J Supercomput (2016) 72:4497–4519
DOI 10.1007/s11227-016-1756-1

Straightforward solutions to reduce HoL blocking in different Dragonfly fully-connected interconnection patterns

Pedro Yébenes[1] · **Jesus Escudero-Sahuquillo**[1] · **Pedro J. García**[1] · **Francisco J. Quiles**[1]

Published online: 27 May 2016

Abstract The performance of interconnection networks is a challenging issue for High-Performance Computing (HPC) systems, which becomes even more important when the number of interconnected endnodes grows. In that sense, Dragonfly interconnection patterns are a very popular option to configure the network topology, especially for large systems, as they are able to achieve a high scalability relying on high-radix switches. This kind of hierarchical topologies has two levels of interconnection (i.e., connections within the element of a group and connections among groups) and each one can be interconnected using different patterns. However, regardless of the Dragonfly interconnection pattern, the Head-of-Line (HoL) blocking effect derived from

This work has been jointly supported by the Spanish MINECO and European Commission (FEDER funds) under the projects TIN2012-38341-C04 and TIN2015-66972-C5-2-R, and the FPI grant BES-2013-063681, and by Junta de Comunidades de Castilla- La Mancha under the project PEII-2014-028-P. Jesus Escudero-Sahuquillo has been funded by the Spanish MINECO under the postdoctoral grant FPDI-2013-18787 until November 2015 and, from that date, he has been funded by the University of Castilla-La Mancha (UCLM) and the European Commission (FSE funds), with a contract for accessing the Spanish System of Science, Technology and Innovation, for the implementation of the UCLM research program (UCLM resolution date: 31/07/2014).

✉ Pedro Yébenes
pedro.yebenes@uclm.es; pedroyebenes@gmail.com

Jesus Escudero-Sahuquillo
Jesus.Escudero@uclm.es

Pedro J. García
pedrojavier.garcia@uclm.es

Francisco J. Quiles
francisco.quiles@uclm.es

[1] Department of Computing Sytems, University of Castilla-La Mancha, Instituto de Investigación en Informática de Albacete (I3A), Campus Universitario s/n, 02071 Albacete, Spain

congestion situations may jeopardize the Dragonfly performance. This paper analyzes the dynamics of congestion in different Dragonfly fully-connected interconnection patterns. Also, we describe a queuing scheme called *Hierarchical Two-Level Queuing* (H2LQ), designed specially to reduce HoL blocking in any fully-connected Dragonfly network that uses minimal-path routing. Finally, we present experiment results which show that this scheme significantly boost Dragonfly performance, regardless the interconnection pattern, especially when congestion arises, while requiring fewer network resources than other techniques oriented to deal with the effects of congestion.

Keywords High performance computing · Interconnection networks · Dragonfly topology · HoL blocking · Congestion control

1 Motivation

High-Performance Computing (HPC) systems are widely used nowadays to support the growing computational demands of applications used in many and diverse fields, like Big-Data, parallel computing, GPU computing, cloud computing, distributed file-systems, databases, social networks, etc. These systems need an interconnection network able to meet their high communication requirements, offering high-bisection bandwidth and low latency. Otherwise, the network may be the system bottleneck and the whole system may not behave as expected. One of the most important factors in the design of a network is its topology, as many other network aspects depend on it. In that sense, Dragonfly topologies [22] are becoming popular as an alternative to traditional direct and indirect topologies. Specifically, the Dragonfly topology offers a high scalability at a very good cost/performance ratio with the current hardware. Indeed, it is able to maintain network performance as the network size grows, at the cost of requiring switches with a higher radix. Besides, it offers a very low diameter, path diversity, and a high bisection bandwidth. Currently, several supercomputer manufacturers use this topology in their systems such us the Cray XC series [3] or the IBM PERCS [6].

The original Dragonfly topology definition is very loose because several aspects are not specified. For instance, this topology is defined as a hierarchical structure with two levels of interconnection (inter- and intragroup), but the specific inter-and intragroup patterns are not restricted. In fact, although fully-connected patterns are suggested, the decision remains in the network designer. Several authors have analyzed thoroughly the Dragonfly topological properties and proposed several fully-connected interconnection patterns [8,18], i.e., different ways of fully interconnecting the network components. The analysis of these configuration patterns is an issue of great importance for the design of the system layout. Depending on the infrastructure and on the physical system components, some patterns for connecting the nodes and switches may be more convenient than others. Therefore, it is important to know the advantages and disadvantages of each pattern from the point of view of the network performance.

Regardless of the specific interconnection pattern, Dragonfly performance may drop significantly due to the negative effects derived from congestion situations. The worst of them is the *Head-of-Line (HoL) blocking* effect [20], which appears when a packet at the head of a buffer blocks the rest of the packets of that buffer, even if they request access to available ports. The impact of this phenomenon is stronger in HPC networks as they do not discard congested packets, in contrast with “lossy” networks such as Internet. Many solutions have been proposed to deal with congestion and HoL blocking. In general, any solution focused on limiting traffic load in the network or avoiding/removing congestion situations may lower HoL-blocking probability. However, we think that it is more efficient if we focus on reducing/removing the HoL blocking instead of proactive or reactive solutions that make attempts to remove congestion trees. The problem with these solutions is that either they require global information of the network status, which may be difficult to obtain, or they are slow removing the congestion of the network. Hence, there exist many techniques specially designed to reduce the HoL-blocking effect. Most of them are based on storing separately different packets flows (or different groups of packet flows) at switch ports, so that the interaction among flows is reduced and so HoL-blocking. This approach requires to have at switch ports separate queues (or similar structures), that can be actually implemented in different ways, depending on switch architecture (for instance, by using Virtual Channels (VCs) [9], or by having buffers with several read ports). Moreover, packet flows can be mapped to these queues according to different policies (queuing schemes).

Some of these solutions are able to prevent completely HoL blocking but require additional resources not available in current switches [10,14,15]. On the other hand, there exist other feasible queuing schemes which reduce just partially HoL blocking [4,23,31]. We have noticed that among these feasible queuing schemes, the most efficient ones are those which are specially designed for specific network topologies and routing algorithms. In [12,17,33], some examples of topology-aware techniques are proposed that reduce the HoL blocking in a more efficient way and/or require fewer queues per port. The idea is to use the most efficient mechanism to reduce HoL blocking, which requires the minimum number of resources.

A key point to take into account when designing topology-aware queuing schemes is the behavior of the traffic inside the specific topology. This paper analyzes traffic flowing in Dragonfly networks configured with different interconnection patterns, using the Dragonfly minimal routing algorithm with the deadlock-avoidance mechanism proposed in [22]. Specifically, we describe congestion dynamics in these networks, as well as the network locations where hotspot points are more likely to appear. This analysis leads to a thorough knowledge of the HoL-blocking effect in the different Dragonfly configurations. On its side, this knowledge eases the design of efficient schemes suitable for these topologies that reduce the probability of HoL-blocking. In that sense, we describe in this paper the queuing scheme know as *Hierarchical Two-Level Queuing* (H2LQ), designed specifically for Dragonfly topologies using the minimal-routing algorithm. H2LQ assumes IQ-switches configured with several virtual channels (VCs) per port. Besides, at each switch port the VCs are grouped in two virtual networks (VNs) to prevent HoL blocking: the standard VN, where packets usually travel, and the escape VN used for avoiding deadlocks. We also evaluate the efficiency and the suitability of this scheme for different fully-connected Dragonfly interconnection patterns, in order to provide network designers with some insights about the dynamics of congestion related with different Dragonfly layouts.

The rest of this paper is organized as follows: Sect. 2 describes in detail the Dragonfly topology. We detail different options for connecting inter- and intra-group subnetworks based on fully-connected interconnection patterns, the deterministic minimal-path routing algorithm and the deadlock-avoidance mechanism. Section 3 explains network congestion and its effects and offers an overview of techniques that try to prevent or reduce them. Also, congestion dynamics in Dragonfly networks are analyzed. In Sect. 4, H2LQ is explained in detail, specifically the assumed switch architecture, the mapping policy, and how HoL blocking is reduced. Section 5 shows the performance evaluation of the different Dragonfly interconnection patterns configuring different queuing schemes. This evaluation is based on simulation experiments for both synthetic and real-based traffic scenarios. Finally, in Sect. 6, some conclusions are drawn.

