Optimized Routing for Fat-Tree Topologies

Bartosz Bogdański

Thesis submitted for the degree of Philosophiae Doctor
Department of Informatics
Faculty of Mathematics and Natural Sciences
University of Oslo
January 2014

Series of dissertations submitted to the
Faculty of Mathematics and Natural Sciences, University of Oslo
No. 1521

ISSN 1501-7710

Cover: Inger Sandved Anfinsen.
Printed in Norway: AIT Oslo AS.

Produced in co-operation with Akademika Publishing.
The thesis is produced by Akademika Publishing merely in connection with the thesis defence. Kindly direct all inquiries regarding the thesis to the copyright holder or the unit which grants the doctorate.

To my family

Abstract

In recent years, InfiniBand has become one of the leading interconnects for high-performance systems. InfiniBand is not only the most popular interconnect used in the fastest, largest and most expensive supercomputers in the world, but it has also entered the enterprise market and today it can be found in thousands of datacenters, in database systems, in financial institutions dealing with high-frequency trading, and even in web mapping services where picture data is stitched together to form the map we see in a web browser.

Many of the InfiniBand systems are built using the fat-tree topology. The fat-tree routing algorithms that are used to distribute the paths in a fat-tree have been scrutinized in recent years and there were multiple efforts that aimed at improving the network performance. This thesis, as a collection of six research papers, contributes to various routing aspects that concern fat-tree routing for InfiniBand. The research work presented here is strongly influenced by requirements of enterprise systems, which are significantly smaller than supercomputers, have space, energy and fixed-cabling limitations and require an integrated approach that combines multiple components so the system is robust, scalable and responsive.

Current fat-tree routing algorithms lack several features that are required by an optimized enterprise system. In Papers I-IV and Paper VI we propose methods that improve the properties of the fat-tree routing algorithm. The first, and most important property of an interconnection routing algorithm is performance. In Paper I we study the problem of non-optimal routing in topologies where switch ports are not fully populated with end-nodes and we propose a solution that alleviates the counter-intuitive performance drop for such fabrics. Paper II goes further and proposes a cheap alternative to congestion control. By using multiple virtual lanes, we are able to remove head-of-line blocking and parking lot problems in fat-tree fabrics, which significantly improves the performance when multiple hotspots are present in the network. Paper III focuses on

route reachability, which is another major property of a routing algorithm. We formally prove that full reachability between all devices can be accomplished in any regular fat-tree without any deadlock concern and we present an algorithm that achieves full reachability between any pair of devices in a fat-tree fabric. The last routing property we analyze is fault tolerance, which we discuss in Paper IV and VI. The former paper is a study of four major routing algorithms where we compare their routing performance on various irregular fat-trees. In this paper, we also propose and analyze methods to make the discovery algorithm for the fat-tree routing more fault-tolerant and scalable. Paper VI presents a multihomed routing algorithm that makes sure that no single point of failure exists in a fabric for a node that has more than one port.

When InfiniBand was standardized, several major features were missing. One of such crucial features is the layer-3 routing or IB-IB routing between IB subnets. Currently, there is very little support for such technology, but the limits imposed on local addressing space, inability to logically segment fabrics, long reconfiguration times for large fabrics in case of faults, and, finally, performance issues when interconnecting large clusters, have rekindled the industry's interest into layer-3 routing. In Paper V we examine the layer-3 routing problems in InfiniBand and we introduce two new routing algorithms for inter-subnet IB routing. We show that the features provided by the fat-tree topology make it an excellent choice for the underlying logical backbone of the fabric and the two routing algorithms reuse the original concepts fat-tree routing is based upon.

Acknowledgements

First and foremost, I wish to express my heartfelt gratitude to my supervisors, Sven-Arne Reinemo, Tor Skeie, and Olav Lysne who gave me encouragement, constructive feedback, and guidance through the whole duration of my studies. Finishing this journey would not be possible without you.

I am also grateful to my friends and colleagues at Simula Research Laboratory for making it an attractive place to do research. I am especially indebted to Wei Lin Guay, Ernst Gunnar Gran and Frank Olaf Sem-Jacobsen, who taught me so many useful things.

I would also like to thank my colleagues from Oracle Corporation, Bjørn Dag Johnsen, Line Holen, Lars Paul Huse, Ola Tørudbakken and Jørn Raastad who supported me during my PhD with their excellent technical skills and motivated me in difficult moments.

I would not have contemplated this journey if not for my parents, Bożenna Bogdańska and Bogdan Bogdański, who infused me with a love of science and creative pursuits. To my parents, thank you.

