some form of topology exploration and embedding global knowledge into the switching logic, which preclude their use for large-scale networks. In fact, experimental work around them is always done with a relatively reduced number of switches (tens of them, at most).

Regarding the second group of algorithms we can also differentiate between those which decouple the creation of paths from the deadlock-free assignment to VCs and those which perform both actions at the same time. DFSSSP [8] and LASH [32] belong to the first group working in a similar way in terms of breaking cycles searching for them in the CDG and moving individual paths to other virtual layers. As both techniques can suffer from a limited number of available virtual layers, LASH was improved in LASH-TOR [31] using Up*/Down* routing in the last VC when unresolvable cycles appear. Finally, the heuristic approach ACRO [16] was proposed to reduce the number of VCs and the time complexity of both LASH and LASH-TOR. On the other hand BSOR [18], Nue [7] and smart routing [5] implement a new approach in which both problems are solved together within the CDG, being able to impose routing restrictions to the path creation on demand (i.e. the use of a fixed number of VCs). However all of them require to perform complex searches onto the CDG being the main drawback of these approaches the computational and memory complexity of the algorithms.

All the above discussed strategies either lack of generality or are excessively complex for our purposes, due to the scale we are aiming at. In addition, none of them is readily available for being integrated in our environment to compare with DAVC, so, given their great complexity, we decided not to re-engineer them for our experimentation purposes.

3 THE DEADLOCK-FREE ROUTING PROBLEM

In this section we define the deadlock-free routing problem for arbitrary network topologies. We start defining terms that will be used thorough the rest of the paper and giving the conditions that guarantee a deadlock-free topology/routing combination.

3.1 Definitions

An IN is composed of a set of nodes (*computing elements* and *switches*) with a number of *ports*. The physical links between nodes are multiplexed into multiple VCs. These connections are defined by a **connection rule** which is the function $\pi : N \times P \rightarrow N \times P$ defined as $\pi(n, p^n) = (n', p^{n'})$ which given a node $n \in N$ and a port $p^n \in P$ within that node returns the node $n' \in N$ and the remote port $p^{n'} \in P$ to which is connected to.

Definition 3.1. An IN is a directed graph $I = G(N, C)$ in which N is the set of nodes and C is the set of channels induced by the connection rule, i.e., given two nodes $n, n' \in N$, the channel $c_{n,n'} \in C \iff \exists p^n \in P : \pi(n, p^n) = (n', p^{n'})$.

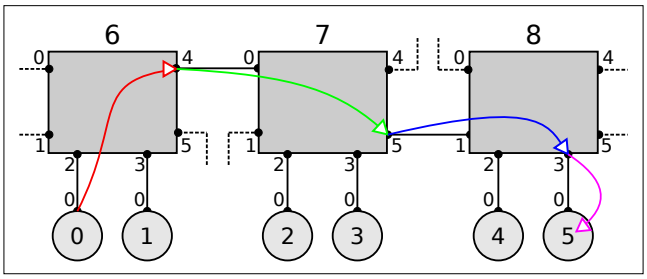


Figure 1: Example of a network topology showing the identifiers of the computing elements (0–5), the switches (6–8) and the ports. With colors, we have also represented the sequence of ports followed by a packet sent from node 0 to node 5.

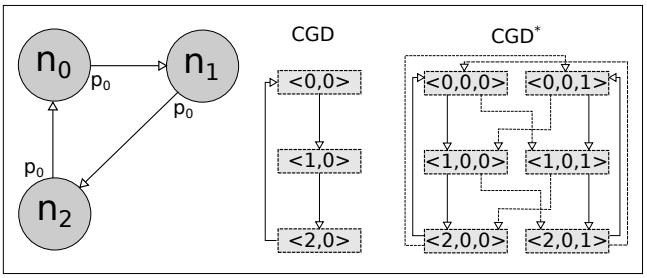


Figure 2: Simple topology (left) and a representation of all channel dependencies (middle) and all channel dependencies considering 2 VCs (right).

As a consequence, a channel between two nodes $n, n' \in N$ is defined as $c_{n,n'} = \langle p^n \rangle = \langle n, p \rangle$ such that $\pi(n, p^n) = (n', p^{n'})$. We define now a path between two nodes as follows:

Definition 3.2. Given a source node n_s and a destination node n_d, a path between n_s and n_d, defined as $P_{n_s,n_d} = (p_0^{n_0}, p_1^{n_1}, \ldots, p_{l-1}^{n_{l-1}}) = (\langle n_0, p_0^{n_0} \rangle, \langle n_1, p_1^{n_1} \rangle, \ldots, \langle n_{l-1}, p_{l-1}^{n_{l-1}} \rangle)$ where $n_i \in N$ and $p_i^{n_i} \in P$, is the sequence of ports within each node n_i that a packet must follow to travel from $n_s = n_0$ to $n_d = n_{l-1}$. The length of the path, l, is defined as the number of hops between n_s and n_d.

In Fig. 1 we have depicted a path between the nodes 0 and 5 which can be represented as $P_{0,5} = (0^0, 4^6, 5^7, 3^8, p_c^5)$ where p_c is the consumption port of the destination node. A generic routing function R assigns the next channel in the path given a destination node and the current channel:

Definition 3.3. An arbitrary routing function $R : N \times C \rightarrow C$ for an IN returns the next channel to be used given the destination node n_d and the current channel c, i.e. $\forall n_d \in N, \exists c' \in C : R(n_d, c) = c'$ which is equivalent to $R(n_d, \langle n, p \rangle) = \langle n', p' \rangle$.

Let us now define the concepts of *inbound port* and *outbound port*.

Definition 3.4. Given a path P and a node $n \in N$ such that $p_i^n \in P$, we call *inbound port* of n to the port $p_{i-1}^{n'}$ and *outbound port* to the port p_i^n.

An example of *inbound port* and *outbound port* is depicted in Fig. 1. If we focus on the third component of the path $p_2 = 5^7$ (green arrow), the *inbound port* would be $p_1 = 4^6 = 4$ and the *outbound port* 5. In the same way we define the concept of *outbound node* as the node id to which an outbound port is connected to. In the previous example the outbound node of the port 5^7 is the node with id 8.