2 The Dragonfly topology

The Dragonfly topology [22] is a high-performance hierarchical topology consisting of a set of groups, each one composed of several switches where several endnodes are attached. Switches belonging to the same group are interconnected by means of local channels that form an intra-group subnetwork. Groups are also interconnected by an inter-group subnetwork through the global channels of the switches. Any network topology can be used for intra- or inter- group subnetworks. However, in [22] it is suggested the use of fully-connected topologies for both levels of subnetworks, in order to improve their cost-effectiveness. Figure 1 shows a generic diagram of this topology. In detail, Dragonfly topology can be defined by three parameters:

- a Number of switches in each group.
- p Number of endnodes connected to each switch.
- h Number of channels within each switch that connect to other groups (global channels).

All this means that each switch has p terminal channels for connecting with the endnodes, $a - 1$ local channels which are part of the intra-group subnetwork, and h global channels which are part of the inter-group subnetwork. In a fully-connected Dragonfly topology there are $ap(ah + 1)$ endnodes, $ah + 1$ groups, and $a(ah + 1)$ switches, each one with $p+a+h-1$ ports. Despite the fact that Dragonfly parameters a, p and h may have any values, $a = 2p = 2h$ are recommended to balance the

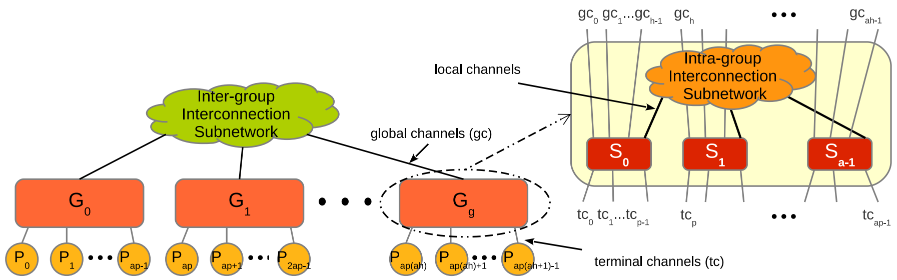

Inter-group Interconnection Subnetwork
Intra-group Interconnection Subnetwork
local channels
global channels (gc)
terminal channels (tc)
G_0
G_1
G_g
P_0 P_1 P_{ap-1} P_{ap} P_{ap+1} P_{2ap-1} $P_{ap(ah)}$ $P_{ap(ah)+1}$ $P_{ap(ah+1)-1}$
S_0 S_1 S_{a-1}
gc_0 $gc_1 ... gc_{h-1}$ gc_h gc_{ah-1}
tc_0 $tc_1 ... tc_{p-1}$ tc_p tc_{ap-1}

Fig. 1 Dragonfly topology diagram

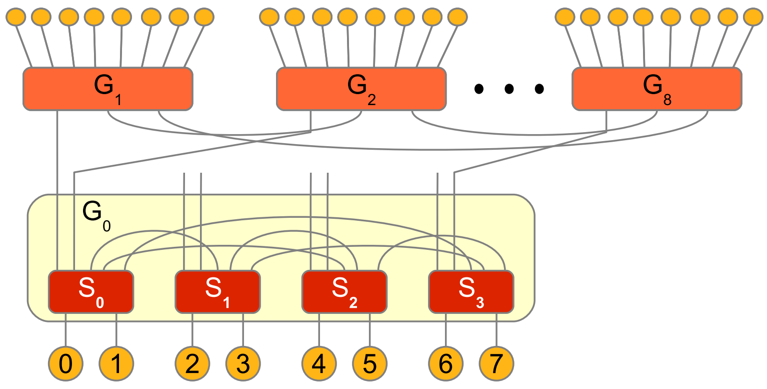


Fig. 2 Interconnection scheme of a fully-connected Dragonfly network ($a = 4$, $h = 2$, $p = 2$)

channel load. An example of interconnection scheme for a fully-connected Dragonfly with $a = 4$, $h = 2$ and $p = 2$ parameters is shown in Fig. 2.

2.1 Interconnection patterns

There are several fully-connected interconnection patterns that can be used for both intra- and intergroup subnetworks. In order to describe easily these patterns, we assume that the network is composed of e elements with $e - 1$ ports. An element may be either a switch or a group, as a group can be seen as a virtual switch with $a \times h$ connections to other switches. In that sense, in [8] and [18] several ways to arrange these connections are proposed:

- The *consecutive arrangement* consists in connecting the port i of element j to element i if $i < j$ and to element $i + 1$ otherwise. For instance, port 0 of element 1 is connected with element 0; port 1, with element 2; and port $e - 2$ with element $e - 1$. Figure 3a depicts this arrangement.
- The *palmtree arrangement* connects the first port of an element with the next element, the second port to the element two ahead, and so on. More formally, port

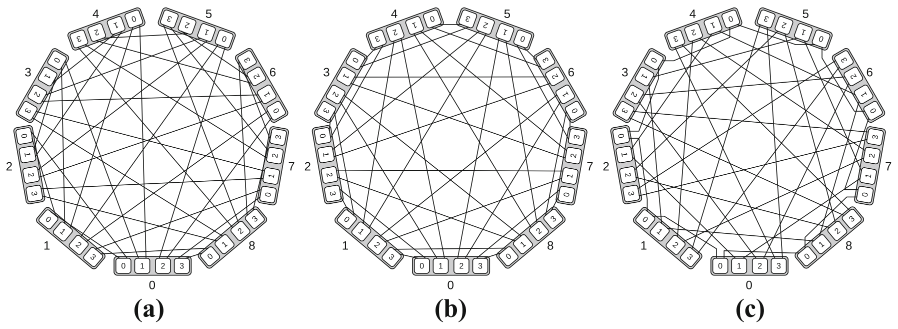


Fig. 3 Different fully-connected Dragonfly configurations. **a** Consecutive, **b** Palmtree, **c** Circular-based

i of element j is connected to element $(i + j + 1)\%e$ being $i \in \{0, \ldots, e-2\}$ and $j \in \{0, \ldots, e-1\}$. For instance, port 0 of element 2 is connected with element 3; port 2, with element 4; and port $e-2$, with element 1. This arrangement is depicted in Fig. 3b.

- The *circular-based arrangement* Each element connects its port 0 to the next element, its port 1 to the previous element, its port 2 to the element two ahead, its port 3 to the element two behind, etc. In a formal way, port i ($i \in \{0, \ldots, e-2\}$) of element j ($j \in \{0, \ldots, e-1\}$) is connected to element $(j + i/2 + 1)$ if i is even and element $(j - \lfloor i/2 \rfloor - 1)$ if i is odd. This arrangement assumes that elements have an even number of ports. If there is an odd number of ports, the last port $e-2$ of each element cannot be connected following the previous pattern. To solve this case, the port $e-2$ of element i must be connected to the port $e-2$ of element $i + e/2$, if $i < e/2$. For instance, port 0 of element 0 is connected with element 1; port 1, with element $e-1$; port 2, with element 2; and port 3, with element $e-2$. In Fig. 3c is shown this connection pattern.

2.2 Routing algorithm

There are several routing algorithms proposed for the Dragonfly topology [22], the most straightforward one being the minimal-path routing algorithm (MIN). Specifically, MIN routing defines three steps to route a packet from a source endnode s attached to switch S_s in group G_s, to a destination node d attached to switch S_d in group G_d:

1. If G_s is different from G_d and S_s does not have a global connection with G_d, packets are routed within G_s to an intermediate switch S_1 connected with G_d.
2. If G_s is different from G_d, packets are sent from the intermediate switch S_1 through a global channel to reach another switch S_2 in G_d.
3. If the arrival switch S_2 is not S_d, packets are routed within G_d from S_2 to S_d.

However, MIN routing is not deadlock free. To solve this problem, in [9] it is proposed to use two disjoint sets of virtual channels (VCs), supported by two queues at each port (see Fig. 4). Basically, each disjoint set of VCs, e.g. the set of all VC0 channels, forms a virtual network (VN) with different purposes. There are two VNs: the standard VN, which is composed of VCs, and the escape VN (EVN), which is composed of escape VCs (EVCs). First, packets use the standard VN (VC0) as long as they are traveling inside their source group or when they are traveling to another group (i.e., they are performing either step #1 or #2 of the routing process). Second, packets moving within the destination group (i.e., they are performing step #3 of the routing process) are mapped to the EVN (EVC0), in order to break the cycles in the channel-dependency graph. Note that the EVN is used only at input ports of the intragroup subnetwork.