Finally, I would like to thank my wife, Beata, for her neverending patience, love and support, and my little son Jan Krzysztof, for his bright and radiant smile.

Table of Contents

List of Figures

Chapter 1

Introduction

The term *Super Computing* was first used in 1929[1], however, it was not until the 1960s that real supercomputers were introduced. First supercomputers consisted of relatively few custom-built processors, but today, massively parallel supercomputers equipped with tens of thousands of commercial off-the-shelf processors, are a standard. This evolution is best shown by comparing the performance of the Cray Titan supercomputer, currently one of the fastest supercomputers in the world (17.6 PetaFLOPS), with the CDC 6600, the first machine considered to be a supercomputer (1 MegaFLOP) - the Titan is $1.76 * 10^{10}$ faster than the CDC 6600. For comparison purposes, this is also the difference between the land speed of a garden snail (0.017 $^{m}/_{s}$) and the speed of light in a vacuum (299 792 458 $^{m}/_{s}$).

However, it is not only the number and the frequency of the processors that makes today's supercomputers so immensely powerful. One of the most important aspects of every high-performance system is the interconnect [4], that is, the network that connects the whole system together. Over the years, various specialized topologies have been invented, and with this, numerous routing algorithms were designed to forward the traffic within the system in the most efficient manner.

With the advent of cloud computing and big data, today's high-performance systems are facing new complex multi-layered challenges. Routing is only a

[1] *New York World* describes the Columbia Difference Tabulator: *New statistical machines with the mental power of 100 skilled mathematicians in solving even highly complex algebraic problems were demonstrated yesterday for the first time...* in an article entitled: *Super Computing Machines Shown* [3].

small subset of the problem, however, improvement here leads to large gains system-wide because it may solve scalability, reliability, security and performance limitations. Needless to say, successfully addressing these limitations has always been on the priority lists of several public and private institutions.

The work presented in this thesis, as a collection of research papers [5–10], aims at contributing to various aspects of the fat-tree routing algorithm and inter-subnet routing (layer-3 routing) in InfiniBand. The remainder of this chapter will serve to present a brief description of published works and applied research methods. Chapter 2 gives a brief description of routing in InfiniBand while Chapter 3 presents the motivation and summary of each paper. In Chapter 4, we will discuss future directions and conclude with final remarks. Lastly, in Appendix A the papers are presented.

1.1 Context of the Work

In this work, we will focus on the *fat-tree topology* and fat-tree *routing algorithms*. The fat-tree topology is one of the most common topologies for high performance computing clusters today, and for clusters based on InfiniBand (IB) technology the fat-tree is the dominating topology. Not only does this include high-profile installations from the Top 500 list [11] like Nebulae/Dawning, TGCC Curie or SuperMUC, but also numerous smaller enterprise IB clusters like Oracle's Exadata Database Machine and Exalogic Elastic Cloud systems.

For fat-trees, as with most other network topologies, the routing algorithm is crucial for efficient use of the underlying topology. The routing algorithm dictates how to select a path in the network along which to send the traffic. Because of the rising popularity of the fat-tree topologies in the last decade, there were many efforts trying to improve the fat-tree routing algorithms. This includes the current approach that the OpenFabrics Enterprise Distribution [12], the de facto standard for InfiniBand system software, is based on [13, 14]. Despite these numerous efforts, several issues remained unresolved and addressing them was the driving force behind this work.

The exact behaviour of the InfiniBand fat-tree routing algorithm depends on the network topology. A single change in the network topology means that the resulting routing tables will differ with regard to the unmodified topology. This means that there exist some specific topologies for which the design flaws in the algorithm can be observed. One of such situations occurs when the number of compute nodes connected to the tree is reduced. This behaviour is a problem in situations where the fat-tree is not fully populated with nodes. Such a situation

often occurs during cluster construction, for underpopulated clusters ready for future expansion (a common scenario), and for power saving clusters (green computing) where end nodes are powered down when not in use. This is also a common issue in enterprise systems that are limited by the number of rack units and often cannot utilize all the ports on a densely packed IB switch.

Another related issue is that for fat-tree routing, it is not possible to achieve all-to-all connectivity between all network nodes. With the general increase in system management capabilities found in IB switches, the lack of deadlock free all-to-all communication is a problem because IB diagnostic tools rely on LID routing and need full connectivity for basic fabric management and monitoring.

Next, the current fat-tree routing algorithm is an oblivious algorithm, that is, it treats every port in the fabric in the same manner. However, this approach is inadequate for modern systems where end-nodes are multi-port devices that are connected to multiple switches for the sake of fault-tolerance. Treating each port on such nodes independently leads to fault-tolerance issues because a multi-port node may have a single point of failure despite being redundantly connected to the fabric.