3.2 Deadlock-free Routing

In this work, we consider a routing function to be valid, if and only if the paths induced are deadlock-free. Notice that in Definition 3.3, in opposition to [7], we remove the cycle-free and destination-based conditions from R meaning that we are able to deal with cycles in the paths and with any kind of routing. In [6] the authors give the necessary and sufficient condition for a routing to be deadlock-free (which was reformulated as only a necessary condition in [28]). Next we define the concept of the channel dependency graph (see Fig. 2) used by them:

Definition 3.5. A channel dependency graph $D = G(C, E)$ is a directed graph in which the node set C is composed by the edge set of I and E is the set of edges defined by the routing function R such as $(c_i, c_j) \in E \iff \exists n \in N : R(n, c_i) = c_j$.

Theorem 3.6. *A set of paths within an IN is deadlock-free if and only if there are no cycles in the corresponding channel dependency graph.*

4 DYNAMIC ASSIGNMENT OF VIRTUAL CHANNELS

As mentioned before, generic deadlock avoidance strategies try to break cyces in the CDG. As a result, all of them are applied offline and then populated into the switches of the IN. The way we tackle the problem is a completely different approach in which cycles are broken on-the-fly while the packets are traversing the network. In order to define DAVC we need to redefine the concepts of channels and CDG used in the traditional approaches. We start this section with some preliminary results which, lately, will be used to proof that DAVC is deadlock-free for any topology/routing combination.

4.1 Preliminaries

Let us define the set S_n of all tuples $\langle x_1, x_2, \ldots, x_n \rangle$ such that $\forall i \in \{1, 2, \ldots, n\} : x_i \in \mathbb{N}$ and the relation "$<_n$" where $\langle x_1, x_2, \ldots, x_n \rangle <_n \langle y_1, y_2, \ldots, y_n \rangle \iff \exists i : y_i > x_i \wedge \forall j \in \{i+1, \ldots, n\} : y_i = x_i$.

Lemma 4.1. *The relation "$<_n$" is a strict order on the set S_n of all tuples $\langle x_1, x_2, \ldots, x_n \rangle$.*

Proof. A relation is a strict order [27] if it is irreflexive, asymmetric and transitive. As the demonstration that "$<_n$" fulfils those properties is straightforward we omit the proof. □

Let us consider now an arbitrary graph $D = G(C, E)$ where C is the set S_n of all tuples of length n, and the edge set E is induced by all pairs of nodes $c_i, c_j \in S_n$ related through "$<_n$", such that if $c_i <_n c_j$ then $(e_{c_i,c_j}) \in E$, that is, $D = (S_n, (S_n, <_n))$.

Lemma 4.2. *$D = (S_n, (S_n, <_n))$ is a directed acyclic graph.*

Proof. The proof is straightforward using Lemma 4.1 because every strict order induces a directed acyclic graph, and hence, the graph D is acyclic. □

4.2 DAVC Strategy

Let us consider a graph $I = G(N, C')$ that represents an arbitrary topology in which C' is the set of channels defined as follows:

Definition 4.3. A channel between two nodes $n, n' \in N$ is defined as $c_{n,n'} = \langle p^n, v^p \rangle = \langle n, p, v \rangle$ such that $\pi(n, p^n) = (n', p^{n'})$ and $v \in V$ is the virtual channel within ports p^n and $p^{n'}$.

Notice that Definition 4.3 extends the definition of channel given in Section 3.1 to include the VCs. It also implies that there exist multiple channels between each pair of ports, one per VC. For example, if the number of VCs is m and node n is connected to node n' through port p, there exist m channels between them: $\forall i \in \{1, 2, \ldots, m\} : \langle n, p, v_i \rangle$.

Now we define the function $F : N \times C' \rightarrow C'$ as $F(R(n_d, \langle n, p \rangle), v) = \langle n', p', v' \rangle$ and $c <_3 c'$ where R is a routing function. Looking at F, paths between nodes have the form

$$P_{n_s,n_d} = (\langle n_0, p_0^{n_0}, v_0^{p_0^{n_0}} \rangle, \ldots, \langle n_{l-1}, p_{l-1}^{n_{l-1}}, v_{l-1}^{p_{l-1}^{n_{l-1}}} \rangle)$$

which is an increasing strict ordered sequence of channels. We call F the *allocation function* and it is denoted as F_{NP}. The definition of F_{NP} also implies that the selection of the VCs is independent from the routing function R and that is performed on each router along the path, after the next hop has been calculated. It is also easy to view that the CDG induced by F_{NP} is acyclic which implies that channel transitions generated using F_{NP} are deadlock-free. We denote this CDG as CDG^* because it uses channels using VCs (see right part of Fig. 2).

Theorem 4.4. *The CDG^* $D = (C', E)$ in which C' is the set of channels and E is the set of edges induced by the function F_{NP} is acyclic.*

Proof. It is straightforward to see that $\forall c_i, c_j \in C', (c_i, c_j) \in E \iff F_{NP}(c_i) = c_j \implies c_i <_3 c_j$. This means that the set E is composed of elements of C' which are related through "$<_n$", so $D = (C', (C', <_3))$, by Lemma 4.2, is acyclic. □

By definition of $<_3$, to order two channels we require the identifiers of both current and next channels (node and port ids). However, instead of using both node and port identifiers, we could just use one of them to select a channel ($\langle n, v^{p^n} \rangle$ or $\langle p^n, v^{p^n} \rangle$) and perform the ordering (VC allocation) using $<_2$. These functions are denoted as F_N and F_P.

Theorem 4.5. *The CDG^* $D = (C', E)$ in which C' is the set of channels and E is the set of edges induced by the functions F_N and F_P is acyclic.*

Proof. The proof is the same as in Theorem 4.4 but using $<_2$. □

We conclude this section showing that the allocation functions are able to deal with loop-paths. Even when these kind of paths are not desirable, we guarantee that they will not cause deadlocks. This property will greatly simplify a practical design as will be discussed later on in Subsection 5.4.

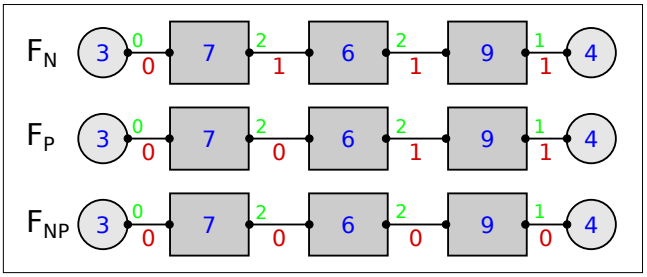


Figure 3: Examples of the VCs allocation using F_N (top), F_P (middle) and F_{NP} (bottom) for a given path. Nodes are represented in blue, ports in green and VCs in red.