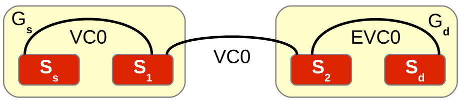


Fig. 4 Virtual channels used for avoiding deadlocks

3 Congestion: problems and solutions

3.1 Congestion derived problems

Congestion consists in intense traffic clogging paths within the network, thus slowing down traffic and degrading the network performance. The origin of congestion is contention, which occurs when several packet flows simultaneously request access to the same output port in a switch. Congestion could also occur when a destination node for one reason or another is not able to remove packets from the network at the speed they are received. In these cases, and assuming lossless networks,[1] any packet stored in a switch or network adapter port, which is not granted, remains blocked in a queue until its request to cross is accepted. If this situation persists in time, blocked packets delay the advance of other packets in the same queue, then the queues fill up and finally the flow-control backpressure propagates this congestion to other switches. Eventually, congestion may spread throughout the network reaching the endnodes, then increasing packet latency and, in general, degrading network performance. It is worth pointing out that in a congestion situation, not only the "hot flows" (i.e., flows contributing to congestion) are affected by the traffic jam. Indeed, flows not contributing to congestion (which are known as "cold flows") end up advancing at the same speed as hot flows because both cold and hot flows share queues. This is a particular case of the effect known as Head-of-line (HoL) blocking, which in general occurs when a packet requesting a busy port is blocked and prevents other packets stored behind in the same queue from advancing, even if these packets stored behind request free ports [11]. Thus, in a congestion situation hot flows may produce HoL blocking to cold flows if a hot packet is blocked at the head of a queue containing cold packets.

Actually, there are two types of HoL blocking derived from congestion, depending on where the congestion arises. On the one hand, if the HoL blocking affects a cold flow at the switch where the congestion is originated, it is called low-order HoL blocking [20]. On the other hand, if a cold flow is affected by congestion trees spreading from other switches, the resulting effect is known as high-order HoL blocking [19]. Regardless the type of HoL blocking, this effect can be considered as the main cause of network performance degradation upon congestion. Really, if hot flows did not interact with cold flows, congestion would not significantly degrade performance as the network paths followed only by hot flows would be used at a maximum speed [15].

3.2 Solutions to congestion

Nowadays two main approaches are followed to deal with congestion. On the one hand, *injection throttling* [28] relies on the switches to detect congestion and then informs the source endnodes which contribute to congestion that they must reduce their injection rates. Once congestion is removed, its derived problems are removed

[1] Lossless networks are those where packet discarding is not allowed. Note that lossless networks are the common option for HPC systems, the InfiniBand technology being the most significant example of HPC-based network technology.

too. This approach is the one followed by the InfiniBand specification. However, this technique presents several drawbacks: it does not scale with network size, and notifications may be too slow, so once source endnodes are warned to throttle the injection, the congestion information may be obsolete [14].

On the other hand, *queue-based flow-separation* techniques consist in the allocation of queues (or virtual channels [9] or virtual lanes [7]) to isolate as much as possible the hot flows, so preventing the HoL blocking caused by them. There are two queuing approaches. The first one consists in explicitly identifying hot flows in order to isolate them in dynamically-allocated queues. Proposals that follow this approach are the one described for ATLAS [21], the *Regional Explicit Congestion Notification* (RECN) mechanism [15], and *Efficient and cost-effective Congestion-Control* (EcoCC) [14]. However, these techniques require additional resources such as mechanisms to detect and separate hot flows, special control messages, or Content-Addressable Memories (CAM) to keep track of congested points at each port, which are not supported by current commercial interconnects. In the second approach, packets from different flows are stored in different queues according to a static mapping policy, independently of the traffic conditions. Some of these solutions have feasible implementations but others do not. For instance, *Virtual Output Queues at network level* (VOQnet) [10] maps each packet destination to a queue, and it prevents totally low- and high-order HoL blocking. However, it requires one queue per destination so it does not scale with network size. Among the feasible ones, the scheme known as Virtual Output Queues at switch level stores separately packets requesting different output ports, so that low-order HoL blocking is eliminated, but not the high-order one. Note that this requires as many queues per port as there are ports in the switch. Actually, this strategy can be implemented in different switch architectures, either by having at input buffers as many read ports as output ports (as in non-blocking switches [13]), or by dividing input buffers into as many logical queues as output ports, these queues sharing a single read port (as in [4] and [31]). Note the latter approach can be based on Virtual Channels to implement the queues, which allows to exploit VC-level flow control. Note also this approach requires look-ahead routing [26]. Other similar queuing schemes that reduce HoL blocking only partially are *Dynamically Allocated Multi-Queues* (DAMQs) [31], *Destination-Based Buffer Management* (DBBM) [23], and *Dynamic Switch Buffer Management* (DSBM) [24].

In general, the described approaches are not aware of neither the topology nor the routing algorithm. By contrast, other solutions are specially designed to be aware of these aspects so that HoL blocking is reduced by using fewer resources. For instance, queuing schemes such as Output-Based Queue Assignment (OBQA) [12,13] and vFTree [17] have been devised for fat-tree topologies and the routing algorithms proposed in [16] and [34], respectively. Similarly, Band-Based Queuing (BBQ) [33] is tailored to KNS topologies [27] with Hybrid-DOR routing algorithm [27]. In general, all these schemes leverage the available queues to separate traffic flows as much as possible, taking into account the traffic distribution determined by the topology and the routing algorithm. The queuing scheme described in Sect. 4 follows this approach, being suitable to Dragonfly topologies, using minimal-path deadlock-free routing algorithm, proposed in [22].

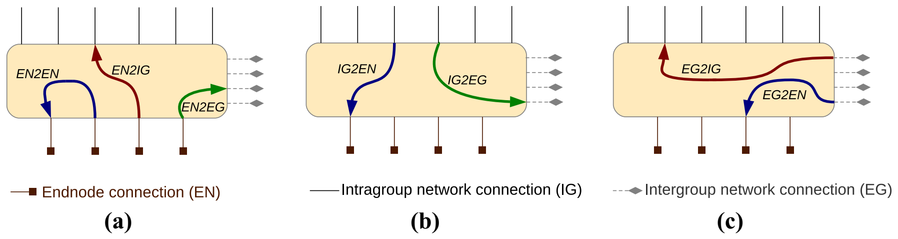

EN2EN
EN2IG
EN2EG
Endnode connection (EN)
(a)
IG2EN
IG2EG
Intragroup network connection (IG)
(b)
EG2IG
EG2EN
Intergroup network connection (EG)
(c)

Fig. 5 Dragonfly types of paths followed by traffic flows. **a** From endnodes. **b** From intragroup network. **c** From intergroup network

3.3 Analysis of congestion dynamics in Dragonfly networks

As explained in Sect. 2.2, the use of MIN routing algorithm in Dragonflies implies that a packet may cross a maximum of four switches (i.e., four hops) from its source until it reaches its destination. The different types of paths performed by the MIN routing algorithm are described below and summarized in Fig. 5. As local and global connections are not specified, these paths are valid for all the connection patterns described in the Sect. 2.1. Note that we use acronyms for referring connections from endnodes (EN), intra-group channels (IG), and inter-group channels (EG). We denote a path from A to B by using the number “2”, so that it can be expressed as “A2B”.

- **From endnodes** (EN). When a packet is generated, it may follow one of the following types of path:
  - To an endnode (EN2EN) attached to the same switch (blue arrow in Fig. 5a). Therefore, packets perform only one hop.
  - To the intragroup subnetwork (EN2IG). The destination endnode is in the same group but attached to a different switch, or it is in another group without connections with the source switch (red arrow in Fig. 5a).
  - To the intergroup subnetwork (EN2EG). The destination endnode belongs to a different group connected with the switch attached to the source endnode (green arrow in Fig. 5a).
- **From intragroup subnetwork** (IG). When a packet is received from the intra-group subnetwork, it may follow one of the following paths (see Fig. 5b):
  - To an endnode attached to the current switch (IG2EN). Packets generated in different groups are stored at the input port in the EVC (blue arrow in Fig. 5b).
  - To a different external group attached to the switch (IG2EN). This is the second hop performed by a packet as it only can change group once (green arrow in Fig. 5b).
- **From intergroup subnetwork** (EG). When a packet is received from the inter-group subnetwork, it may follow one of the following paths (Fig. 5c):
  - To an endnode of the current switch (EG2EN). Note that in this case, the EVN is not used (blue arrow in Fig. 5c).
  - To a different switch of the same group (EG2IG). Packets arrive from another group but their destinations are not attached to the arrival switch, i.e., this is

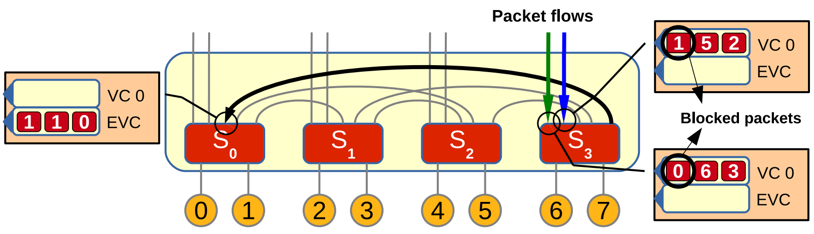


Fig. 6 Congested input ports due to contention in an EVC, producing high-order HoL blocking

their penultimate hop. Once they reach the next switch, they are stored in the EVC at the input port (red arrow in Fig. 5c).