Finally, many network resources in IB are not utilized to their full extent. One such example are the Virtual Lanes (VLs). It is a well known fact that multiple virtual lanes can improve performance in interconnection networks [15, 16], but this knowledge has had little impact on real clusters. This is especially important for fat-trees that are innately deadlock-free so the VL resources are not used for anything else than Quality of Service (QoS).

The abovementioned issues give rise to the following research questions (RQs) that are addressed in this thesis:

- RQ1: *How to adapt the fat-tree routing algorithm to modern enterprise fat-tree topologies?*
- RQ2: *What network resources can be used to achieve high routing performance and full reachability?*

Another set of problems concern fault tolerance aspects of fat-tree topologies. The fat-tree routing is very prone to switch failures. If any failure in the fabric occurs or if the fabric does not comply with the strict rules that define a pure fat-tree, the subnet manager fails over to another routing algorithm, which may be undesirable from the performance point of view. This challenge has led us to formulate the following research question:

- RQ3: *How to make the fat-tree routing more resilient to switch failures?*

Lastly, the major issue is that the IB specification lacks any detailed description of inter-subnet routing methods. While router devices are well defined, not addressing subnet management interaction virtually stopped any IB-IB router development. However, the issue of the limited layer-2 addressing space in IB led to a renewed interest in IB-IB routers, and, especially for enterprise systems, the IB-IB router concept is currently gaining momentum. The set of challenges related to native inter-subnet routing for IB have led us to devise the following research question:

- RQ4: *What are the requirements for the InfiniBand inter-subnet routing algorithms and how these algorithms can use the local routing information supplied by intra-subnet routing algorithms?*

This dissertation presents and discusses mechanisms aimed at answering the abovementioned questions. These mechanisms improve the network performance, scalability and fault-tolerance. When designing each mechanism, our main concern was that it should be practical and usable in real scenarios. What is unique about these improvements is that each of them can work independently or as a plugin to the unmodified routing algorithm [14]. Combined together, their feature-base allows to efficiently route enterprise fat-tree-based systems.

1.2 Contributions

In this dissertation, we present a set of routing algorithms that build upon the original fat-tree routing by Zahavi [14]. The main goal of these algorithms is to improve the issues encountered when routing modern fat-tree IB systems.

First, we propose four improvements to the intra-subnet fat-tree routing algorithm: Balanced Enhanced Fat-Tree Routing (BEFT) [5], vFtree [6], sFtree [7], and mFtree [10]. BEFT is a major improvement over the original fat-tree routing because it solves the problem in which the throughput per node deteriorates both when the number of nodes in a tree decreases and when the node distribution among the leaf switches is non-uniform. vFtree is the first routing algorithm for IB networks that uses VL not for deadlock-avoidance but for performance increase in case of hot-spots. This solution is not only inexpensive, scalable, and readily available, but also does not require any additional configuration. sFtree is an extension to the fat-tree routing that enables full connectivity between any of the switches in a fat-tree when using IPoIB, which is crucial for fabric management features such as the Simple Network Management Protocol (SNMP) for management and monitoring, Secure SHell (SSH) for arbitrary switch access,

or generic web-interface access. This extension is also essential for systems that run the subnet manager on spine switches, so that Local IDentifier (LID) routed IB packets can be sent from one switch to another. mFtree is the last extension that makes sure that redundancy is achieved for multi-port end-nodes. Each improvement can be treated as a separate routing algorithm, but because they all touch upon a different problem, they can be combined together to deliver optimal performance when routing fat-tree topologies.

Furthermore, we propose two new routing algorithms for inter-subnet routing [9]. First, inter-subnet source routing (ISSR) is a generic routing algorithm that uses the source routing concept to send the packets between two or more subnets without any a priori knowledge about the target subnet. Second, the inter-subnet fat-tree routing (ISFR) is a specialized routing algorithm that is designed to work only on subnets that are fat-trees themselves.

Lastly, we propose a method for discovering the fat-tree fabric that uses reliable fabric information (switch roles) that are coded into the devices instead of trying to use the best-effort method that uses Global Unique IDentifier (GUID) lists or leaf switch marking by discovering the Host Channel Adapters (HCAs) [8]. Using the switch roles, we are able to assign a vendor specific switch role to each switch in the fabric, so the routing algorithm does not have to conduct any discovery when encountering a specific switch. Using the GUID lists shall yield the same effect, but it requires non-trivial effort to maintain a correct list following multiple component replacement operations. On the other hand, switch roles can be saved and restored as part of normal switch configuration maintenance following component replacements since it is not tied to the actual hardware instance like hardware GUIDs.

