Implementation and Evaluation of a Hybrid Network Topology in an Infiniband-based HPC cluster

Gabriel Gomez-Lopez (✉ gabriel.gomez@uclm.es)
Universidad de Castilla-La Mancha

Jesus Escudero-Sahuquillo
Universidad de Castilla-La Mancha

Pedro Javier Garcia
Universidad de Castilla-La Mancha

Francisco J. Quiles
Universidad de Castilla-La Mancha

Research Article

Keywords: High-Performance computing, Datacenters Interconnection networks, hybrid topologies, InfiniBand

Posted Date: October 18th, 2023

DOI: https://doi.org/10.21203/rs.3.rs-3449785/v1

License: ☕️ This work is licensed under a Creative Commons Attribution 4.0 International License. Read Full License

Additional Declarations: No competing interests reported.
Implementation and Evaluation of a Hybrid Network Topology in an Infiniband-based HPC cluster

Gabriel Gomez-Lopez*, Jesus Escudero-Sahuquillo†, Pedro Javier Garcia†, Francisco J. Quiles†

*Department of Computing Systems, Universidad de Castilla-La Mancha, Spain.

*Corresponding author(s). E-mail(s): gabriel.gomez@uclm.es; jesus.escudero@uclm.es; pedrojavier.garcia@uclm.es; francisco.quiles@uclm.es

†These authors contributed equally to this work.

Abstract

The InfiniBand networking technology is widely utilized in modern high-performance systems. This work describes the implementation of the hybrid network topology known as KNS in a real HPC cluster using an InfiniBand interconnection network. We have used the cluster CELLIA (Cluster for the Evaluation of Low-Latency Architectures), consisting of 50 computing nodes equipped with InfiniBand network cards, and up to 50 8-port InfiniBand switches, which allow us to build several topologies. We have implemented the KNS routing algorithm in the subnet manager software (OpenSM). We have also evaluated the performance of the KNS topology using well-known benchmarks, such as HPCC, HPCG, Graph500, and Netgauge. The obtained results show that the low-diameter KNS topology is an efficient and cost-effective alternative to interconnect the computing and storage nodes in HPC clusters. As far as we know, no known InfiniBand system has implemented this topology before.

Keywords: High-Performance computing, Datacenters Interconnection networks, hybrid topologies, InfiniBand
1 Introduction

In recent years, the fastest-growing applications and services, among those requiring high-performance computing, have been the ones related to Big-Data and Deep Learning [1], which have revolutionized the design trends of supercomputers and data centers. These technologies are characterized by handling a large amount of data, which must be processed in a short period of time. As a result, the systems for processing this data require ever-increasing computing and storage capacity.

The computational capacity required to solve the complex problems caused by the above applications can be huge. Therefore, high-performance computing (HPC) clusters or supercomputers aim at increasing computational capacity year by year. Indeed, the objective of the previous decade was to reach the ExaFlop computing barrier (i.e., $10^{18}$ Floating Point Operations per Second) by 2018, but this achievement was postponed until June 2022 when the Frontier supercomputer reached 1.1 ExaFlops [2].

Today’s HPC systems are composed of a large number of processing nodes working in parallel. One of the most crucial requirements to guarantee maximum performance in these systems is to provide high bandwidth and low latency to the communication operations among these nodes. Therefore, in an HPC system, the interconnection network is a key subsystem, which must be sufficiently fast to avoid being the bottleneck of the entire system.

In the last few years, there have been a lot of efforts both from the industry and academia to design high-performance interconnection networks offering high communication bandwidth and low latency, while being reliable and cost-effective. These efforts have been focused on different design issues, such as network topology, routing algorithms, or policies for congestion management or quality of service. Among them, the network topology and routing algorithm are very important aspects as they determine the maximum bandwidth and expected latency that the application communication patterns will experience.

The network topology defines the disposition of the HPC system nodes and how these nodes are interconnected. Over the years, topologies have evolved from traditional networks, such as buses or rings to more complex but more efficient ones, such as Tori or Fat trees. In the last few years, the design trend for network topologies has been to provide low diameter (i.e., the number of hops that a packet needs to cross the network is as reduced as possible), path diversity (i.e., multiple paths between a pair of nodes), and low acquisition cost.

Furthermore, cost-effective network topologies require efficient routing algorithms. The routing algorithm defines the path that a packet must follow since it is injected into the network until it reaches its destination. To leverage the properties of the newly proposed network topologies, they have been devised efficient routing algorithms either deterministic (i.e., a packet always follows the same route between two nodes), oblivious (i.e., a packet is sent randomly through one of the possible paths in the network) or adaptive (i.e., a packet is sent through the less loaded path in the topology).

Regarding the topologies taxonomy, they can be classified depending on the way the devices are connected: direct, indirect, and hybrid [3]. Direct topologies have orthogonal structures with the nodes organized in several dimensions. Each node is connected to a neighboring node in each dimension. These networks are easy to implement if they
only have 2 or 3 dimensions while using more than 3 dimensions increases the layout complexity and the network diameter. Moreover, they are not scalable in terms of performance. By contrast, indirect topologies appeared as an alternative to improve the bandwidth and latency provision of direct networks. In this topology, the nodes are connected via switches organized in different stages. As the number of nodes grows, indirect networks achieve smaller network diameters than direct networks, but the number of devices required grows significantly, thus the network cost increasing as well.

The third type of the mentioned topologies, i.e., hybrid topologies, aims to take the best qualities of direct and indirect networks and avoid their disadvantages. Indeed, hybrid topologies provide high bandwidth, low latency, and a reduced diameter (like indirect topologies), while they achieve a reduced cost (like direct ones) while keeping the rich path diversity offered by both indirect and direct topologies. One example of a hybrid network is the KNS [4], which is the one in which we base this work.