In a congestion situation, these types of paths are likely to cause HoL blocking, especially when a hotspot appears. As it is described in Sect. 2.2, packets are mapped to the EVC at input ports when they move inside a group but they were generated at other groups. If the EVC at an input port of a switch becomes full, packets generated at other groups but requesting to reach that input port from switches of the same group (red arrow in Fig. 5c) will block due to flow control. Consequently, congestion spreads upstream as a group has only one connection with any other specific group. Indeed, if the queue of an input port directly connected to the intergroup subnetwork (EG) contains a packet addressed to a switch whose EVC is congested, all the packets coming from the group connected to this input port (i.e., EG2IG and EG2EN flows) cannot advance. Note that this happens even if these packets are not addressed to the switch with the congested EVC, so they suffer high-order HoL blocking.

Figure 6 shows an example of the described situation. It depicts one group of a Dragonfly network ($a = 4, h = 2, p = 2$) consisting of 9 groups, with two arrival flows of packets at each input port of the intergroup subnetwork of switch 3 (S_3). These EG2IG flows are addressed to endnodes 0 and 1, so that the packet at the head of each VC in these input ports is addressed to an endnode attached to switch 0 (S_0), sharing the same IG connection. However, as the EVC at the input port of S_0 is full, all the packets shown at the input ports of S_3 cannot flow, even if they are addressed to switches different to S_0 (so all of them are affected by high-order HoL blocking). Moreover, note that there are two groups unable to send packets to this group because of this HoL blocking situation, specifically the two groups connected to S_3, as each one has only one connection to the group in Fig. 6.

On the other hand, Dragonfly topology also suffers low-order HoL blocking when packets coming from other groups make their last hop and are stored in the EVC, waiting for being sent to their destination endnode. As there is a single VC (i.e., the EVC) for all these packets, which can be addressed to different endnodes, it may happen that there is contention to reach a specific endnode, so packets addressed to other endnodes are affected by HoL blocking. An example of this behavior is depicted in Fig. 7: note that the EVCs have packets addressed to endnode 0, but they cannot be delivered despite being free the port connecting endnode 0, as they are blocked by packets addressed to endnode 1.

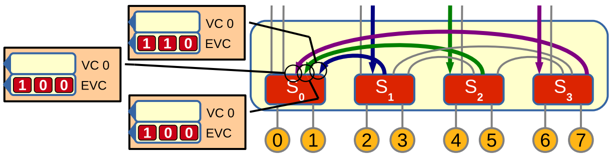


Fig. 7 Low-order HoL blocking produced in the final hop of packets from different groups

It is worth pointing out that the effects described above can happen regardless the specific Dragonfly interconnection pattern (see Sect. 2.1). Thus, all these interconnection pattern configurations require to develop solutions to reduce both high- and low-order HoL blocking.

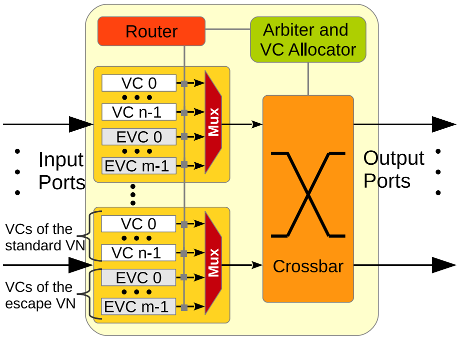
4 Hierarchical two-level queuing (H2LQ)

In this section, we describe a straightforward queuing scheme that reduces the HoL-blocking effect derived from congestion while the deadlock-freedom policy is respected. We assume the use of input-queued (IQ) switches with buffers at input ports being divided into several queues, each one of these queues being mapped to a virtual channel (VC). Besides, VCs at switch ports are grouped in two virtual networks (VNs): the standard VN and the escape VN (EVN). Note that, as described in previous section, HoL blocking must be reduced in both the standard VN and the EVN. Our approach proposes to expand both VNs so that they consist of several VCs. That is why we have called this technique *Hierarchical Two-Level Queuing* (H2LQ). H2LQ has been specially designed for fully-connected Dragonfly topologies (see Sect. 2.1) which use the MIN routing algorithm (Sect. 2.2), regardless the interconnection pattern.

4.1 Switch architecture

Figure 8 shows the switch architecture, based on the one proposed in [30], assumed in the design of H2LQ: an input-queued (IQ) and pipelined, switch architecture similar to those adopted in real products. For the sake of clarity, this example shows an

Fig. 8 Assumed input-queued switch architecture with 2 ports



unidirectional switch, but H2LQ is intended for bidirectional switches with any number of ports. Specifically, for a given Dragonfly size, the number of ports required for each switch is $a + h + p - 1$ (see Sect. 2). For instance, switches in a 1056-endnode Dragonfly ($a = 8$, $h = 4$, $p = 4$) require 15 ports while in 26406-endnode Dragonfly ($a = 18, h = 9, p = 9$), switches require 35 ports. The main components of the architecture are the input buffers, the crossbar, the router, and the arbiter and VC allocator.

As mentioned before, input buffers are divided into virtual channels (VCs), grouped in two disjoint sets of VCs or virtual networks (VNs): the standard VN and the escape VN. We assume that there are n VCs for packets traveling in the standard VN and m EVCs for packets in the EVN. The *router* performs the MIN routing algorithm explained previously in Sect. 2.2. The *arbiter and VC allocator* implements the *three-phased arbitration* described in [30] as well as the mapping policy described further in Sect. 4.2. We also assume the *Virtual Cut-Through*. In the next section, we describe the mapping policy of packets to VCs in both VNs.

4.2 Mapping packets to VCs in H2LQ

The aim of H2LQ is to reduce the HoL blocking affecting different traffic flows by separating them into the two VNs: the standard VN and the EVN. The number of VCs that each VN should configure is directly related with the Dragonfly parameters a and p. Ideally, each buffer should implement a VCs for the standard VN and p VCs for the EVN. However, this amount of VCs may not be available at the switches, especially when the size of the network grows. For that reason, the number of VCs configured for each VN has to be a divisor of a or p, respectively, in order to map the same number of flows to each VC.

H2LQ defines that in the standard VN, packets are mapped to VCs according to their destination switch, i.e., the switch where its destination node is attached, specifically as indicated by Eq. 1. In the EVN, packets are mapped to VCs based on their destination node, specifically as indicated by Eq. 2. Thereby, packet flows addressed to different switches are stored separately as long as they remain in the standard VN. While in the EVN, packet flows sent to different destinations do not share a VC.

$$\mathrm{VC}_{St\_\mathrm{VN}} = \frac{\mathrm{Destination}\%(a \times p)}{p} \%\#\mathrm{VCs}_{St\_\mathrm{VN}} \quad (1)$$

$$\mathrm{VC}_{\mathrm{EVN}} = \mathrm{Destination}\%\#\mathrm{VCs}_{\mathrm{EVN}} \quad (2)$$

In Eq. 1, “$\mathrm{VC}_{St\_\mathrm{VN}}$” is the VC in the standard VN where the packet will be stored in the next switch using its “Destination” as a parameter, a and p are the Dragonfly parameters, and “$\#\mathrm{VCs}_{St\_\mathrm{VN}}$” is the number of VCs configured in the standard VN. In Eq. 2, “$\mathrm{VC}_{\mathrm{EVN}}$” is the EVC where the packet will be stored in the next switch using its “Destination” as a parameter, and “$\#\mathrm{VC}s_{\mathrm{EVN}}$” is the number of VCs inside the EVN.