In the same paper, we also suggest a method for improving the fat-tree routing on degraded fat-trees. Until our proposal, the fat-tree routing was very strict when it came to fat-tree compliance checks, that is, even a failure of a single link could make the subnet manager fall back to suboptimal MinHop routing. This not only had detrimental effect on the performance, but also made the deadlock occurrence a real possibility. With our proposal, the subnet manager does not fall back to MinHop routing in case of simple failures, and we have shown that this leads to much better overall performance.

1.3 Research Methods

In this section, we describe the methodologies and approaches that we applied when conducting the research. First, the research problem was defined by us-

ing *exploratory research* [17], which was based on code analysis, data analysis and literature review. Having familiarized ourselves with a problem, *constructive research* [18] was applied to test theories and propose a final solution to the problem. It involved designing (constructing) the solution with diagrams, data-blocks and models, and later creating the pseudocode and prototyping the solution in a software language. If applicable, at this stage, also the simulation model was built for testing the solution. Lastly, for the solution to be viable, data had to be collected, analysed and compared with previous software implementations. In our case, for all the papers this was done by means of *simulations*, however, where applicable, *hardware experiments* were also conducted.

1.3.1 Exploratory Research

To define, investigate and understand the initial problem a variety of research methods were used. The most used ones were data analysis and code analysis. Since most of the research presented in this thesis builds upon existing systems, it was enough to test the current state-of-the-art to understand its shortcomings by isolating the key relationships between various variables influencing the behaviour of the system. This allowed us to quickly obtain valid results, which is essential during initial study. This method was applied to address *RQ1*, *RQ2* and *RQ3*, which required the system to be understood thoroughly before any improvements were proposed.

In the context of this thesis, the process of measurement was a critical step during this initial research phase. Narrow tests were conducted to analyse a specific feature on a well-defined system and data about its behaviour was collected. Since using a real system would be time consuming and expensive, a model of each system was built and tested using available network simulation tools like ibsim or IBMgtSim [19,20]. These tools are intended to allow building large scale IB systems mainly for troubleshooting purposes by running native IB management tools on a simulated fabric. Next, the data dumped by the subnet manager running on such a simulated fabric was analysed against the code of the routing algorithm to understand the behaviour of the system. This process was repeated multiple times on different systems, so that patterns could be established and the problem could be well-defined. Furthermore, literature review was used as a secondary research method during the initial research phase. Using the research databases, a background study in the related literature was conducted, so that similar solutions and approaches could be analysed for additional input.

1.3.2 Constructive Research

The first step in designing and implementing the solution was to present it using the available software modelling language [21] by means of diagrams, data-blocks and models. Next, the necessary data structures and algorithms were established and a high-level pseudocode was written to describe the final solution. Lastly, the solution was implemented in a software language and refined multiple times until it was tested on a simulated fabric or on a small cluster. Such an approach was especially useful to address *RQ1*, *RQ2* and *RQ3* where the proposed algorithms required multiple stages of validation.

The refining of the solution was conducted in a similar manner as the initial data analysis. Data dumped from the network simulator was analysed against the software code, and, if any discrepancies were found between the desired result and the actual results, the software code was modified. This was quite expensive and time consuming, however, it was the main research method applied in this thesis to develop the solutions, and allowed for easy repetitions of experiments using different parameters and topologies. In our case, this method was applied for all research papers apart from Paper V [9]. Using this research method allowed us to test the solution on large-scale systems comprising of thousands of nodes, which would not be ordinarily possible due to availability and security requirements of real systems. Furthermore, by testing our solutions in such a manner, we made sure that they do not break the system and they could be later safely introduced into commercial products.

Another research method used to construct the solution was simulation. It was mostly used for the inter-subnet routing in InfiniBand (*RQ4*) because there are no commercially available native IB-IB routers that suited the needs of the research and, as such, hardware experiments could not be conducted. Simulation itself required us to first create a model of the simulated object using a network description language [22] that corresponded with the real-life entity and later to abstract its function in the software layer. The main simulator tool used for evaluations was the OMNeT++/OMNEST network simulator [23].

1.3.3 Simulations

Simulation is a research method that abstracts the functions of a real system using a software model. Such a software model is especially useful for large-scale evaluation where the size of the system is limited only by the memory of the machine on which the simulation is run. Since most of our systems were predictable in a sense that the number of variables influencing the system was

known, the simulations were used to evaluate the performance of various routing algorithms for different scenarios. Simulations also have the additional benefit of being non-disruptive, so there was no risk in causing downtime to a real system by failing links or disconnecting devices. In our case, simulations were the most important tool for evaluating the performance and scalability of our solutions and were used in each of the research papers as method confirming the viability of the solution.