In this work, we implement and evaluate the KNS network topology in a real HPC cluster using the InfiniBand interconnection network technology. Specifically, the KNS follows a dimensional structure as a direct network, where the nodes in each dimension are connected by a small indirect network. Hence, the KNS topology offers similar bandwidth and latency to indirect networks but at a lower cost. Moreover, as the KNS topology defines its own routing algorithm, i.e., the Hybrid DOR, which is an adaptation for the KNS network of the Dimension Order Routing (DOR) algorithm, we have also implemented the Hybrid DOR routing algorithm in the InfiniBand control software.

To evaluate the implementation of the KNS topology and the Hybrid DOR routing, we have used the research CELLIA cluster Cluster for the Evaluation of Low-Latency Interconnection Architectures. This, composed of up to 50 computing nodes uses an InfiniBand-based interconnection network, so all the nodes contain InfiniBand network cards. Moreover, the CELLIA cluster, which is not used for production but for research activities, contains up to 50 8-port switches that allow us to configure different network topologies. Note that the InfiniBand technology currently supports many direct, indirect, and hierarchical networks but, as far as we know, the KNS topology has not been yet implemented using the InfiniBand technology. Indeed, the InfiniBand hardware allows the cabling for the KNS connection pattern. However, we need to implement the Hybrid-DOR routing algorithm in the network control software. Specifically, we will use OpenFabrics Software (OFS), which includes an open-source implementation of the InfiniBand subnet manager: OpenSM.

In summary, the main contributions of this work are the following:

- We have built for the first time the KNS topology in a real InfiniBand-based cluster. We propose a wiring scheme and a simple layout for this topology.
- We have implemented the Hybrid-DOR routing for KNS networks in the OpenSM software, so the network devices are identified and the routing tables at network switches are conveniently populated.
- We have evaluated the performance of the proposed implementation using well-known parallel applications and benchmarks, such as HPCC, HPCG, Graph500, and Netgauge. The obtained results show that the low-diameter KNS topology is an efficient and cost-effective alternative to Fat trees in HPC systems.
The remainder of this work is organized as follows. In Section 2, we review the background of HPC networks, the hybrid KNS topology, and the InfiniBand architecture. Section 3 describes the tools used to implement Hybrid-DOR in OpenSM. It also describes the cluster used to evaluate the performance of the topology and the tools used to measure the performance of the topology. In Section 4, the performance results of several experiments performed on a real InfiniBand cluster are analyzed. Finally, some conclusions are drawn in Section 5.

2 Background

In this section, we describe the basic background concepts of interconnection networks. In particular, we pay attention to the different types of network topologies, and their advantages and disadvantages, which will help to understand the efficiency of the KNS topology. Next, we focus on the properties of the KNS topologies, including their routing algorithm, i.e., the Hybrid-DOR routing. Finally, we provide some details regarding the InfiniBand network technology.

2.1 Interconnection Networks

High-performance computing (HPC) systems require an interconnection network that allows the computing nodes to communicate quickly and efficiently when running parallel applications. Designing an efficient interconnection network poses numerous challenges to guarantee that it is not the bottleneck of the whole system, such as scalability, physical limitations, cost constraints, and performance needs.

Specifically, an interconnection network consists of links that make point-to-point connections between two network components, these components can be two switches, a node connected to a switch, or two nodes directly connected. The switches are composed of a variety of ports and are responsible for interconnecting the network devices and forwarding the incoming packets from one port to the appropriate output port to reach each packet destination. Finally, the network interfaces connect a node’s internal system bus to the physical network link. The topology defines how these elements are connected. The ideal network would have all its nodes directly connected, and this is not possible with a large number of nodes. Therefore, different topologies have been studied to connect many nodes in the most efficient way possible. According to their topology, there are mainly the following types of interconnection networks: direct, indirect, and hybrid.

In a direct network, the nodes are distributed in orthogonal structures. Each node has a router, which is responsible for sending and receiving data. Each node’s router is directly connected to the router of other neighboring nodes, so these networks are also called router-based networks. In HPC systems, it is also common to use dedicated routers, which replace the router integrated into the nodes; in this case, the network interface of the nodes is more straightforward. One example of this integration is made in the Fugaku supercomputer and its proprietary Tufu-D network technology [5]. Direct networks are scalable networks in terms of cost, which grows linearly with the number of network devices. However, they do not scale well in terms of performance since increasing the number of nodes increases both the average distance
between them and the latency. In addition, the bandwidth of the bisection is small compared to that of other topologies. Direct networks are defined with the notation $k$-ary $n$-cubes, where $k$ is the number of nodes per dimension, and $n$ is the number of dimensions. The most common networks of this type are Meshes, Torus, and Hypercubes. The simplicity of the connection patterns of direct networks allows efficient and straightforward routing algorithms to be implemented. Hypercubes offer the best performance when the number of nodes in the network grows. This is because having only two nodes per dimension, the average distance of the network will be less than in other direct topologies, wherein in each dimension, more nodes have to be traversed to reach the destination. The problem with hypercubes is the cost since many more links are needed to connect all the dimensions, and the devices need more ports since they must have one port per dimension.