Lemma 4.6. *A routing function R that generates paths which contain loops is deadlock-free if channels are allocated using F_{NP}, F_N or F_P.*

Proof. A path P that contains a loop has the form

$$P_{n_s,n_d} = (c_0, \ldots, c_i, \ldots, c_j, \ldots, c_i, \ldots, c_{l-1})$$

in which at least one channel is visited twice, (c_i) in the example. However, we know that by definition of the allocation functions, $c_i <_2 c_i$ or $c_i <_3 c_i$ implies that $\langle n_i, p^{n_i}, v \rangle <_3 \langle n_i, p^{n_i}, v' \rangle$ that is only possible if $v' > v$ by definition of $<_3$. This implies that, even when using the same node and port twice, the path does not create a loop in the CDG^* because the VCs differ. □

In the following sections we analyse DAVC in terms of implementation and hardware requirements. First, we show how these allocation functions can be easily implemented on any router with very low overhead. After this, we perform an analysis of the number of VCs required to implement them.

5 IMPLEMENTATION OF DAVC

The allocation functions which translate the paths generated by the routing function into an ordered sequence of channels using the available VCs can be easily implemented in hardware. As we will see, the overhead added to the routing process is negligible requiring a small amount of logic in each router. In Fig. 3 we have depicted three examples of how channel allocation is performed using F_N, F_P and F_{NP} along the same path.

5.1 Node ID based allocation function

We start with the allocation function F_N which orders the channels along a path based only on node identifiers. The information required to perform the VC transition are just the identifiers of the current and the next node in the path. The later is provided after the routing function has been applied in order to support non-deterministic routing. The pseudocode to implement this function is shown in Alg. 1. As we can see Alg. 1 returns the next VC to be used using the current node identifier (*currentNID*), the current VC (*currentVC*) and the outbound node identifier (*outboundNID*) as defined in Section 3, which is provided by the function *getOutboundNID()*. When the outbound node is lower or equal than the current one we need to perform a VC transition (+1) to maintain the order established by $<_2$. In case the outbound identifier is higher,

the order is already maintained and, hence, the VC transition is not required. Let us illustrate how this allocation function works with an example.

Algorithm 1 Node ID based VC assignment

```
1: procedure NEXT_VC(currentNID, currentVC)
2:     outboundNID ← getOutboundNID()
3:     if outboundNID ≤ currentNID then
4:         currentVC ← currentVC + 1
5:     end if
6:     return currentVC
```

Given the path shown in Fig. 3 (top) in which a packet travels from node 3 to node 4, the routing function R that returns the next channel to be used and the allocation function F_N, we start applying $R(4, \langle 3, - \rangle) = \langle 7, 0 \rangle$. Notice that source nodes are connected to the switches through port 0. Then we apply F_N which returns $\langle 7, 0^3, 0 \rangle$ because injection is always performed in the VC 0. After the packet is moved to node 7 using port 0 and VC 0, the process is repeated: $F_N(R(4, \langle 7, 0 \rangle), 0) = \langle 6, 2^7, 1 \rangle$. Now, as the packet travels from node 7 to 6 (lower identifier) the allocation function increases the VC to be used. The next channel to be used is calculated as $F_N(R(4, \langle 6, 2 \rangle), 1) = \langle 9, 2^6, 1 \rangle$; here, the next identifier is higher, so a VC transition is not required. Finally, the hop from node 9 is the destination, so a VC transition is not required.

5.2 Port ID based allocation function

The second allocation function F_P orders the channels along a path using only the port identifiers. In this case it only requires the inbound and outbound ports (as defined in Section 3), both provided by the routing function. The pseudocode to implement this function is shown in Alg. 2.

Algorithm 2 Port ID based VC assignment

```
1: procedure NEXT_VC(inboundPID, currentVC)
2:     outboundPID ← getOutboundPID()
3:     if outboundPID ≤ inboundPID then
4:         currentVC ← currentVC + 1
5:     end if
6:     return currentVC
```

F_P works in a similar way as F_N so the example provided is similar (see Fig. 3 (middle)). The only difference is the sequence of VCs used in the path, which are allocated considering the ports identifiers. Let us take a look at the routing and allocations steps:

(1) $F_P(R(4, \langle 3, - \rangle), 0) = \langle 7, 0^3, 0 \rangle$
(2) $F_P(R(4, \langle 7, 0 \rangle), 0) = \langle 6, 2^7, 0 \rangle$
(3) $F_P(R(4, \langle 6, 2 \rangle), 0) = \langle 9, 2^6, 1 \rangle$

In the first switch (step 2), the VC is kept because the packet travels from port 0 to 2 (higher identifier). However, in step 3, the hop is performed from port 2 to port 2 (same id) and in consequence a VC transition is enforced.

5.3 Node-Port ID based allocation function

We finish this section with the allocation function F_{NP} which uses both the node and port ids. As we can see in Fig. 3 (bottom), using more information can help to use less VCs. The pseudocode to implement this function is shown in Alg. 3.

Algorithm 3 Node-Port ID based VC assignment

```
1: procedure NEXT_VC(currentNID, inboundPID, currentVC)
2:     outboundNID ← getOutboundNID()
3:     outboundPID ← getOutboundPID()
4:     if outbound_PID < inboundPID or
         (outboundPID = inboundPID and
         outboundNID ≤ currentNID)      then
5:         currentVC ← currentVC + 1
6:     end if
7:     return currentVC
```

F_{NP} works as a combination of F_N and F_P. Let us take a look at the routing and allocations steps shown in Fig. 3:

(1) $F_{NP}(R(4, \langle 3, - \rangle), 0) = \langle 7, 0^3, 0 \rangle$
(2) $F_{NP}(R(4, \langle 7, 0 \rangle), 0) = \langle 6, 2^7, 0 \rangle$
(3) $F_{NP}(R(4, \langle 6, 2 \rangle), 0) = \langle 9, 2^6, 0 \rangle$

In this case, the condition to perform a VC transition is much stricter (combines together the conditions from F_N and F_P), and hence, we expect less VC transitions. In the example, the conditions are never fulfilled, so only one VC (0) is used along the path.

5.4 Considerations for the practical design

We will move now to discuss about how to translate these simple algorithms into a practical switch design. In contrast with the deadlock alternatives discussed in Section 2, which require distributing channel transitions all across the system, our proposals only require very little information about the local and neighbouring nodes: ids for the nodes and ports. The logic to store this information is very minimal, just a small register per port–to store its id plus the id of the neighbouring node and port–plus another, even smaller, register per node to store the local id. Obviously *outboundPID*, *inboundPID*, *currentNID* and *outboundNID* would read the information from these registers, so the implementation of their respective functions will be straightforward.