1.3.4 Hardware Experiments

Experiments are an expensive research method and due to the size of the systems analysed in this thesis, they were used only in Paper II [6], Paper III [7] and Paper VI [10] (all address *RQ2* and Paper II and Paper III also address *RQ1*). Experiments allow us to dynamically control the system by being able to fine-tune its every aspect. Even though setting up hardware experiments takes much longer than setting up a simulation, it is more rewarding in a sense that obtaining the results for a ten second experiment takes exactly ten seconds while obtaining the same results from a simulation may take several days.

Because experiments are performed on a real system, extra layers of complexity are added that are not present during simulations. One case can be the selective resetting (depending on the cable manufacturer) of fibre links when the switch firmware is upgraded, which may cause unexpected behaviour. Another case is the very limited management capabilities of older IB switches, which makes conducting some experiments impossible. Furthermore, a large problem for IB is that many of the advanced features are embedded into the switch firmware. This not only means that these features are not available on older systems, but also that they are usually proprietary and not compatible between various switch manufactures.

1.4 Published works

The conclusions presented in this thesis are based on results that were published or accepted for publication.

The results regarding Balanced Enhanced Fat-Tree Routing were published in [5]. Next, the vFtree work was presented in [6]. Work related to sFtree routing algorithm was covered in [7]. The switch roles, fat-tree discovery and routing of degraded fat-trees was published in [8]. Work related to inter-subnet InfiniBand routing is a result of collaboration with researchers at Departamento

de Informática de Sistemas y Computadores (DISCA) at Technical University of Valencia, Spain and was published in [9]. Lastly, the final paper on multihomed routing in InfiniBand was accepted to 22nd Euromicro International Conference on Parallel, Distributed and Network-Based Processing. All paper titles, including place of publication and authors, are listed below:

Research Papers

Paper I:

Title: Achieving Predictable High Performance in Imbalanced Fat Trees
Authors: Bartosz Bogdański, Frank Olaf Sem-Jacobsen, Sven-Arne Reinemo, Tor Skeie, Line Holen and Lars Paul Huse
Venue: Proceedings of the 16th IEEE International Conference on Parallel and Distributed Systems (ICPADS 2010), pages 381–388, IEEE Computer Society, December 2010, Beijing, China.

Paper II:

Title: vFtree - A Fat-tree Routing Algorithm using Virtual Lanes to Alleviate Congestion
Authors: Wei Lin Guay, Bartosz Bogdański, Sven-Arne Reinemo, Olav Lysne, and Tor Skeie
Venue: Proceedings of the 25th IEEE International Parallel & Distributed Processing Symposium (IPDPS 2011), pages 197–208, IEEE Computer Society, May 2011, Anchorage, USA.

Paper III:

Title: sFtree: A fully connected and deadlock free switch-to-switch routing algorithm for fat-trees
Authors: Bartosz Bogdański, Sven-Arne Reinemo, Frank Olaf Sem-Jacobsen, and Ernst Gunnar Gran
Venue: ACM Transactions on Architecture and Code Optimization (ACM TACO), Volume 8 Issue 4, January 2012, Paris, France.

Paper IV:

Title: Discovery and Routing of Degraded Fat-Trees
Authors: Bartosz Bogdański, Bjørn Dag Johnsen, Sven-Arne Reinemo, and Frank Olaf Sem-Jacobsen
Venue: Proceedings of the 13th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT 2012), pages 689–694, IEEE Computer Society, December 2012, Beijing, China.

Paper V:

Title: Making the Network Scalable: Inter-subnet Routing in InfiniBand
Authors: Bartosz Bogdański, Bjørn Dag Johnsen, Sven-Arne Reinemo and José Flich
Venue: Proceedings of the Euro-Par 2013 International Conference, pages 685–698, Springer Berlin Heidelberg, August 2013, Aachen, Germany.