In indirect networks, nodes are connected through switches, rather than directly. The nodes in these networks have only one link that is connected to a switch, and through this, they have access to the rest of the network. In indirect networks, a switch can be connected to end nodes and other switches simultaneously or only be connected to other switches. The ideal way to connect the $n$ nodes of an indirect network would be through a switch with $n$ ports [3], but the physical and cost limitations would be prohibitive in systems with thousands of nodes. When the number of nodes increases in these networks, so does the number of switches and links. The average diameter and distance do not increase or do in small proportions, so these networks are scalable in terms of performance. The disadvantage is that the number of resources required (switches, cables, and power consumption) grows considerably, so they are limited in cost and physical constraints. The most common indirect networks are the Multistage Interconnection Networks (MIN), often referred to as Fat trees because this is the connectivity pattern most commonly used in MINs. Its topology consists of a series of stages formed by switches that connect to end nodes or switches of other stages. There are different types of Fat trees, depending on how the switches of some stages are connected to others. An example of Fat trees is the Slimmed Fat trees (SFT) [6]. In an SFT network, the first stages are connected with a larger number of links than in the upper stages. This increases the bandwidth between nodes closer to each other and, therefore, takes better advantage of the locality required by some traffic patterns in the network. These networks allow the connection of many nodes without losing the performance that is lost in direct networks, although the number of devices required increases considerably.

Hybrid networks cannot be classified as direct or indirect networks because their main characteristic is the use of elements of both types. The intention is to use the best that both direct and indirect networks offer to create a more efficient network. The objective is to create a network that, as the number of nodes increases, the devices needed to connect the network do not grow exponentially, as happens in indirect networks, and the distance between nodes does not grow as much as in direct networks.
2.2 The KNS network topology

The KNS hybrid topology combines a direct network of \( n \) dimensions with small indirect subnetworks connecting the nodes of each dimension. By combining the advantages of direct and indirect networks, this network is able to connect nodes with low latency, higher bandwidth, and higher fault tolerance, but at a lower cost than indirect networks. In this network, the routers provide connection to the nodes to the rest of the nodes as it is in direct networks, but this router is not connected to the router of another node; it is connected to a small indirect network that provides connection to all the nodes of a dimension. A router provides connectivity to \( n \) indirect networks, each of which connects the nodes of each dimension.

Considering the above, this network can be defined with 3 parameters: the number of nodes per dimension \( (k) \), the number of dimensions \( (n) \), and the number of stages of the indirect network connecting the nodes of each dimension \( (s) \). Note that the number of dimensions \( n \) and the number of nodes per dimension \( k \) are the same parameters as for direct topologies. The number of nodes in the KNS topology is given by \( N = k^n \). Moreover, the number of stages of the indirect network is represented by \( s \), which depends on the number of switch ports \( (d) \) and the \( k \) nodes in each dimension. If \( k \leq d \), then one switch would be sufficient for the indirect network, so \( s = 1 \). By contrast, if \( k > d \), it will be necessary to interconnect the \( k \) nodes per dimension with an indirect network (e.g., a Clos or Fat tree), whose number of stages \( (s) \) will be given by \( \log_d K \).

Note that the name KNS given to this topology is based on the \( k \), \( n \), and \( s \) parameters. Another way of referring to this hybrid topology is \( k \)-ary \( n \)-direct \( s \)-indirect. Table 1 summarizes these parameters, and Figure 1 shows an example of a 16-node 4-ary 2-direct 1-indirect KNS network. For the sake of simplicity, we only show the nodes on the corner routers.

Table 1: KNS Parameters

<table>
<thead>
<tr>
<th>KNS Topology parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( k )</td>
<td>Number of nodes per dimension</td>
</tr>
<tr>
<td>( n )</td>
<td>Number of dimensions</td>
</tr>
<tr>
<td>( s )</td>
<td>Number of stages indirect network</td>
</tr>
<tr>
<td>( d )</td>
<td>Number of ports in switches</td>
</tr>
</tbody>
</table>

Regarding the routing algorithms for KNS topologies, it is proposed an adaptation of the dimension-order routing (DOR) algorithm used in meshes and tori [7]. The main idea behind DOR routing is that packets traverse the network in the order given by the available dimensions, first the X dimension, then the Y dimension, etc. Thus, it is guaranteed that the minimum path is used between two nodes and no deadlocks can occur. Specifically, the hybrid-DOR routing defined for KNS topologies defines that switches and routers are identified by indicating their position in the network. Routers are labeled as indirect networks, with a set of coordinates \( [r_{n_k-1}, r_{n_k-2}, ..., r_1, r_0] \) (as many coordinates as there are dimensions). Each coordinate represents the position of the router in each dimension. To identify the switches, two parameters \([d, p]\) will be used, where \( d \) is the dimension in which the switch is located and \( p \) is the position of that switch in dimension \( d \).
As we can see in Figure 1, we assume that switches only interconnect the nodes corresponding to the same dimension, so when they have to forward a packet, they send it to the router whose $r_d$, being $r_d$ the dimension in which the switch is located, matches the $r_d$ of the destination. If a router compares all the coordinates and they all match, the destination must be a node directly connected to it. If the indirect network is formed by more than one switch, it can be built using a Fat-tree topology, the implementation of the routing algorithm is a bit more complex, but it can be used with some of the existing ones.

The Hybrid-DOR pseudo-code for k-ary n-direct 1-direct topologies is shown below:

Algorithm 1 Hybrid-DOR for Routers

1: $i = 0$
2: $Done = false$
3: while ($i < n_h$) \& \& $!Done$ do
4: \hspace{1em} if $x_i \neq r_i$ then \hspace{1em} $Done = True$
5: \hspace{2em} \hspace{1em} link = i
6: \hspace{1em} end if
7: \hspace{1em} $i = i + 1$
8: end while

Algorithm 2 Hybrid-DOR for Switches

\hspace{1em} link = $x_d$

According to this definition of the KNS topology, a new type of network is obtained, which allows scaling to a large number of nodes at a cost not as high as the indirect topologies and without losing the efficiency that is lost when increasing the number of nodes in the direct ones. Instead of traversing each dimension node by node, as in direct
networks until reaching the destination, in KNS all the nodes of each dimension have a direct connection through an indirect network, which reduces the average distance of the network. The cost does not increase as much as in traditional indirect networks, since the networks needed to connect the nodes of each dimension are simple and do not require many switches and cables to be implemented.