Special attention requires how these registers would be loaded with their respective information. Given that this information seldom varies–local ids can be fixed and neighbouring info only varies when cables are unplugged or broken–we can consider it static for the purposes of this discussion. Ports within a switch are implicitly numbered from 0 to $r - 1$ for the different functions within a switch (e.g., switch allocation), where r is the radix of the switch or, in other words, the number of ports. This numbering will serve very well for our purposes and can be safely hard-coded into each port with a few bits, e.g. 10 bits would support switches with up to 1024 ports. Alternatively, they could be assigned dynamically by the switch logic as ports are brought up (i.e. connected to another switch). However, the extra hardware to deal with port-id allocation and consistency seems like an excessive overhead which serves no real purpose, so dynamic port-id allocation would be discouraged for a power-efficient design such as ours.

Regarding node ids, up until now our examples have assumed that endpoints and switches are numbered consecutively and that all ids are unique. This is done for the sake of simplicity,but is not a requirement of our algorithms as they only rely on the existence of a strict order operation. For this reason, while it is possible (although far from trivial) to implement a system-level function that ensures node ids are consecutive and unique, such a system-level facility is not required. Hence, we can rely on a simpler methodology for generating node ids. The simpler, and aligned with current practice for many networking equipment would be to have a hard-coded physical address incorporated into the switch at fabrication time. However, in a FPGA-based set-up as the one we are aiming at providing, hard-coding the id would require generating a separated firmware for each FPGA. For this reason, a small module that reads several local sensors (e.g. voltage, temperature, internal clock, etc) and hashes them together to generate a random id at boot up seems like a more flexible solution.

Finally, now that we have established how local id information can be generated, we look into how this information can be distributed. The simplest, most effective way of sharing information with the neighbours is to implement it in a per-port basis during the negotiation/handshaking process that occurs when the ports are brought up once a cabled is plugged in. Instrumenting the ports to interchange node and port ids and store them in the local register as one of the last steps of the interface up process should be trivial and require very little extra logic.

All these considerations show the feasibility of our approach and also that it imposes very low overhead to the switch architecture and no system-level support.

6 ANALYSIS OF VC REQUIREMENTS

As we have seen, the allocation functions work in a distributed and online manner making decisions independently on each router. The overhead of implementing them on each router is negligible just requiring a few extra logic to perform one or two comparisons over information which is readily available (current and next channel). Furthermore, no inter-router communication is required, nor any off-line process to generate the dependency graphs. The main overhead is the number of VCs. Most current interconnection technologies support a relatively large number of VCs; 8 and 16 VCs are not uncommon. However, surpassing that number would pose a great limitation for the use of DAVC. Hence, we will look now at the requirements in terms of VCs of each proposed allocation function when dealing with different topology/routing combos.

6.1 Analysis of algorithms

The number of VCs required depends on the type of routing used. Clearly, when minimal routing is used this number is bounded by the diameter of the network because the number of VC transitions is determined on each hop of the path. Similarly, when non-minimal routing is used, the number of VCs is bounded by the longest path length which can be longer than the diameter of the network. Notice that the allocation functions are independent of the type of routing and can be used with single- or multi-path routing strategies. The following lemma formalises the maximum number of VCs required (the proof is straightforward):

Lemma 6.1. *Given a routing function R and any VC allocation function F, the maximum number of VCs used is lower than the number of hops of the longest path.*

Notice that in practice, as the injection is performed in VC 0 and consumption does not require a VC transition, the maximum number of VCs equals to the length of the longest path minus 2. The worst case happens when one of the longest paths requires a VC transition in every hop, which depends on the allocation function used. Let us analyse the worst case in terms of VC transitions on each of the allocation functions for the longest path P reported by a routing function:

- F_N: As the VC assignment on this function depends on the ids of the nodes along the path, the worst case appears when the sequence of node ids is decreasing.
- F_P: In this case the VC assignment is performed according to the sequence of port ids along the path. The worst case appears when the sequence of port ids is decreasing.
- F_{NP}: This function uses a combination of node and port ids to perform a VC transition which only happens when the sequence of nodes and ports ids is decreasing. This imposes a more strict condition to perform a VC transition which can result in the use of less number of VCs (as happened in the example shown in Fig. 3.

Although DAVC is independent from the topology/routing, the number of required VCs is not. However, it is easy to determine this number just analysing the topology and the routing implemented.

6.2 Analysis of DAVC on different topology/routing combinations

Now we analise the most used topology/routing combinations to determine the number of VCs required to implement DAVC. As our proposal is bounded by the length of the longest path, the most appropriate topologies to be uses with DAVC are those which have short path lengths. We will discuss the feasibility of DAVC for two topologies, Dragonfly and Jellyfish [30] which belongs to the Regular Random Graphs (RRG) family.

6.2.1 Torus topologies. These topologies have been widely used in the past to build supercomputers. However, they are not the most suitable for DAVC because of their large diameter which would impose a large number of VCs. Furthermore, torus topologies already have efficient ways to avoid deadlock, such as the bubble-router algorithm [25] which only requires one VC.

6.2.2 Fat-tree topology. Regarding fat-trees and other multi-stage topologies, DAVC would be suitable because the diameter grows logarithmically with the number of endpoints. For normal application traffic, Up*/Down* routing is very well suited for these topologies and requires of a single VC to avoid deadlock. However, the management traffic for InfiniBand-based systems requires of special consideration for tree-like networks. In this case, communication between switches are required which means that Up*/Down* routing can not always be applied (e.g., communications between root switches) [2, 3]. In practice this means that a separate deadlock-avoidance for management traffic is needed, a role for which DAVC can be an appropriate and elegant solution.

6.2.3 Dragonfly topology. Dragonfly [17] is a direct topology whose main design objectives was to minimize the diameter, which is the best scenario to use DAVC. Using minimal-routing the longest path is 5 hops. However, minimal routing is known to not balance well the traffic in Dragonflies. For this reason Valiant routing [25] is typically used to maximize network utilization. This routing is randomized and non minimal and has a longest path of 7 hops. These paths lengths would keep the number of VCs needed for DAVC relatively low (3 and 5, respectively). The availability of short paths together with the existence of a specific HPC deadlock-avoidance mechanism [17] we can compare with motivate this topology being included in our experimental work.