Paper VI:

Title: Multi-homed Fat-Tree Routing with InfiniBand
Authors: Bartosz Bogdański, Bjørn Dag Johnsen, Sven-Arne Reinemo
Accepted to: 22nd Euromicro International Conference on Parallel, Distributed and Network-Based Processing, IEEE Computer Society, February, 2014, Turin, Italy

Patents

Most of the methods presented in this thesis are patented or patent pending. The patents are as follows:

- ORACL-05282US1, System and method for using virtual lanes to alleviate congestion in a fat-tree topology (granted as US20130121149)
- ORACL-05279US1, System and method for providing deadlock free routing between switches in a fat-tree topology (granted as US20130114620)
- ORACL-05383US0, System and method for supporting sub-subnet in an InfiniBand (IB) network (granted as US20120307682)
- ORACL-05387US1, System and method for routing traffic between distinct InfiniBand subnets based on source routing (granted as US20130301645)

- ORACL-05387US2, System and method for routing traffic between distinct InfiniBand subnets based on fat-tree routing (granted as US20130301646)
- ORACL-05418US1, System and method for supporting discovery and routing degraded fat-trees in a middleware machine environment (non-provisional patent submitted)
- ORACL-05477US0, System and method for supporting multi-homed fat-tree routing in a middleware machine environment (provisional patent submitted)

Chapter 2

Background

This chapter starts with a brief introduction to lossless interconnection networks contextualized in terms of topologies and routing in Section 2.1. Next, in Section 2.2, a comprehensive description of InfiniBand Architecture is presented.

2.1 Interconnection Networks

Interconnection networks were traditionally defined as networks that connect multiprocessors. However, interconnection networks evolved dramatically in the last 20 years and nowadays play a crucial role in other areas like storage area networks (SAN) or high-performance computing (HPC) clusters. The focus of this thesis is on the routing protocols for InfiniBand interconnection technology. InfiniBand is commonly used as the networking component in HPC, SAN or enterprise database deployments that handle very large amounts of data.

Interconnection networks, as all data networks, consist of two basic elements: *network nodes* that generate, route and consume the data, and the *communication medium* that is used to connect the network nodes to form a data network. Network nodes include both the hosts and the networking hardware such as switches or routers. In case of interconnection networks, the hosts generate and consume (we shall refer to those as *source nodes* and *destination nodes*, respectively) the majority of data while switches and routers forward the traffic through the network from the source node to the destination node. The communication media used to connect the network nodes to form an interconnection network may include electrical cables or optical cables of varying speeds

and characteristics. In this thesis, we will only consider switched networks with point-to-point links between the network nodes, and we will not discuss in detail any shared media networks[1].

Even the simplest interconnection network consisting of two interconnected network nodes forms a *network topology*. A network topology is the layout or the organization of the interconnected nodes that describes the structure of the network. A topology can be either *physical* when it describes the shape of the communication medium and placement of the network nodes, or *logical*, when it describes the paths that the data takes between the network nodes.

The logical topologies are usually determined by the *routing* whose goal is to select the paths (sequences of intermediate nodes and links) for the data flowing from the source nodes to the destination nodes. Because in most of the topologies there are many paths that the data can take, there are also many approaches as how to compute the best path for the data. Based on the approach the routing algorithm takes to select the path for the data it can be classified according to multiple criteria [25, p. 140-145]. A good path is the one that has a minimal number of hops (number of traversed network nodes) and uses the network resources in a balanced manner [26, p. 13]. In the majority of data networks, the selection of a routing algorithm is a critical decision that has an impact on both the network *latency* and the network *throughput* [27]. Latency is the time elapsed between the time a message is generated at the source node and the time it is delivered to the destination node [25]. Throughput is the maximum amount of data delivered by the network per unit of time [25].

The data flowing between the network nodes usually forms traffic patterns that depend on the characteristics of the applications running on the nodes and their relative contribution to the overall traffic. One of the problems with selecting an optimal routing algorithm is that the choice varies depending on the network traffic conditions. For uniform traffic pattern, where the distribution of destinations is uniform for each contributing source, the performance of some algorithms may be higher than the performance of others. When traffic is not uniformly distributed and there are regions of traffic *congestion* (hotspot areas), the situation may be reversed [25, p. 199].

Another aspect of interconnection networks that influences their performance is the *switching technique*, which is also tightly related both to *flow control* and *buffer management*. The switching technique defines how packets flow through a switch or a router from the ingress to the egress port while flow

[1]Shared media networks are those where a set of nodes is connected to a common medium like for example the original Ethernet over a shared coaxial cable [24].

control and buffer management mechanisms determine the packet flow through a link between two network nodes.

In Section 2.1.1 a more detailed information will be presented about popular topologies present in today's interconnection networks. Section 2.1.2 is devoted to the fat-tree topology, which is the main topology studied in this thesis. Switching techniques and flow control mechanisms will be discussed in Section 2.1.3. Finally, a discussion about routing and routing algorithms will follow in Section 2.1.4.