As an example, a Dragonfly network configured with $a = 8, h = 4$, and $p = 4$ may implement 8 VCs at the standard VN and 4 VCs at the EVN (8 + 4). If 12 VCs are not available at the input port of the switches, any combination of 8, 4, or 2 VCs

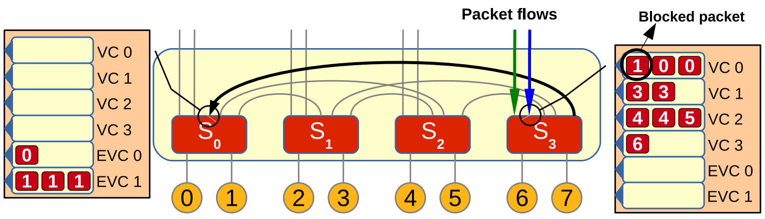


Fig. 9 Several VCs prevent high-order HoL blocking in the standard VN

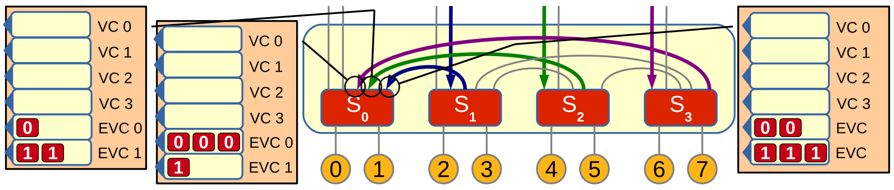


Fig. 10 Several EVCs reduce low-order HoL blocking produced in the EVN

could be used in the standard VN and 4, 2, or 1 at the EVN for instance 8 + 2, 4 + 1, 2 + 2, etc.

4.3 Advantages of the H2LQ mapping

Having several VCs in the standard VN reduces the high-order HoL blocking which arises in the input ports of the switches connected to other groups as explained in Sect. 3.3 (and depicted in Fig. 6). Figure 9 shows the same scenario as Fig. 6, but this time H2LQ is used. Notice that when we use H2LQ with 4 VCs in the standard VN, packets addressed to different switches do not share queue. Consequently, despite the fact that the input buffer of S_0 is full, as packets addressed to that switch are stored in $VC0$, packets stored in the remaining VCs (i.e., routed to other switches) can flow freely. If $VC0$ at $S3$ gets full, congestion could still spread to other groups, but only packets stored in $VC0$ will be affected.

On the other hand, as explained in Sect. 3.3, low-order HoL blocking may also appear in the EVN. Figure 10 shows the same scenario as Fig. 7, but this time H2LQ is used. Notice that in this case packets addressed to endnode 0 cannot be blocked by packets requesting the port connecting endnode 1, as they are stored in a different EVC. Thus, low-order HoL blocking is prevented.

Overall, combining the HoL-blocking reduction in both VNs leverages significantly the network performance, especially in hotspot situations, as the evaluation made in the next section shows.

5 Evaluation

In this Section, we show experiment results of H2LQ, described in Sect. 4, carried out by means of simulation. All the experiments have been performed with the simulator described in [32], based on the OMNeT++ framework [25], which has been adapted to include Dragonfly topologies, different Dragonfly connection patterns, and their MIN routing algorithm. We assume the switch architecture described in Sect. 4.1. Both switch-to-switch and endnode-to-switch channels are made through serial full-duplex pipelined links of 5 GB/s (40 Gbps) and 6 nanoseconds of link-propagation delay, i.e., a length of 1.2 meters and a delay of 5 ns/m. The propagation delay value corresponds to the InfiniBand specification [7]. The flow-control policy is credit-based and packet size (i.e., MTU) is 4096B. Buffers can store 64 packets, i.e., their size is 256 KB.

We have evaluated H2LQ compared to other queuing schemes from the state of art which were designed just to reduce HoL blocking, but not to provide deadlock-freedom. Therefore, in order to avoid deadlocks, we have provided these schemes with an EVN to perform properly the MIN routing algorithm (see Sect. 2.2). Note that this will improve the performance of some of them, as deadlock-freedom was not one of their features. These queuing schemes are named according to the following pattern: *technique-M_N*, where M is the number of VCs used in the main VN and N is the number of EVCs used in the EVN. Note that if N is 1, there is no HoL-blocking prevention within the EVN, but just deadlock-freedom. Specifically, the modeled queuing schemes are the following:

- **Deadlock Avoidance** (DLA). This is the basic scheme which only uses two VCs. One is used as a EVC so that this scheme does not prevent HoL blocking.
- **Virtual Output Queues at switch level**[2] (VOQsw) [4]. The standard VN supports at each input port as many VCs as ports are available in the switch, while the EVN has one EVC. In the standard VN, packets are mapped to VCs according to their requested output port at the switch.
- **Destination-Based Buffer Management** (DBBM) [23]. This scheme assigns a VC of the standard VN to a packet according the next formula: $Destination\% \#VCs$, where "Destination" is the packet destination and "#VCs", the number of VCs inside the standard VN. Note that as this technique uses EVCs in the EVN, the mapping policy used in that VN is the one described for our proposal in Sect. 4, so that DBBM gets some benefits in HoL-blocking reduction thanks to the use of EVCs in the EVN.
- **Hierarchical Two-Level Queuing** (H2LQ). This is the proposal described in Sect. 4.

In the next subsections, these queuing schemes are tested for each network configuration using synthetic traffic patterns (e.g. uniform and hotspot traffic), as well as communication patterns obtained from real applications.

5.1 Uniform traffic results

This section shows the evaluation of the different queuing schemes under the uniform (or random) synthetic traffic pattern. This traffic pattern consists in generating packets to random destinations in the network in a uniform way. The evaluated metrics are the

[2] Note that the switch model that we use is based on an IQ-switch architecture where each input buffer is divided into several logical queues sharing a single read port, as described in [4] and [31].

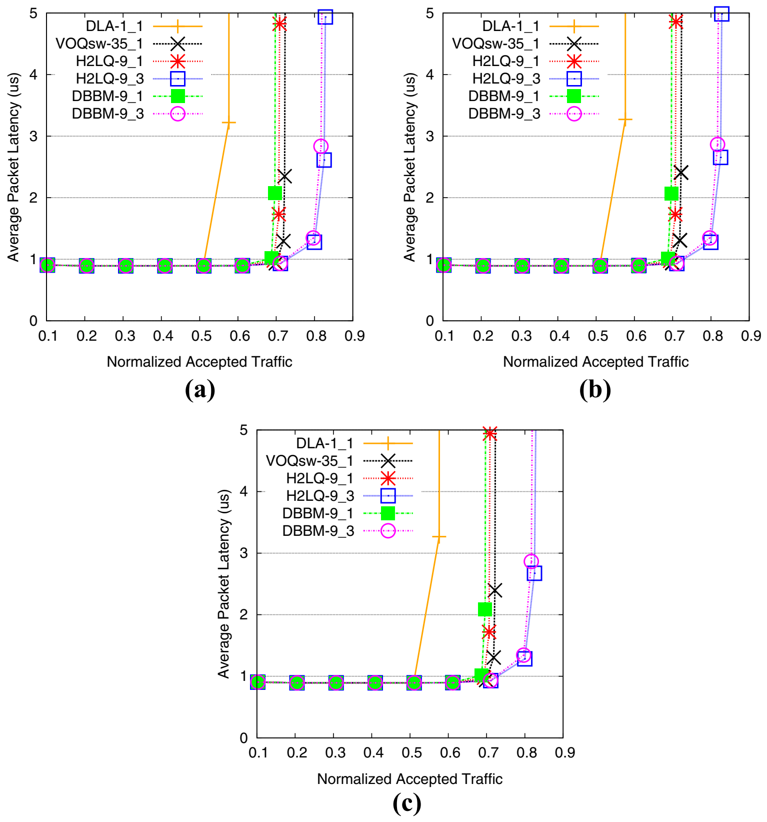


Fig. 11 Average packet latency versus normalized accepted traffic under uniform traffic, for different fully-connected Dragonfly interconnection patterns in a Dragonfly network configured with $a = 18$, $h = 9$, $p = 9$. **a** Absolute. **b** Palmtree. **c** Circular

accepted traffic normalized with respect to the link bandwidth and the packet latency. In the experiments, the generation rate of the endnodes is increased by 10 % from 10 % up to 100 % of the maximum theoretical performance.[3] For each point of generation rate, the network works during a million of nanoseconds and then, the metrics performance are recorded. Three network configurations have been evaluated, each one with a different interconnection pattern. In all of them, the Dragonfly topology parameters are $a = 18$, $h = 9$, $p = 9$, so there are 26,406 endnodes and 2,934 switches with 35 ports. The obtained data of the absolute, palm tree, and circular-based patterns are depicted in Fig. 11a–c respectively.