6.2.4 RRG topologies. The diameter of the RRGs depends on the radix of the switches. Using realistic switches with up to 64 ports, we can build very large topologies with thousands of computing elements with very low diameter. This fact together with the absence of practical deadlock-avoidance mechanisms [4] makes these kind of networks the main target for DAVC.

6.2.5 Other topologies. Finally, there is a plethora of topologies proposed for the data-centre domain for which there exists no deadlock avoidance mechanisms. They use Ethernet interconnects and rely on TCP-based packet dropping and retransmission. Some examples worth mentioning are DCell [14], BCube [15], FiConn [20] and DPillar [21]. All of these topologies feature low diameter and could benefit greatly from an advanced deadlock-avoidance mechanisms such as DAVC maybe even to the point where they could be used within HPC domains. At any rate, looking into data-centre topologies is outside of the scope of this paper so we will stick to HPC topologies for the remaining of this paper.

7 EXPERIMENTAL SET-UP

In this section we present the simulation environment used to analyse the performance of DAVC. First we describe the experimental environment which is composed of the INSEE simulator [23] and the different traffic generation models. We conclude the section describing the set of experiments performed.

7.1 Simulation environment

The evaluation has been carried out using INSEE, a widely use and tested interconnection networks simulator. INSEE uses a very detailed model of the router at a phit level and supports both the generation of several synthetic traffic patterns and the execution of traces extracted from real applications. In addition, it also implements several deadlock-avoidance mechanisms for many different topology/routing combinations.

We carried out two set of experiments with two topologies of the same size, a Dragonfly(6,12,6) and a RRG(876,23,17), both composed of 876 23-port switches (6 ports for compute nodes plus 17 ports for inter-switch) and connecting a total of 5256 computing nodes. In each set of experiments we first focus on raw performance metrics such as throughput and VC utilization and use two synthetic traffic patterns:

- Uniform: This is the traditional pattern in which packet destinations are selected uniformly at random. This generates a uniform distribution of traffic and serves to understand the load-balancing capability of the different schemes.
- Adversarial: This is a completely unbalanced traffic pattern which was designed for Dragonfly. It generates a pathological distribution of traffic in which all the nodes in a group try to use the same output port in the group by sending traffic randomly to the nodes in the next group. For RRGs, we have adapted this traffic pattern such that the network is divided in groups of a given size and each group sends the traffic to the next group.

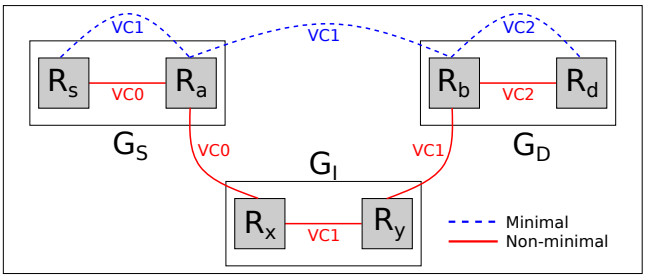


Figure 4: VC assignment to avoid deadlock in Dragonfly topologies for both minimal and non-minimal routing.

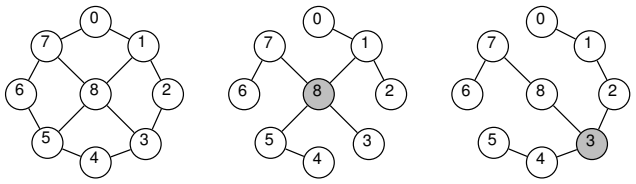
The second set of experiments is similar but using traffic patterns typically present in HPC applications:

- Stencil 2D and 3D: These patterns resemble typical communication patterns in HPC applications with large matrices. The application tasks are arranged in a 2D or 3D lattice and compute for a certain part of the matrix and only need communicating with its neighbours in this virtual topology.
- Butterfly: This pattern represents an optimised, binary implementation of collective operations in which each node communicates with other nodes located at power of 2 distances.
- Waterfall: This traffic pattern consists of a large collection of small messages, with very tight causal dependencies. We can consider this pattern heavy because during the most of the execution time, most of the nodes are injecting messages at once.

In order to compare the performance of DAVC we have used one deadlock-avoidance mechanism for each topology. For Dragonflies we have implemented the one used in the original work [17] which we refer as *Dally* in this work. This mechanism uses two VCs for minimal routing and three VCs for Valiant routing as depicted in Fig. 4. For RRGs we have used SPDA which creates a spanning-tree per VC to spread the traffic over them (see below).

7.2 Spanning-tree protocol deadlock-avoidance (SPDA)

SPDA is a simple generic strategy that, given a network topology and a number m (configurable) of VCs to be used, calculates m spanning trees which will be used to route the traffic. The creation of the spanning trees is performed by selecting a root node at random and using a breadth first search (BFS) to create a minimal spanning tree from it. When a packet is injected, one of the m VCs is selected



(a) Example topology (b) ST0 – VC0 (c) ST1 – VC1

Figure 5: Example of two spanning trees (b) and (c) calculated from a non deadlock-free network topology (a). Gray nodes represent the root of the spanning tree.

randomly and the packet will follow the path determined by the corresponding spanning tree. Deadlock-free routing is guaranteed as spanning-trees have no cycles. An example of SPDA is depicted in Fig. 5 which shows two spanning trees of a network topology which are assigned to VC 0 and 1 respectively. Notice that SPDA is also a routing policy as the packets are routed to their destination following the selected spanning-tree.

7.3 Generic routing policies

We have used Minimal and Valiant routing policies for Dragonfly and SPDA for RRGs. However, in order to test the capacity of DAVC to use any kind of routing we have also used the following three generic routings for both topologies:

- Shortest Path (SP): Uses only one of the existing shortest paths between each source and destination. The longest path is equal to the diameter of the topology
- Equal cost Multiple Paths (ECMP): This is a multi-path policy which, instead of using just one of the shortest paths between each source and destination, spreads the traffic between all of them. The length of the longest path is again the diameter.
- ALLPATH(k) (AP) [9]: This policy is similar to ECMP but uses all existing paths whose length is no longer than the shortest path plus k. Notice that these longer paths may not exist for a particular topology.