So far, there is no known implementation of a KNS network in a real system using InfiniBand. The objective of this work is to perform a real implementation of a KNS network to evaluate its real performance.

2.3 Infiniband

InfiniBand is one of the most widely used and efficient interconnect technologies for HPC systems. The InfiniBand Architecture (IBA) is detailed in the InfiniBand specification [8], which allows InfiniBand technology to be used to build both small networks and networks with a large number of nodes. The main advantage over other technologies is that it provides higher bandwidth per link and reduces the CPU load on the processing nodes using the Remote Direct Memory Access (RDMA) transport protocol, thereby improving communication latency.

The InfiniBand architecture (IBA) uses Host Channel Adapter (HCA) network cards, switches and links to interconnect them. Moreover, IBA-based networks require a software entity, called the subnet manager, to identify all the network devices and orchestrate the communication among them. Specifically, the HCAs are responsible for connecting the physical network link to the internal bus of the processing node. Within the nodes, and at the application layer, the IBA provides the verbs, a software API that allows the applications to use specific procedures to directly access the HCAs and perform communication operations without the intervention of the CPU and OS kernel. Note that the support for transport protocols on the HCAs, such as RDMA, is also handled by the verbs. On the other hand, the switches are responsible for forwarding packets from one port to another. Each switch has a Linear Forwarding Table (LFT) with as many entries as there are destinations on the network. Each destination in IBA-based networks is identified by a local identifier (LID), which is a 16-bit address assigned to each device by the subnet manager at the network setup stage. Each input to an LFT consists of a destination LID and the output port through which the switch must route a specific packet to reach its destination.

One of IBA’s most important elements is the subnet manager (SM), which runs on only one node in the network, although there are switches capable of running it. The main functions of the SM are to discover the network topology, identify each network device (HCAs and switches) with a specific LID, and populate each device’s LFTs. Moreover, each network device runs a subnet agent (SA), which receives the information generated by the SM through the subnet manager agent (SMA). In other words, the SM is responsible for calculating the routing by populating the LFTs of each device in a centralized manner in the node where it is executed and sending it to each device. The port assigned to each destination LID depends on the routing algorithm given to the SM when it is initialized.

Furthermore, the IBA specification defines an API of protocols and services bundled in a middleware between user applications and IBA hardware. Note that the IBA
specification does not say how to implement the API but rather defines the functionality that InfiniBand devices must support. These functionalities, referred to as verbs, represent actions solicited by applications. The collection of verbs defines the range of interactions that an application can have with the IBA hardware. These functionalities are provided in the OpenFabrics Software (OFS), developed by the OpenFabrics Alliance (OFA). The IBA OFS comprises open-source implementations for the SM, the InfiniBand drivers, and the libraries necessary for communication types supported by IBA, such as RDMA and MPI. The Open Subnet Manager (OpenSM), provided by OFS, is also open source and it offers different routing algorithms or routing engines, such as upd [9], ftree [10], dor [11], lash [12], torus-2QoS [9], SSSP [13], DFSSSP [14], and minhop [9]. The OpenSM architecture permits us to implement other routing algorithms that are better suited to other topologies.

3 Description of the KNS implementation

In this section, we describe the different steps carried out to implement the KNS topology in our cluster. First, we describe the hardware and software infrastructure that we have used. Next, we provide some details of the implementation of the Hybrid DOR routing. Finally, we describe the cabling and layout for the KNS topology in the CELLIA cluster.

3.1 Hardware/Software infrastructure

The hardware and software infrastructure used in this work includes the OFS, which offers the OpenSM source code. Additionally, we have used the simulation tool ibsim to emulate the control traffic generated in the setup phase of Infiniband networks by the subnet manager to discover and identify the network devices. Moreover, we use the CELLIA cluster, which uses InfiniBand technology for the interconnection network. Figure 2 illustrates a general diagram of this work’s hardware and software infrastructure.

We have implemented the Hybrid DOR routing algorithm for KNS topologies into the OpenSM control software. During the code development stage, we simulated the
control traffic of an InfiniBand network utilizing the ibsim simulator, which is an open-source program that enables us to execute OpenSM and emulate its behavior without an actual InfiniBand network. This permits us to modify OpenSM by creating new features and examining them prior to testing on real hardware. The ibsim simulator requires a topology description file that defines an InfiniBand network, listing all the devices that compose it (HCAs and switches) and the connection pattern that each device’s ports follow. To debug possible network failures, we have used the IB tools, which are also provided by the OFS and the InfiniBand drivers installed in the CELLIA cluster. Specifically, we have used ibnetdiscover, which displays the list of InfiniBand network devices and their connection patterns in the same format used by the ibsim simulator. Note that this approach for the hardware/software infrastructure separates the software development from the hardware configuration. Therefore, the routing engines of IBA-based networks can be implemented and validated without requiring large hardware installations.

Finally, we wired a KNS network on the CELLIA cluster and launched openSM with our Hybrid-DOR implementation to check that it runs correctly. We have also evaluated the performance of this topology by running different HPC applications and benchmarks, comparing the results obtained on the KNS with our implementation of Hybrid-DOR and other routing engines available in OpenSM used by other topologies, such as Fat trees, which we describe below.