As it can be observed in Fig. 11, the uniform traffic results are very similar regardless the used interconnection pattern. The worst performance is achieved by DLA-1_1, as it does not provide any HoL-blocking prevention. On the other hand, VOQsw-35_1, DBBM-9_1, and H2LQ-9_1 techniques, which do not use EVCs, achieve a similar performance. VOQsw obtains better performance (i.e., smaller latency and

[3] We assume that this value is the $number\_of\_nodes \times link\_bandwidth$.

more accepted traffic) than the others at the cost of using more than three times the number of VCs (35 VCs are used), which makes it unfeasible in the current switch technology. Both, DBBM-9_1 and H2LQ-9_1 use the same amount of VCs which means the proposed mapping of them behaves almost identically with this traffic pattern. For the case of H2LQ-9_3 and DBBM-9_3, they obtain better performance than the previous techniques thanks the use of several EVCs to prevent HoL blocking in the EVN. In this case, our proposal also slightly improves DBBM in all the considered network configurations. Note that uniform traffic do not generate a significant amount of HoL blocking, since the congestion trees last a small number of cycles. Note also that DBBM takes advantage of our approach to use several EVCs to alleviate the HoL blocking in the EVN.

Therefore, H2LQ achieves the highest performance/VC-per-port ratio. When we use several VCs in the standard VN, the performance is increased by about 15 % in comparison with DLA-1_1. If several EVCs are used in the EVN, H2LQ performance is increased by an additional 15 %. Besides, if DBBM technique uses EVCs as our proposal does, its performance is improved by an additional increment.

5.2 Hotspot traffic results

The hotspot traffic pattern consists in configuring a set of several nodes to send packets to a single destination, known as *hotspot*. This situation generates a congestion tree in the network, so that the HoL blocking arises and strongly jeopardizes the network performance. This scenario is very a common situation in programs using parallel libraries such as MPI, where a process invokes a *gather* function (i.e., some nodes has to send traffic to the same node running the root MPI task), thus generating a hotspot. Specifically, in our experiments 25 % of the nodes send packets only to one of four possible hotspot destinations, so that each hotspot destination receives traffic from 6.25 % of the nodes generating the hotspot. The remaining nodes (75 %) generate uniform traffic. The metric used in the evaluation is the throughput normalized with respect the link bandwidth. Also, as for the uniform pattern, the generation rate of all the endnodes (i.e., those generating uniform traffic and those generating hotspot traffic) is increased by 10 % from 10 % up to 100 % of the maximum possible load to be injected. For each generation rate ratio, the network works during a million of nanoseconds and then, the performance metrics are recorded. As in the previous subsection, we have evaluated three network configurations with the mentioned interconnection patterns: absolute, palm tree, and circular-based. The Dragonfly topology parameters are $a = 18, h = 9, p = 9$, thus the network connects 26,406 endnodes and 2,934 switches with 35 ports.

Figure 12 shows that the obtained network performance is quite similar in the three scenarios, regardless of the Dragonfly interconnection pattern. As expected, DLA-1_1 obtains the worst performance because it is unable to reduce the HoL-blocking effect. The best performance is obtained by VOQSW-35_1 at the cost of using too many VCs. Both DBBM-9_3 and H2LQ-9_3 techniques obtain less performance than VOQsw but requiring only a third of VCs. DBBM-9_1 always achieves about 15 % lower performance than the two previous techniques. By contrast, H2LQ-9_1 is able

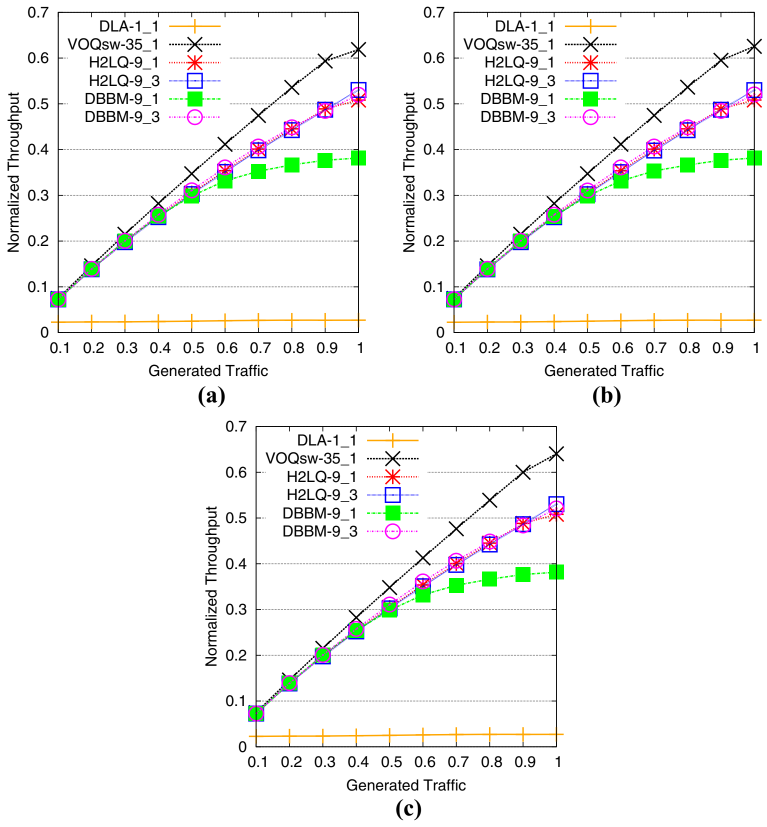


Fig. 12 Normalized throughput vs generated traffic for different fully-connected Dragonfly interconnection patterns in a Dragonfly network configured with $a = 18$, $h = 9$, $p = 9$ under hotspot traffic. **a** Absolute. **b** Palmtree. **c** Circular

to obtain a performance much closer to its version with EVCs in the EVN, showing that the mapping of flows to VCs in the standard VN indicated by Sect. 4.2 is more efficient to reduce HoL blocking than the DBBM mapping with just one EVC.

Therefore, H2LQ has been designed taking into account the dynamics of congestion in Dragonfly networks, so that it efficiently deals with HoL blocking in congestion situations. Indeed, our proposal achieves the best performance/ VC-per-port ratio as it is able to keep the performance when no EVCs are configured. Besides, other queuing schemes can benefit from the use of EVCs in the EVN, as DBBM does, increasing its performance by a 15 %. Note that, as it has been mentioned before DBBM does not provide any additional mechanism for deadlock prevention in Dragonfly networks, so that we have provided EVCs to it for performing a fair comparison.

5.3 Traces based on real traffic results

This Section shows the experiment results obtained by simulating the queuing schemes mentioned before, under traffic traces obtained from two real applications: the

Table 1 Evaluated configurations

#	Network	Pattern	Trace	# Traces	#Endnodes	#Tasks
1	6a3h3p	Absolute	smtv	1	342	256
2	6a3h3p	Palm Tree	smtv	1	342	256
3	6a3h3p	Circular-based	smtv	1	342	256
4	6a3h3p	Absolute	graph500	1	342	256
5	6a3h3p	Palm Tree	graph500	1	342	256
6	6a3h3p	Circular-based	graph500	1	342	256
7	8a4h4p	Absolute	smtv	4	1056	1024
8	8a4h4p	Palm Tree	smtv	4	1056	1024
9	8a4h4p	Circular-based	smtv	4	1056	1024
10	8a4h4p	Absolute	graph500	4	1056	1024
11	8a4h4p	Palm Tree	graph500	4	1056	1024
12	8a4h4p	Circular-based	graph500	4	1056	1024

Graph500 benchmark [1] and the STMV test included in the NAMD scalable molecular dynamics application [29]. The communication pattern of these MPI applications has been modeled by means of the VEF traces suite [5] and it generates the VEF traffic traces including the task dependencies (i.e., self-related traces). This framework also provides the TraceLIB library which can be integrated with any third-party simulator as long as it uses its simple API. In order to generate the VEF traces traffic into our OMNeT++-based simulator, we have integrated TraceLIB with our simulator. We have used 256-tasks traces from the repository at [2] both for the Graph500 and SMTV applications.