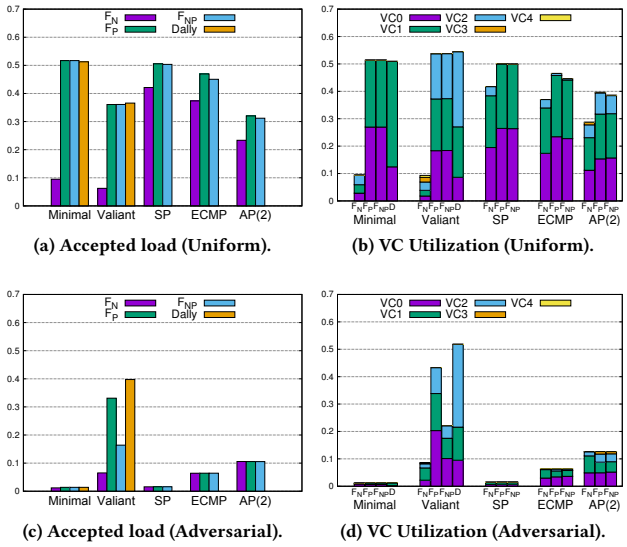
8 EXPERIMENTAL RESULTS

We move now to analyse the results of our experimental work.

8.1 Dragonfly topology

First, we will focus on the results for the Dragonfly topology. As mentioned above, the main objective of this set of experiments is to assess the performance of DAVC vs a HPC deadlock-avoidance mechanism, and not the performance of the routing policies.

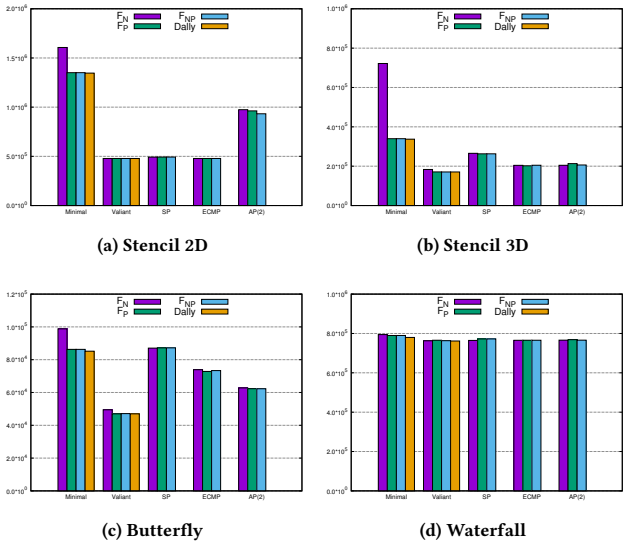
Fig. 6 shows the throughput of the dragonfly topology (at maximum injection load) and the VC occupancy. Let us start analysing the performance using uniform traffic. There are several points to highlight here. First of all, we can see that F_P and F_{NP} are able to operate at a same throughput level as the efficient Dally implementation for both minimal and Valiant. Also they are capable to support other routing functions (SP, ECMP and ALLPATH), all of which are able to outperform Dally+Valiant for this specific case. While the objective of this work is not to compare routing functions,



(a) Accepted load (Uniform). (b) VC Utilization (Uniform).

(c) Accepted load (Adversarial). (d) VC Utilization (Adversarial).

Figure 6: Accepted load and VC utilization of a DF(6,12,6) using uniform and adversarial traffic.



(a) Stencil 2D (b) Stencil 3D

(c) Butterfly (d) Waterfall

Figure 7: Execution time (in cycles) for realistic workloads in a DF(6,12,6).

it is clear that being able to support more routing functions can be beneficial. The case of F_N requires special consideration as it produces a pathological scenario for this topology when combined with the standard routing functions (Min and Valiant), but performs relatively well with the other routings. The culprit for its low performance is the network becoming highly congested, which could be alleviated by implementing a congestion management algorithm, e.g., [13]. A detailed examination showed that with other Dragonfly

link-arrangements, the performance of F_N with Min and Valiant is much closer to that of F_P and F_{NP}, but still substantially lower.

Moving on to adversarial traffic, it is more difficult to draw conclusions because most of the networks are suffering from exceedingly high congestion scenarios. However, it is clear that, in this case, multipath non-minimal routes (Valiant, ECMP and AP) are able to reduce significantly the effects of congestion.

Only Valiant+Dally, and to some extent, Valiant+F_P are able to reach acceptable performance levels, though. While this is expected as ADV was designed to highlight the Valiant+Dally combo, it is still interesting to see that F_P is capable of getting competitive performance values as well. Moving from Valiant, now, the differences between F_N and F_P and F_{NP} are diluted and the three of them seem to perform roughly the same for all routing functions.

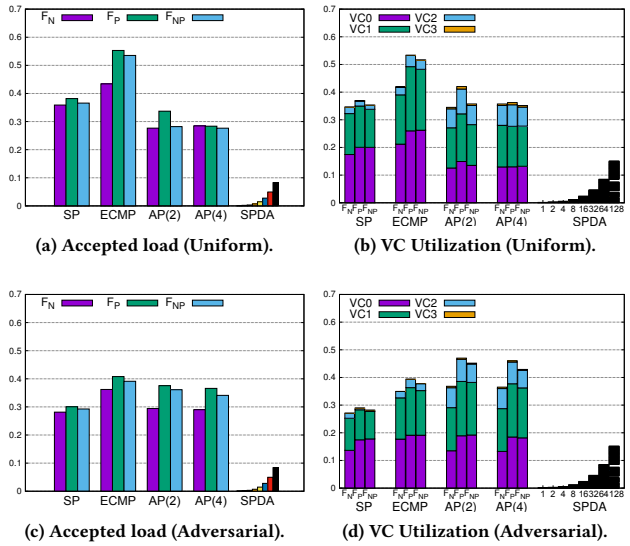
Also of interest is the VC utilization of the different algorithms. Both F_P and F_{NP} require the same number of VCs as Dally for both Min and Valiant, regardless of the traffic pattern. We can see that this is not the case for F_N which always requires a higher number of VCs.

If we focus on the realistic workloads, shown in Fig. 7 we can see that when dealing with application traffic the differences between Dally and our proposals is very small, except for Min+F_N as expected from the throughput figures shown above. Furthermore, the non-standard routing algorithms can outperform either Min or Valiant in all scenarios, which again, highlights the benefits of providing higher flexibility.