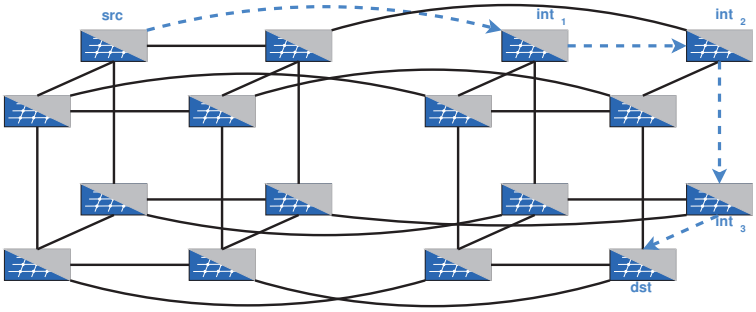
2.1.1 Topologies

Topologies for interconnection networks can be classified into four major groups: shared-bus networks, direct networks, indirect networks and hybrid networks [25]. In this thesis, the focus is on indirect networks. The choice of topology is one of the most important steps when constructing an interconnection network. The chosen topology combined with the routing algorithm and application's workload determines the traffic distribution in the network.

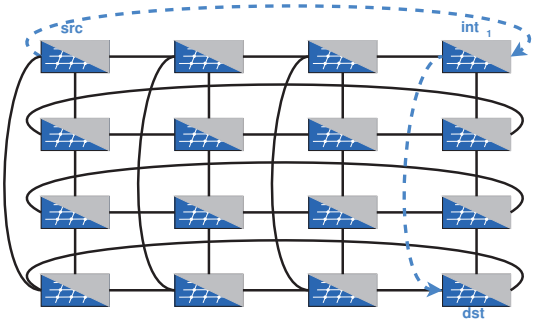
Ideally, a topology would have no need for routing. This could be achieved by connecting each node with every other node so a fully-connected topology (or a full-mesh) is built, however, such a design is cost-prohibitive and very impractical due to scalability issues that become an issue for larger networks. First, in a topology consisting of n nodes, one would require $\frac{n(n-1)}{2}$ bidirectional cables to connect all nodes in a full-mesh pattern. Second, each node would have to have enough available ports to connect to all the other nodes. Last, the cables occupy space and the wiring area in a data center rack is usually very limited. Because of these reasons, there are many topologies that try to compromise between the cost of constructing the network and the achieved performance.

2.1.1.1 Shared-Bus Networks

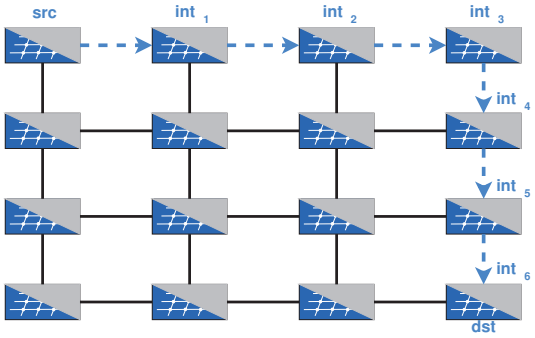
The simplest and the easiest way to connect multiple nodes together is to use a shared-medium network [28, p. 17]. This network is characterized by all nodes being connected to a single bus and the communication is broadcast in nature, that is, all nodes receive the packets injected into the network but only the destination node interprets them. When two nodes want to transmit at the same time a collision may occur, so many such networks will have a collision avoidance scheme in place. An example of such a network is a ring topology.



(a) 4D hypercube



(b) 2D torus



(c) 2D mesh

Figure 2.1: The strictly orthogonal direct networks.

2.1.1.2 Direct Networks

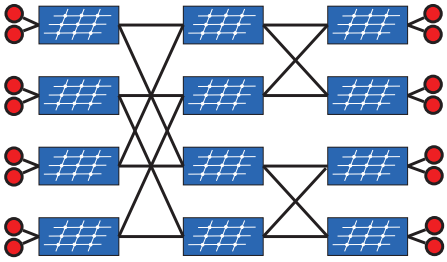
The second group of topologies are the direct networks. The term direct network refers to topology in which each network node acts as both a switch and a computing node. There is a large range of different direct networks existing in current HPC systems. The most popular ones are built according to the k-ary n-cube definition[2] [29]. Such topologies consist of n dimensions with k switching elements in each dimension. The radix, k, may be different for each dimension. An example of such topologies: a *hypercube*, a *torus*, and a *mesh* are shown on Fig. 2.1(a), Fig. 2.1(b), and Fig. 2.1(c), respectively. A mesh can be visualized as a grid structure, usually in two or three dimensions and a torus is a mesh with added wraparound links to the edges of the mesh grid, thus, making it set of interconnected rings. Meshes, hypercubes and tori belong to a family of topologies referred to as strictly orthogonal. Such topologies are characterized by having at least one link in every direction, which makes routing simple [25] because of their regular structure. The dotted lines depicted on Fig. 2.1 mark the links carrying data traffic from a *src* node to a *dst* node through a set of intermediate (*int*) nodes when a simple dimension-order routing algorithm is applied to each topology. Since each node is described by a set of $(x, y, z, ...)$ coordinates, such an algorithm uses a finite-state machine that nullifies the offset between the *src* and the *dst* nodes in one dimension before routing the next dimension.