Table 1 shows the evaluated Dragonfly configurations. Two Dragonfly network topologies with different size have been evaluated: the first one has 342 endnodes ($a = 6, h = 3, p = 3$) and the second one has 1056 endnodes ($a = 8, h = 4, p = 4$). Each network is evaluated using the three different interconnection patterns. The network configurations using the 342-endnode network only executes one trace, while the configurations using the 1056-node network combine four traces at the same time. These trace tasks are mapped to endnodes in a random way and, as there are more endnodes than tasks, several endnodes are idle.[4] The metric evaluated is the runtime required to complete all the trace communications normalized against the execution time of the DLA-1_1 technique, as this scheme does not provide any HoL-blocking reduction. Therefore, we want to measure the impact of using different queuing schemes in the application runtime.

In Fig. 13 it is shown the obtained data from simulating configuration #1, #2, and #3. The SMTV communication pattern does not create significant network congestion, so that under this traffic conditions, it is clear that configuring several VCs for reducing HoL blocking has small impact on the reduction of the application runtime (about

[4] Note that in real HPC clusters it is typical that a queuing-based scheduler is in charge of mapping jobs to the available processing nodes, so that several applications can be run at the same time.

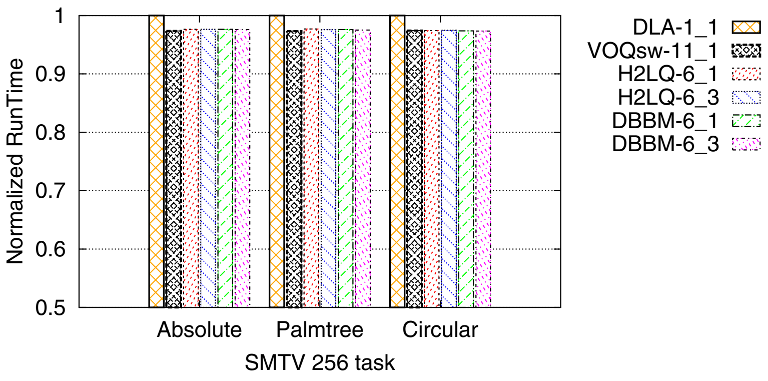


Fig. 13 Normalized runtime of the communication pattern obtained from the execution of SMTV virus benchmark in NAMD scalable molecular simulator in Dragonfly $a = 6, h = 3, p = 3$ with different interconnection patterns

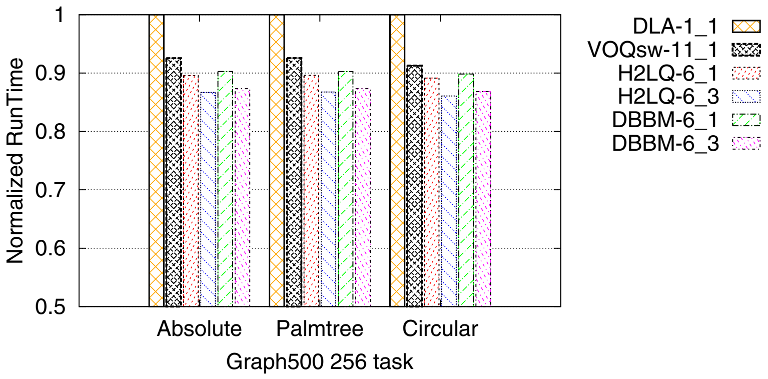


Fig. 14 Normalized runtime of the 256-task communication pattern obtained from Graph500 benchmark in Dragonfly $a = 6$, $h = 3$, $p = 3$ with different interconnection patterns

3–4 %). However, this trace has been used to test and validate the consistency of the simulation model.

Figure 14 depicts the results of configurations #4, #5, and #6 of Table 1. The obtained results for the three different Dragonfly interconnection patterns for these configurations are barely the same which confirm that no matter what of the connection pattern is used to cable the Dragonfly. Specifically, VOQsw-11_1, H2LQ-6_1, and DBBM-6_1 techniques, which only use VCs in the standard VN, achieve less improvement in the runtime than H2LQ-6_3 and DBBM-6_3, as they configure several EVCs. VOQsw-11_1 reduces about 7 % the runtime with respect to to DLA-1_1, while H2LQ-6_1 or DBBM-6_1 improve it about 10 %. On the other hand, H2LQ-6_3 and DBBM-6_3 are able to reduce the runtime about 14 %.

Figure 15 depicts the results from configuration #7, #8, and #9, i.e., Dragonflies of 1056 endnodes. As it was described before, the SMTV trace is not able to generate a significant network congestion. Therefore, only between 3 and 4 % of improvement in the runtime is obtained, regardless of the used technique for preventing HoL blocking,

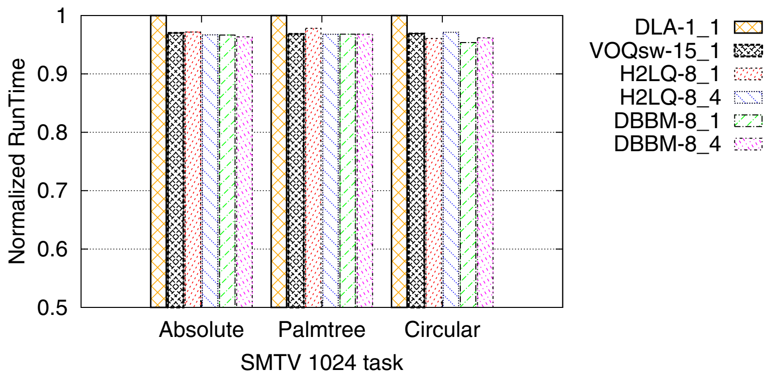


Fig. 15 Normalized runtime of the communication pattern obtained from the execution of SMTV virus benchmark in NAMD scalable molecular simulator in Dragonfly $a = 8, h = 4, p = 4$ with different interconnection patterns

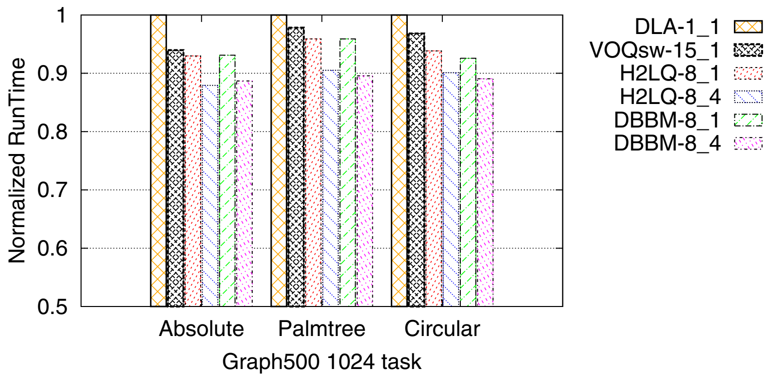


Fig. 16 Normalized runtime of the communication pattern obtained from Graph500 benchmark in Dragonfly $a = 8$, $h = 4$, $p = 4$ with different interconnection patterns

but we have used it just to verify the consistency of our simulation model when the Dragonfly size increases.

Figure 16 shows the normalized runtime from the traces of configuration #10, #11, and #12. As these scenarios are a combination of four graph500 traces, the results are similar to the obtained for configuration #4, #5, and #6. Although the represented data is different for each configuration, the results are barely the same. These differences are due to the random mapping of tasks to nodes, as the traffic behavior changes. The techniques that obtain less improvement (between 3 and 7 %) are those which do not prevent HoL blocking in the EVN, i.e., VOQsw-15_1 , H2LQ-8_1, and DBBM-8_1. However, H2LQ-8_4 and DBBM-8_4 achieve the best runtime reducing the DLA-1_1 one about 10–12 %.

Summing up, using an appropriate queuing scheme to prevent HoL blocking reduces the runtime of the evaluated communication patterns obtained from real applications, regardless the Dragonfly interconnection pattern. Although using several VCs in the standard VN decrease the required time, the most efficient techniques are those which also prevent HoL blocking in the EVN, as our proposal does, which reduce even more the execution time.

6 Conclusions

Nowadays, the Dragonfly family of topologies has become very popular for interconnecting processing and storage nodes of HPC systems and data centers, due to their good properties for high bisection bandwidth, low latency, low diameter, path diversity, etc. However, the Head-of-Line (HoL) blocking derived from congestion situations may decrease their performance. This paper explains the technique know as *Hierarchical Two-Level Queuing* (H2LQ), which prevents the HoL blocking effect. H2LQ is designed specially for fully-connected Dragonfly topologies, regardless of the fully-connected interconnection pattern used for inter-group or intra-group subnetworks. Also, H2LQ assumes the use of the Dragonfly minimal routing algorithm as well as the use of an escape virtual network for avoiding deadlocks. The benefits provided by H2LQ come not only from using a clever mapping of packets to virtual channels to prevent HoL blocking in the standard virtual network, but also from implementing an efficient VC-mapping policy of traffic flows to VCs in order to prevent this effect in the escape virtual network. Other techniques implementing this idea would take advantage of the benefits provided by H2LQ to reduce HoL blocking.