3.2 Hybrid-DOR implementation in OpenSM

As we have described before, the first step for all the routing engines in InfiniBand-based networks is to discover the topology. The OpenSM performs a broadcast of management datagrams (MADs), which are routed through the network without any knowledge, using all the physical paths available in the topology. When a random MAD reaches a specific device (HCA or switch), that device responds directly to the endpoint running OpenSM and communicates its assigned list. This initial communication permits OpenSM to address all the network HCAs and switches and assign them with a unique LID. Moreover, this network discovery phase also allows OpenSM to identify the physical connections between the different network devices and build a logical view of the network topology. Note that OpenSM is running in a single computing node of the network, so this node will initiate and receive all the control traffic (i.e., MADs) during the discovery phase.

Once the OpenSM discovers the topology and creates the logical network view, the next step is to perform the placement of the different LIDs in the different dimensions of the KNS topology, and differentiate between the router and switch LIDs. For the sake of simplicity, we assume hereafter a 2-dimensional (2D) KNS topology, which uses a single switch in the indirect network, i.e., a $k$-ary 2-direct 1-indirect KNS topology.

To perform the placement of the HCAs in the KNS topology, we need to specify a coordinate for each HCA LID in the same way as these coordinates are set for direct networks. Therefore, in a 2D KNS topology, each one of the HCAs has a $[x, y]$ coordinate. Moreover, we need to identify which routers are directly connected to the

---

[1] For instance, if we use 64-port switches, the number of nodes that can be interconnected by a 64-ary 2-direct 1-indirect KNS is $N = 64^2 = 4096$. 

10
HCAs and the different switches that are connected to each router at each one of the dimensions. Therefore, to identify each device type, the Hybrid-DOR routing engine examines all the devices that are not HCAs (i.e., switches or routers). For each one of them, it examines all the devices that are reachable from each one of the ports. If there is at least one HCA that is connected to one of the ports of the current device, that device is considered a router. Note that in the KNS topology, routers interconnect the HCAs with the switches (see Figure 1). By contrast, if some of the connected devices to the current device are other switches acting as routers or other switches, the current device is identified as a switch that is part of the indirect network.

After the switches and routers have been identified, the Hybrid-DOR routing engine assigns the corresponding coordinates to each device (i.e., HCAs, routers, and switches). To accomplish this, the Hybrid-DOR goes through the list of devices already identified in the network. First, the indirect network’s first switch is assigned the coordinates [0,0], i.e., the switch 0 for both the X- and the Y-dimension. Moreover, this switch assigns coordinates to each router connected to it in ascending order ([0,0], [0,1], ..., [0,k − 1]) through all its ports. Note that, as we assume a 2D KNS topology, all the ports for a given switch will be connected to routers, and the routers will share the same coordinates with the HCAs they are connected to. When a router receives its coordinates from the switch in the X-dimension, it assigns the coordinates to the other switch connected to that router in the Y-dimension. That switch preserves the Y-coordinate received by the router and replaces the X-coordinate value with the Y-coordinate. For instance, if the connected router is at [0,3], the switch in the Y-dimension will be located at [1,3]. Once all the switches of dimension Y have coordinates assigned, the rest of the switches in the X dimension can be assigned coordinates. Hybrid-DOR searches for switches without coordinates in the device list and assigns them coordinates in ascending order. To complete the identification of devices, Hybrid-DOR searches in the list of devices for routers without coordinates and assigns coordinates based on the switches to which they are connected. For instance, if a router is connected to the switch [0,4] and switch [1,2], the coordinates for the router will be [4,2]. This method of identifying devices offers flexibility in wiring, as a specific port order is not required, thus preventing potential wiring mistakes.

When the Hybrid-DOR routing engine has identified all the devices, the next step is to fill the LFT tables at switches and routers. Note that OpenSM is in charge of populating the LFTs at its logical view of the topology. Later, when all the LFTs are populated, the OpenSM will send this information to the switches. As we have described before, each one of the switches and routers has an LFT table containing as many entries as LIDs exist in the network. For each one of these LIDs at a given switch or router, the Hybrid-DOR routing engine calculates the output ports for all the network LIDs. Specifically, for each LID Hybrid-DOR checks if the device associated with that LID is directly connected to the current switch or router. If so, it assigns the port connecting to that device to that LID in the LFT. Otherwise, if that LID corresponds to a device that is not directly connected to the current switch or router, then its coordinates are compared to those of the current device. If the not-connected device is an HCA, its coordinates are the same as the router to which it is connected and the routing engine selects the port of the router that connects to the HCA.
By contrast, routers check, following the dimension order, if the coordinates of the source device differ from the destination coordinates. If the coordinates in a dimension do not match, the port associated with that LID will be the port that connects to a switch that connects the routers of that dimension. In the case of switches, the port for the target LID will be the one that connects to the router whose coordinate of the dimension in which the switch works (X or Y) matches that of the target device in that dimension.

Finally, when the Hybrid-DOR routing engine populates the LFTs in the logical view of the KNS topology, the OpenSM will send MADs directly to each one of the switches and routers and will fill the routing tables. After the LFTs have been filled in the physical devices, the setup phase has finished and the network is ready to receive traffic from applications.

3.3 Wiring scheme and layout

Figure 3 shows a diagram of the 36-node 2D KNS topology (i.e., 6-ary 2-direct 1-indirect) that we have built in the CELLIA cluster. Note that the routers are depicted using circles (colored in orange), and their disposition is like in a 2D mesh (i.e., rows and columns), so each one of the routers has a \([x, y]\) coordinate. Note that each router is connected to an HCA (only the four HCAs in the corners are shown, while the rest are committed for clarity). Moreover, switches are illustrated using rectangles colored blue. Note that switches, routers and HCAs have been labeled using the coordinates as described in the previous section.