8.2 RRG topology

Let us analyse now the results for the RRG topology. Looking at the results with synthetic traffic in Fig. 8, we can see that in no case the network seems to become highly congested as happened with the Dragonfly. This is because of the high path diversity of the RRG topologies. In this case, ECMP seems to be the best routing solution as it can sustain the highest throughputs for both traffic patterns. The rationale for that is that ECMP leverages the gains of using shortest paths for balanced traffic (uniform), with those of using multipath for unbalanced traffic (adversarial). Note that SP out performs ALLPATH for Uniform, whereas with Adversarial the results are just the opposite. Focusing on the different DAVC functions, we can see that F_P seems to be able to offer the best performance whereas, again, F_N seems to suffer from saturation more than the others, regardless of the routing function. At any rate, we can see how these algorithms hugely outperform the spanning tree deadlock avoidance, due to the strict restrictions imposed by such algorithms, which both leave many channels unused and make paths longer. We can see how allowing more VCs for more parallel spanning trees has a major influence in the performance, but even with hundreds of VCs, the performance is not near comparable to that of DAVC.

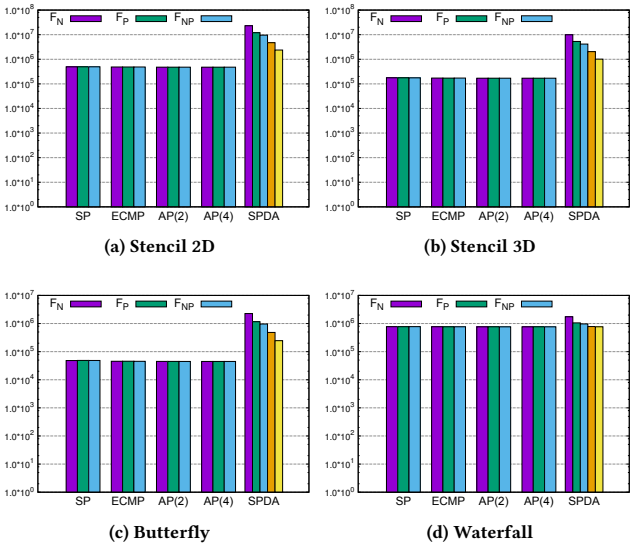
These trends translate neatly to the application-like traffic (see Fig. 9) where we can see that the differences between DAVC implementations are subtle, whereas the SPDA takes up to 2 orders of magnitude longer when a small number of VCs are used. Although the purpose of the paper is not to compare the topologies, we can see how the running times of the different applications over the RRGs and Dragonflies are similar to each other, which shows that



(a) Accepted load (Uniform). (b) VC Utilization (Uniform).

(c) Accepted load (Adversarial). (d) VC Utilization (Adversarial).

Figure 8: Accepted load and VC utilization of a RRG(876,23,17) using uniform and adversarial traffic.



(a) Stencil 2D (b) Stencil 3D

(c) Butterfly (d) Waterfall

Figure 9: Execution time (in cycles) for realistic workloads in a RRG(876,23,17). Notice the logarithmic scale of the y axis.

the performance with the former is comparable to that of the Dragonfly with Dally, which further demonstrates the good performance of our approach.

We conclude our discussion of results with some final remarks about the different allocation functions for DAVC. In general, F_N seems to be the worst alternative, since it is the one that requires the most VCs and also the one that seems to be more prone to congestion. The difference between F_P and F_{NP} seems to be, in general, much smaller with the only exception of DF-Valiant we

mentioned above. In the rest of the cases, F_P seems to be slightly better in terms of performance in many cases, but the difference may very well be compensated by the, *a priori*, lower number of VCs required by F_{NP}.

9 CONCLUSIONS AND FUTURE WORK

In this paper we have presented DAVC, a new mechanism to avoid deadlock in arbitrary topology/routing combinations. DAVC is an online, low-overhead mechanism that must be implemented on each router.

After theoretically proving that DAVC guarantees deadlock-avoidance, we evaluated the performance against specific mechanisms designed for two specific topologies. In addition we also evaluated the performance of generic routing policies as alternatives to specific ones.

Results show that DAVC achieves similar performance of the deadlock-avoidance mechanism used in Dragonfly while at the same time allowing the use of other generic routing policies. In the case of RRGs, DAVC allows the use of any routing in these random topologies while delivering much higher performance than a specific deadlock-avoidance mechanism using spanning-trees.

The results presented in this paper show the raw performance of DAVC without any optimizations. However there are still many improvements to be done in order to further reduce the number of VCs used. Two of the ideas that we are investigating for future works are: (i) Analysis of the generated paths so to be able to instrument routing functions in such a way that we reduce the number of VC transitions. (ii) Strategies for reordering the node and port ids so to reduce the number of VC transitions.

ACKNOWLEDGMENTS

Anonymised for Blind Review

REFERENCES

[1] IEEE Standard Association. 2012. IEEE 802.1D Standard. (2012).

[2] Bartosz Bogdanski. 2013. System and method for providing deadlock free routing between switches in a fat-tree topology. (May 9 2013). http://www.google.com/patents/US20130114620 US Patent App. 13/653,303.

[3] Bartosz Bogdanski, Sven-Arne Reinemo, Frank Olaf Sem-Jacobsen, and Ernst Gunnar Gran. 2012. sFtree: A Fully Connected and Deadlock-free Switch-to-switch Routing Algorithm for Fat-trees. *ACM Trans. Archit. Code Optim.* 8, 4, Article 55 (Jan. 2012), 20 pages.

[4] Cristóbal Camarero, Carmen Martínez, and Ramón Beivide. 2017. Random Folded Clos Topologies for Datacenter Networks. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 193–204.

[5] Ludmila Cherkasova, Vadim Kotov, and Tomas Rokicki. 1996. Fibre channel fabrics: evaluation and design. In *Proceedings of HICSS-29: 29th Hawaii International Conference on System Sciences*, Vol. 1. 53–62 vol.1.

[6] William J. Dally and Charles L. Seitz. 1987. Deadlock-Free Message Routing in Multiprocessor Interconnection Networks. *IEEE Trans. Comput.* 36, 5 (May 1987), 547–553.

[7] Jens Domke, Torsten Hoefler, and Satoshi Matsuoka. 2016. Routing on the Dependency Graph: A New Approach to Deadlock-Free High-Performance Routing. In *Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing (HPDC '16)*. ACM, New York, NY, USA, 3–14.

[8] Jens Domke, Torsten Hoefler, and Wolfgang E. Nagel. 2011. Deadlock-Free Oblivious Routing for Arbitrary Topologies. In *2011 IEEE International Parallel Distributed Processing Symposium*. 616–627.

[9] Peyman Faizian, Md Atiqul Mollah, Xin Yuan, Zaid Alzaid, Scott Pakin, and Michael Lang. 2018. Random Regular Graph and Generalized De Bruijn Graph with k -Shortest Path Routing. *IEEE Transactions on Parallel and Distributed Systems* 29, 1 (Jan. 2018), 144–155.