Furthermore, we have evaluated H2LQ through simulation experiments under certain traffic scenarios by using synthetic traffic and communication patterns extracted from real applications. From this evaluation experiments, we can conclude that H2LQ obtains successful results compared with other solutions for reducing HoL blocking, regardless of the fully-connected pattern used in the Dragonfly topology. In addition, H2LQ achieves a high scalability and it is able to keep the network performance by using an affordable amount of VCs. Besides, thanks to this idea of the H2LQ approach, other queuing schemes may be improved by using several VCs in the escape network.

References

1. The Graph 500 List. www.graph500.org
2. VEF traces: An easy way to model MPI traffic in network simulators. http://www.i3a.uclm.es/VEFtraces/
3. Alversons B, Froese E, Kaplan L, Roweth D (2012) Cray XC Series Network. Tech. rep. Cray Inc,
4. Anderson T, Owicki S, Saxe J, Thacker C (1993) High-Speed Switch Scheduling for Local-Area Networks. ACM Transactions on Computer Systems 11(4):319–352
5. Andujar FJ, Villar JA, Sanchez JL, Alfaro FJ, Escudero-Sahuquillo J (2015) VEF Traces: A Framework for Modelling MPI Traffic in Interconnection Network Simulators. In: Cluster Computing (CLUSTER), 2015 IEEE International Conference on, pp. 841–848. doi:10.1109/CLUSTER.2015.141
6. Arimilli B, Arimilli R, Chung V, Clark S, Denzel W, Drerup B, Hoefler T, Joyner J, Lewis J, Li J, Ni N, Rajamony R (2010) The PERCS High-Performance Interconnect. In: High Performance Interconnects (HOTI), 2010 IEEE 18th Annual Symposium on, pp. 75–82. doi:10.1109/HOTI.2010.16
7. Association IT (2007) InfiniBand Architecture Specification. http://www.infinibandta.org
8. Camarero C, Vallejo E, Beivide R (2014) Topological Characterization of Hamming and Dragonfly Networks and Its Implications on Routing. ACM Trans. Archit. Code Optim. 11(4):39:1–39:25. doi:10.1145/2677038

9. Dally W (1992) Virtual-Channel Flow Control. IEEE Trans. on Parallel and Distributed Systems 3(2):194–205
10. Dally W, Carvey P, Dennison L (1998) Architecture of the Avici terabit switch/router. In: 6th Hot Interconnects, pp. 41–50
11. Dally WJ, Towles B (2003) Principles and Practices of Interconnection Networks. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA
12. Escudero-Sahuquillo J, García PJ, Quiles FJ, Flich J, Duato J (2011) OBQA: Smart and cost-efficient queue scheme for Head-of-Line blocking elimination in fat-trees. J. Parallel Distrib. Comput. 71(11):1460–1472
13. Escudero-Sahuquillo J, Garcia PJ, Quiles FJ, Reinemo SA, Skeie T, Lysne O, Duato J (2014) A New Proposal to Deal with Congestion in InfiniBand-based Fat-trees. J. Parallel Distrib. Comput. 74(1):1802–1819
14. Escudero-Sahuquillo J, Gran E, Garcia-Garcia P, Flich J, Skeie T, Lysne O, Quiles F, Duato J (2014) Efficient and Cost-Effective Hybrid Congestion Control for HPC Interconnection Networks. Parallel and Distributed Systems, IEEE Transactions on PP(99):1–1. doi:10.1109/TPDS.2014.2307851
15. Garcia P, Quiles F, Flich J, Duato J, Johnson I, Naven F (2006) Efficient, Scalable Congestion Management for Interconnection Networks. Micro, IEEE 26(5):52–66
16. Gomez C, Gilabert F, Gomez M, Lopez P, Duato J (2007) Deterministic versus Adaptive Routing in Fat-Trees. In: Workshop CAC in conjunction with the IPDPS, p. 235
17. Guay WL, Bogdanski B, Reinemo SA, Lysne O, Skeie T (2011) vFtree - A Fat-Tree Routing Algorithm Using Virtual Lanes to Alleviate Congestion. In: Proc. of IPDPS, pp. 197–208
18. Hastings E, Rincon-Cruz D, Spehlmann M, Meyers S, Xu A, Bunde DP, Leung VJ (2015) Comparing Global Link Arrangements for Dragonfly Networks. In: Cluster Computing (CLUSTER), 2015 IEEE International Conference on, pp. 361–370. doi:10.1109/CLUSTER.2015.57
19. Jurczyk M, Schwederski T (1996) Phenomenon of Higher Order Head-of-Line Blocking in Multistage Interconnection Networks under Nonuniform Traffic Patterns. IEICE Transactions on Information and Systems E79–D(8):1124–1129
20. Karol MJ, Hluchyj MG, Morgan SP (1987) Input versus output queuing on a space-division packet switch. IEEE Transactions on Communications. COM–35:1347–1356
21. Katevenis M, Serpanos D, Spyridakis E (1998) Credit-flow-controlled ATM for MP interconnection: The ATLAS I single-chip ATM switch. In: High-Performance Computer Architecture, 1998. Proceedings., 1998 Fourth International Symposium on, pp. 47–56
22. Kim J, Dally WJ, Scott S, Abts D (2008) Technology-Driven, Highly-Scalable Dragonfly Topology. SIGARCH Comput. Archit. News 36(3):77–88
23. Nachiondo T, Flich J, Duato J (2010) Buffer Management Strategies to Reduce HoL Blocking. Parallel and Distributed Systems, IEEE Transactions on 21(6):739–753. doi:10.1109/TPDS.2009.63
24. Olesinski W, Eberle H, Gura N (2009) Scalable alternatives to virtual output queueing. In: Proc. IEEE ICC, pp. 1–6
25. OpenSim Ltd: OMNeT++ Discrete Event Simulator. http://omnetpp.org/
26. Peir JK, Lee YH (1993) Look-ahead routing switches for multistage interconnection networks. Journal of Parallel and Distributed Computing 19(1):1–10. doi:10.1006/jpdc.1993.1085
27. Penaranda R, Gomez C, Gomez M, Lopez P, Duato J (2012) A New Family of Hybrid Topologies for Large-Scale Interconnection Networks. In: Network Computing and Applications (NCA), 2012 11th IEEE International Symposium on, pp. 220–227
28. Pfister G, Gusat M, Denzel W, Craddock D, Ni N, Rooney W, Engbersen T, Luijten R, Krishnamurthy R, Duato J (2005) Solving Hot Spot Contention Using InfiniBand Architecture Congestion Control. In: Proc. of Int. Workshop HPI-DC
29. Phillips JC, Braun R, Wang W, Gumbart J, Tajkhorshid E, Villa E, Chipot C, Skeel RD, Kalé L, Schulten K (2005) Scalable molecular dynamics with NAMD. Journal of Computational Chemistry 26(16):1781–1802. doi:10.1002/jcc.20289
30. Pinkston TM, Duato J (2006) Appendix E. In: Elsevier (ed.) Computer Architecture: A Quantitative Approach. Morgan Kaufmann Publishers
31. Tamir Y, Frazier G (1992) Dynamically-Allocated Multi-Queue Buffers for VLSI Communication Switches. IEEE Trans. on Computers

32. Yebenes P, Escudero-Sahuquillo J, Garcia P, Quiles F (2013) Towards Modeling Interconnection Networks of Exascale Systems with OMNet++. In: Parallel, Distributed and Network-Based Processing. doi:10.1109/PDP.2013.36
33. Yebenes Segura P, Escudero-Sahuquillo J, Gomez Requena C, Garcia P, Quiles F, Duato J (2013) BBQ: A Straightforward Queuing Scheme to Reduce HoL-Blocking in High-Performance Hybrid Networks. In: Euro-Par 2013 Parallel Processing, vol. 8097, pp. 699–712
34. Zahavi E, Johnson G, Kerbyson DJ, Lang M (2010) Optimized InfiniBand^{TM} fat-tree routing for shift all-to-all communication patterns. Journal of CCPE 22(2):217–231