![36-node topology KNS built on CELLIA cluster.](image-url)
The CELLIA cluster is composed of 50 server nodes HP Proliant DL120 Gen9, each having an Intel Xeon E5-2630v3 8-core 1.80GHz processor and 32GB of RAM. Each one of these nodes is equipped with an InfiniBand HCA with QDR technology, providing 40 Gbps of link speed. The cluster is also equipped with 50 Mellanox™ IS5022 8-port switches with InfiniBand QDR technology, providing 40 Gbps of bandwidth per port. We have used copper cables and QSFP transceivers from Mellanox™ compatible with QDR technology. All these devices, together with the administrative network are racked in three racks, two for the nodes and one for the switches. The switch rack contains the 50 switches, which can be accessed for cabling at the front and the back.

Given the physical disposition of the racks, we have analyzed the most proper layout with the objective of maximizing the utilization of the shortest-distance cables and avoiding using lengthy ones. We have defined six groups of routers in the KNS topology (see Figure 3). The routers of each group will be placed closely to each other in the switches rack. Figure 4 illustrates the layout of the switches and routers in the switches rack both for the front and the back sides.

As we can see, there are six groups of routers, each with the same Y coordinate. On the front side, there are four groups of six routers each (from G0 to G3), connected by four switches in the X dimension. There are also six switches in the Y dimensions placed on the front side. On the back side, there are two groups of six routers (G4 and G5), connected by two X-dimension switches. This layout ensures the use of shorter-distance cables, as we describe below.

Regarding the cabling, Figure 5 illustrates the cabling for the X dimension. As we can see, the six routers of each group are connected with the corresponding switch in the X dimension. For instance, routers R00, R01, R02, R03, R04, R05 are connected using short 1-meter copper cables to switch S00.

![Fig. 4: Switches and routers layout in the CELLIA rack.](image-url)
On the other hand, Figure 6 illustrates the cabling performed in the switches and routers rack for the Y dimension. Note that we want to minimize the number of 3-meter cables (the ones colored in green) going from the back to the front side of the rack.

Finally, note that 2D KNS topologies could be built with fewer hardware components if the HCAs offered routing functionality so that traffic could be forwarded from one HCA port to the other without the host intervention. As far as we know the InfiniBand hardware does not provide this functionality out-of-the-box. However, we can still use the routers to improve connectivity so more HCAs could be connected to each router and parallel links could be used between switches and routers to increase the bandwidth.
4 Experiments and results

This section evaluates the performance of the 2D KNS topology described in section 2.2, which has been implemented in the CELLIA cluster. For this evaluation, we have conducted a set of experiments using well-known HPC benchmarks. In the following sections, we describe in more detail the experiments performed, and the configuration parameters of these experiments. Subsequently, we analyze the experiments’ results.

4.1 Benchmarks set-up

As we have described above, this section describes the benchmarks that we have compiled and executed in CELLIA to evaluate the performance of the 2D KNS topology. For this study, we have selected some of the most relevant benchmarks in HPC based on the Message Passing Interface (MPI) paradigm.

4.1.1 HPCC

HPC Challenge (HPCC) [15] is a collection of benchmarks that measure computing performance along with a large quantity of memory access patterns. Essentially, HPCC comprises the following seven tests:

- **HPL.** It is the Linpack [16] benchmark, which performs floating point operations solving a series of equations systems.
- **DGEMM.** It performs multiplications of matrices of real numbers and returns the number of FLOPs that have been achieved.
- **STREAM.** This benchmark is utilized to gauge the sustainable memory bandwidth (in GB/s) and associated compute rate for a simple vector kernel.
- **Parallel matrix transpose (PTRANS):** It is a valuable tool for testing the overall communications capacity of the network.
- **RandomAccess.** It measures the speed of random integer updates in memory. It returns the result in Giga Updates Per Second (GUPS). This program chooses a random position in memory, reads an integer, modifies it in the processor, and writes it back.
- **FFT.** It measures the FLOPs of the systems using the Discrete Fourier Transform (DFT).
- **b_eff:** It gets the communication latency and bandwidth of different communication patterns that are run simultaneously.

4.1.2 HPCG

The High-Performance Conjugate Gradient (HPCG) [17] benchmark project aims to establish a new metric for ranking HPC systems, in addition to the current Linpack rankings used for the Top500 list. The HPCG includes computational and data access patterns designed to more closely resemble real-world problems, encouraging computer system designers to explore how to improve performance with patterns closer to reality.
4.1.3 Graph500

The Graph500 [18] benchmark focuses on data-intensive applications, which have become very important for business analytics, finance, and all those fields where a high number of data needs to be handled, such as deep learning and big data applications. This benchmark has five types of problems depending on the size of the input; it can range from 17 GB to 1.1 PB. The benchmarks’ tests basically involve creating a graph to run typical graph operations to evaluate the performance of the system.

4.1.4 Netgauge

Netgauge [19] is a benchmark focused on evaluating the performance of high-performance system networks. Although it supports different communication patterns, the most representative ones are the following:

- 1toN. A randomly chosen node sends data to the other nodes, which send back the information received.
- Nto1. All nodes in the network send information to a randomly chosen node and like the 1toN pattern, this node sends back the information received.
- Effective Bisection Bandwidth (EBB). It generates different bisections in the network and generates traffic to calculate the bisection bandwidth in each one and then calculate the average.