[10] Jose Flich, Pedro López, José C. Sancho, Antonio Robles, and Jose Duato. 2002. *Improving InfiniBand Routing through Multiple Virtual Networks*. Springer Berlin Heidelberg, Berlin, Heidelberg, 49–63.

[11] Anonymized for Blind Review. 2017.

[12] Anonymized for Blind Review. 2018.

[13] Marina García, Enrique Vallejo, Ramón Beivide, Mateo Valero, and German Rodríguez. 2013. OFAR-CM: Efficient Dragonfly Networks with Simple Congestion Management. In *2013 IEEE 21st Annual Symposium on High-Performance Interconnects*.

[14] Chuanxiong Guo, Wu Haitao, Kun Tan, Lei Shi, Guohan Lu, Yongguang Zhang, and Songwu Lu. [n. d.]. ([n. d.]).

[15] Chuanxiong Guo, Guohan Lu, Dan Li, Haitao Wu, Xuan Zhang, Yunfeng Shi, Chen Tian, Yongguang Zhang, and Songwu Lu. 2009. BCube: A high performance, server-centric network architecture for modular data centers. *SIGCOMM Comp. Comm. Review* 39, 4 (October 2009), 63–74.

[16] Ryuta KAWANO, Hiroshi NAKAHARA, Seiichi TADE, Ikki FUJIWARA, Hiroki MATSUTANI, Michihiro KOIBUCHI, and Hideharu AMANO. 2017. A novel channel assignment method to ensure deadlock-freedom for deterministic routing. *IEICE Transactions on Information and Systems* E100D, 8 (8 2017), 1798–1806.

[17] John Kim, William J. Dally, Steve Scott, and Dennis Abts. 2008. Technology-Driven, Highly-Scalable Dragonfly Topology. In *Procs. of the 35th Annual Intl. Symposium on Computer Architecture (ISCA '08)*. IEEE Computer Society, Washington, DC, USA, 77–88.

[18] Michel A. Kinsy, Myong Hyon Cho, Tina Wen, Edward Suh, Marten van Dijk, and Srinivas Devadas. 2009. Application-aware Deadlock-free Oblivious Routing. In *Procs. of the 36th Intl. Symp. on Computer Architecture (ISCA '09)*. ACM, New York, NY, USA, 208–219.

[19] M. Koibuchi, A. Funahashi, A. Jouraku, and H. Amano. 2001. *L-turn routing: An adaptive routing in irregular networks*. Vol. 2001-January. Institute of Electrical and Electronics Engineers Inc., 383–392.

[20] Dan Li, Chuanxiong Guo, Haitao Wu, Kun Tan, Yongguang Zhang, Songwu Lu, and Jianping Wu. 2011. Scalable and Cost-Effective Interconnection of Data-Center Servers Using Dual Server Ports. *IEEE/ACM Transactions on Networking* 19, 1 (February 2011), 102–114.

[21] Yong Liao, Jiangtao Yin, Dong Yin, and Lixin Gao. 2012. DPillar: Dual-port server interconnection network for large scale data centers. *Comp. Networks* 56, 8 (May 2012), 2132–2147.

[22] Olav Lysne and Tor Skeie. 2001. Load Balancing of Irregular System Area Networks through Multiple Roots *(2nd International Conference on Communications in Computing (CIC'01))*.

[23] Javier Navaridas, José Miguel-Alonso, Jose A. Pascual, and Francisco J. Ridruejo. 2011. Simulating and evaluating interconnection networks with INSEE. *Simulation Modelling Practice and Theory* 19, 1 (2011), 494 – 515.

[24] Radia Perlman. 2000. *Interconnections (2nd Ed.): Bridges, Routers, Switches, and Internetworking Protocols*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[25] Valentín Puente, Cruz Izu, Ramón Beivide, Jose A. Gregorio, Fernando Vallejo, and Jose M. Prellezo. 2001. The Adaptive Bubble Router. *J. Parallel and Distrib. Comput.* 61, 9 (2001), 1180 – 1208.

[26] Thomas L. Rodeheffer, Chuck Thacker, Andrew Birrell, Tom Rodeheffer, Hal Murray, Michael Schroeder, Ed Satterthwaite, Roger Needham, Mike Burrows, M. D. Schroeder, and Mike Schroeder. 2006. Autonet: A High-speed, Self-configuring Local Area Network Using Point-to-point Links. *IEEE Journal on Selecetd Areas in Communications* 9, 8 (Sept. 2006), 1318–1335.

[27] Alex Sakharov. 2018. Strict Order. From MathWorld—A Wolfram Web Resource created by Eric W. Weisstein. (2018). http://mathworld.wolfram.com/StrictOrder.html Last visited on 29/1/2018.

[28] Loren Schwiebert. 2001. Deadlock-Free Oblivious Wormhole Routing with Cyclic Dependencies. *IEEE Trans. Comput.* 50, 9 (Sept. 2001), 865–876.

[29] Federico Silla and Jose Duato. 2000. High-Performance Routing in Networks of Workstations with Irregular Topology. *IEEE Transactions on Parallel Distributed Systems* 11, 7 (July 2000), 699–719.

[30] Ankit Singla, Chi-Yao Hong, Lucian Popa, and P. Brighten Godfrey. 2012. Jellyfish: Networking Data Centers Randomly. In *Procs. of the 9th USENIX Conf. on Networked Systems Design and Implementation (NSDI'12)*. Berkeley, CA, USA, 17–17.

[31] Tor Skeie, Olav Lysne, Jose Flich, Pedro Lopez, Antonio Robles, and Jose Duato. 2004. LASH-TOR: A Generic Transition-Oriented Routing Algorithm. In *Procs. of the Parallel and Distributed Systems, 10th Intl. Conf. (ICPADS '04)*.

[32] Tor Skeie, Olav Lysne, and Ingebjorg Theiss. 2002. Layered Shortest Path (LASH) Routing in Irregular System Area Networks. In *Procs. of the 16th Intl. Parallel and Distributed Processing Symp. (IPDPS '02)*.

[33] Leslie G. Valiant. 1982. A Scheme for Fast Parallel Communication. *SIAM J. Comput.* 11, 2 (1982), 350–361.

[34] Dong Xiang and Jiangxue Han. 2012. Multiple spanning tree construction for deadlock-free adaptive routing in irregular networks. In *IEEE 10th Intl. Symp. on Parallel and Distributed Processing with Applications*. 9–16.