4.2 Experiments setup

As stated previously, we have used the CELLIA cluster to conduct the experiments to evaluate the performance of the 2D KNS topology, when compared to a Fat-tree topology. Specifically, apart from the KNS topology, we have built a slimmed fat-tree topology (SFT). Figure 7 shows the SFT configuration. We can see that 36 nodes are indirectly connected by 21 switches distributed over three stages. There are 3 groups of 12 compute nodes each, each group connecting 3 subgroups of 4 server nodes. Note that the stages 1 and 2 of the SFT topology have more switches and links (and so more bandwidth) than the stage 3, which improves the communication speed of close server nodes. In the upper stage, there are parallel links between each switch to increase bandwidth in that part of the network.

Fig. 7: 36-node topology SFT built on CELLIA cluster.
Furthermore, in order to evaluate the performance of the KNS and SFT topologies, we conducted tests by configuring different routing engines in OpenSM to determine which algorithm best suits each topology. We used the \texttt{lash}, \texttt{updn}, and \texttt{dfsosp} routing engines, which work on any topology. Besides, for the KNS topology, we have also incorporated the \texttt{hdor} routing engine, our implementation of Hybrid-DOR in OpenSM, and for the SFT topology, we employed the \texttt{ftree} routing engine, designed especially for this kind of indirect topologies. We have executed each benchmark in both topologies with each of the previously mentioned routing engines. For each benchmark and routing engine, we executed the benchmarks by distributing the MPI tasks linearly and randomly across the nodes, to observe how this factor affects the performance obtained in each experiment.

4.3 Results analysis

In this section, we examine the results of the experiments obtained for the HPCC, HPCG, Graph500, and Netgauge benchmarks. Each graph follows the same structure. Each graph shows both the results obtained when the MPI tasks are distributed linearly (on the left) and randomly (on the right). For each metric, the left graph displays the results achieved for KNS, while the results for SFT are presented on the right.

4.3.1 HPCC

We will begin with an analysis of the HPCC benchmark results. Firstly, we present the results obtained with the PingPong test, which consists of sending many pings from an origin node to a destination node in order to calculate the maximum, minimum, and mean for the latency and the bandwidth metrics.

Figure 8 shows the plots corresponding to the bandwidth in the PingPong test. The maximum bandwidth values (Figure 8a and Figure 8b) are very similar for the two topologies and all routing algorithms, standing at almost 4 GB/s. This makes sense since the CELLIA cluster links go up to 40 Gbps (i.e. 5 GB/s) but for the 8B/10B coding protocol, the effective bandwidth is 32 Gbps (4 GB/s). From the minimum bandwidth values shown in Figures 8c and 8d, it is evident that the KNS topology outperforms the SFT topology when MPI tasks are randomly mapped onto the nodes. In KNS, the bandwidth remains almost constant regardless of the routing engine used, whereas, in SFT, it decreases by almost 2 GB/s in the case of the \texttt{updn} algorithm. For the KNS, the maximum and minimum values are quite similar, and therefore the average bandwidth (Figure 8e) is close to the maximum, giving better average results than SFT (Figure 8f), which drops a little for some routing engines. It should be noted that the HPCC benchmark, despite being frequently employed, this traffic pattern PinPong introduces a light load on the topology.

Figure 9 shows the average, maximum, and minimum results the latency in the PingPong test. The results are similar to those of the bandwidth: for the maximum values of such latency (9a and 9b), there is a case in the SFT topology where it reaches up to 14 microseconds and for the random task mapping it also gets worse by 1 microsecond, while in the KNS the results are the same for both mappings. For the minimum latency values (9a and 9b) it is true that the SFT offers better results, due
to the minimum distance being two hops and in the KNS it is 4 hops. It is important to highlight that these 4 hops, 2 of them are made in routers, whose latency in a KNS using HCAs with routing function, would be much lower than in this configuration. As for the average latency (see Figures 9e and 9f), similar results have been obtained for the two topologies and all the routing used, although it gets a little worse when mapping the MPI tasks on the nodes randomly in the SFT.

Continuing with the HPCC benchmark, let’s look at the results of the MPI Random Accesses test. Figure 10 shows the results of evaluating random memory accesses using MPI. The unit of measurement is GUPs (Giga Updates Per Second) and indicates the number of integers the system can write in one second. In the linear distribution of MPI tasks on the nodes we achieve about 0.05 GUPS for KNS and for SFT (slightly more for KNS). But when these tasks are randomly distributed on SFT the performance decreases compared with the KNS, which is maintained and even improves with the hedor routing engine.

In Figures 11 and 12, we can see more latency-related results, in this case using the ring communication pattern. This pattern distributes the processes in a ring structure, where each process only receives and sends data to the processes on its left and right. The processes are distributed in a natural manner or randomly. We obtained comparable results to previous cases. When MPI tasks are distributed linearly on nodes, both KNS and SFT obtain similar results. By contrast, when the MPI tasks are randomly distributed among the nodes, the latency in the SFT increases significantly. As discussed above, the latency in the KNS is higher than it should be in an implementation using HCAs with router functions, since the minimum path is 4 hops on 4 switches, rather than 2 routers and 2 switches.

Concluding with the HPCC benchmark results, let’s take a look at the results obtained in the PRTRANS test. Figure 13 shows bandwidth results and Figure 14 shows latency results. As has been the case during the previous tests, the KNS obtains better results than the SFT topology, both bandwidth and latency, especially when MPI tasks are assigned to nodes in a random rather than linear fashion.
Fig. 8: HPCC: PinPong bandwidth results.
Fig. 9: HPCC: PingPong latency results.
Fig. 10: HPCC: MPI Random Access results.

Fig. 11: HPCC: Naturally ordered ring latency.

Fig. 12: HPCC: Randomly ordered ring latency.
Fig. 13: HPCC: Bandwidth results in PTRANS test.

Fig. 14: HPCC: Execution time in seconds for PTRANS test.
4.3.2 HPCG

In this section, we will analyze the results obtained with the HPCG benchmark. Specifically, we will compare the execution time and the GFLOPS (10^9 floating point operations per second) achieved by each execution. In this case, a curious result has been observed: when MPI tasks are assigned in random order, better results are obtained than when they are linearly distributed (see Figure 15) in the KNS. Generally, the KNS obtains the results in less time than the SFT.

![Fig. 15: HPCG: Execution time for the benchmark.](image)

Figure 16 shows the computational performance of the CELLIA cluster, measured in GFLOPS. Both topologies reach a peak of nearly 30 GFLOPS. Notably, the routing engine hdor performs best on the KNS topology (see Figure 16a), likely due to its specific development for this topology.

![Fig. 16: HPCG: GFLOPS](image)
4.3.3 Graph500

This section compares the performance obtained by the Graph500 benchmark in each topology. The focus is on analyzing the number of Traversed Edges Per Second (TEPS) obtained in each execution, a metric specific to this benchmark and related to the number of floating operations per second that can be performed (FLOPS in LINPACK) and the number of operations with memory (GUPS of HPCC). For a linear distribution of MPI tasks across the nodes, the TEPS values remain the same for the KNS and SFT topologies (see Figure 17), staying close to 2.5 GTEPS. However, for a random distribution of tasks across the nodes, the performance slightly decreases except for hdor routing engine in KNS topology, which remains constant at 2.5 GTEPS (refer to Figure 17a).

![Graph500: Traversed Edges Per Second (TEPS).](image)

(a) TEPS in KNS.  
(b) TEPS in SFT.

Fig. 17: Graph500: Traversed Edges Per Second (TEPS).

4.3.4 Netgauge

We will conclude the analysis of benchmark results with the benchmark Netgauge. It offers various traffic patterns, including Effective Bisection Bandwidth (EBB). In Figure 18, it can be observed that this value is significantly higher for KNS than for SFT. KNS reaches almost 2 GB/s, while SFT’s maximum is 1.5 GB/s. Additionally, hdor improves the bisection bandwidth for both distributions of the MPI tasks.

4.4 Complexity analysis

To conclude the comparison of both topologies, we will analyze the complexity of building each. We will compare the resources used in each topology. Table 2 presents a summary of the resources employed in these topologies. The final column shows the resources used to configure the same KNS utilizing HCAs instead of switches as routers to perform the same function.

It takes 21 switches to build an SFT network like the one we have used. In contrast, just 12 switches are requisite for the KNS, almost half the amount. The configured KNS
needs an extra 36 switches that perform the router function, thereby augmenting the number of connections. If HCAs with the router function were available, only switches for indirect networks connecting nodes in each dimension would be required. This would also decrease the hops between nodes, as intermediate routers between the HCAs and indirect network switches would not be necessary. This would lead to improved performance, as latency would be lower. Based on this analysis, the KNS network offers better results than SFT at a lower cost if HCAs with routing functionality are available. Furthermore, as previously stated, excluding the switches that perform the routing function would result in lower latency and improved performance.

### 4.5 Experiments conclusions

To summarize the results analysis, it is relevant to note that latency in the KNS topology tests is higher when switches are used as routers. However, using HCAs with traffic forwarding and routing capabilities would reduce latency. The KNS results demonstrate that hdor functions well as a routing engine in our KNS topology configuration. While various routing engines may effectively work in the KNS topology, hdor is the most effective. The benefits of hdor are clear; it provides better results than the other routing engines, demonstrating that this algorithm improves the performance of a KNS. In comparison to the SFT topology, we have observed that some tests show similar results, especially when using the same routing algorithms (updn, lash, and
The ftree routing engine gives better results in the SFT topology. Despite this, KNS with hdor typically gives better results than SFT with ftree. Therefore, we can conclude that using hdor in a KNS topology improves the performance of the SFT topology with comparable attributes, regardless of the routing engine used.

5 Conclusions

With this study, it can be concluded that InfiniBand is among the most extensively implemented technologies for HPC systems. According to the most recent Top500 list, 40% of supercomputers use InfiniBand as their interconnect technology. As mentioned in this paper, the interconnect network is one of the main points for improvement in HPC systems. The interconnection network must not be a bottleneck, hindering nodes from reaching their maximum computational capacity and becoming idle when messages cannot be sent or received due to network congestion. Efficient topologies and routing algorithms can be used to improve network performance. The KNS topology examined and utilized in this study is relevant to the present and future of interconnection networks because of its particular characteristics, which we have seen explained in section 2.2. After implementation in a real cluster and testing with standard benchmarks, it can be concluded that this topology operates efficiently. Comparatively, it produces significantly better results than the SFT topology, particularly when utilizing the Hybrid-DOR routing algorithm. The results obtained are promising, considering that better results could be obtained by applying queuing schemes and congestion control, which are used by other routing algorithms. Moreover, the implementation of the Hybrid-DOR routing algorithm in OpenSM is relatively straightforward. On the other hand, the complexity of KNS in terms of the resources required for its physical implementation can be a problem if there are no InfiniBand HCAs that can act as routers since using switches for this function is expensive and a waste of ports. However, if HCAs with router capability are available, the KNS topology utilizes resources very efficiently, as demonstrated in the complexity analysis.

Declarations

• Funding: This work has been jointly funded by the BBVA foundation and Becas Leonardo (call 2020) with the grant IN[20], TIC, TIC_0042, by the Ministry of Science, Innovation and Universities and the European Commission (FEDER funds) under the project RTI2018-098156-B-C52 (MCIU/FEDER), by the Junta de Comunidades de Castilla-La Mancha under the project SBPLY/17/180501/000498, and by the Universidad de Castilla-La Mancha under the project 2023-GRIN-34056.

References