Another example of a newly proposed random graph topology is Jellyfish [30] in which switches are randomly connected with each other and all nodes are distributed uniformly among all the switches. Jellyfish allows constructing arbitrary-size networks and easy expansion of existing networks, however, routing, cabling and troubleshooting challenges are still not fully resolved.

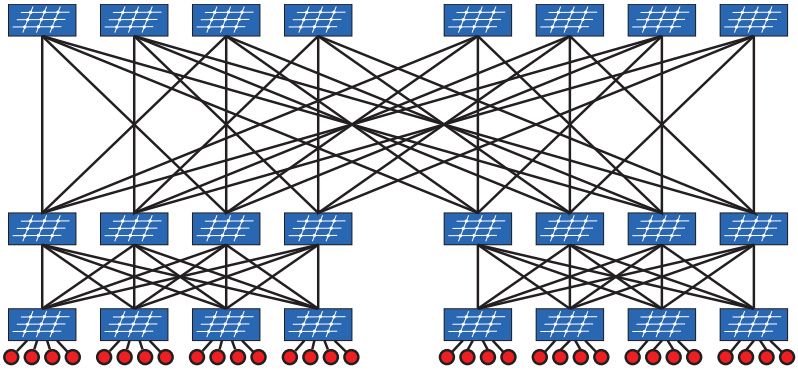
2.1.1.3 Indirect Networks

The third group of topologies, and the one that will will be the main topic of this thesis, are the indirect networks. Similarly to the direct networks, there are many possible varieties of indirect networks. The most popular ones are called *multistage interconnection networks* (MINs) where computing nodes are connected to the same switching stage, and traffic between the source and the destination nodes flows through one or more intermediate switching stages. MINs are considered to be expensive networks to build, but due to their high capacity, they are often seen in modern high-performance systems. Examples of MINs include

[2] Sometimes a mesh topology is referred to as a k-ary n-mesh.



(a) A butterfly network with 16 nodes and 12 switches.



(b) A fat-tree network with 32 nodes and 24 switches.

Figure 2.2: Multistage interconnection networks.

Clos [31], Beneês [32], Delta [33], butterfly [26, p. 75] (shown on Fig. 2.2(a)), flattened butterfly [34] and fat-tree [35] (shown on Fig. 2.2(b)) topologies. The fat-tree topology will be discussed in more detail in Section 2.1.2.

2.1.1.4 Hybrid Networks

The last group of topologies are the hybrid ones. Usually, these are the topologies that cannot be strictly classified as direct or indirect because they combine the characteristics of both of these groups. An example of such a topology is the dragonfly [36]. It is a hierarchical topology which consists of n group topologies and an inter group topology. The topology inside a single group can be

any topology[3]. Each group is connected to all other groups directly using the inter group topology. Another example of a hybrid topology, that combines the aspects of shared-medium and direct networks is the hypermesh [37] that can be described as a "mesh of buses", that is, a topology where multiple buses are connected in multiple dimensions in such a way that each node is connected to every other node in the same dimension.

The distinction between direct and indirect networks is largely academic since every direct network can be redrawn into an indirect network by splitting each node into a separate switching unit and a computing unit [26, p. 47]. However, we can distinguish the direct and indirect networks in another way: for direct networks every switching unit is associated with computing nodes whereas in indirect networks only a subset of switches are connected with the computing nodes.

Many of the topologies mentioned here are used in today's supercomputers and enterprise data center systems: fat-tree topologies are used for example in in Tianhe-2 or SuperMUC supercomputers or in Oracle's engineered systems [38]. Another example is Fujitsu's 6D Mesh/Torus that is used in the Japanese K computer [39] that treats 12 compute nodes connected as a 2x3x2 3D Mesh as nodes of a 3D Torus.

2.1.2 Fat-Trees

The fat-tree topology is one of the most common topologies for HPC clusters today, and for clusters based on InfiniBand (IB) technology the fat-tree is the dominating topology. There are three properties that make fat-trees the topology of choice for high performance interconnects: *(a)* deadlock freedom, the use of a tree structure makes it possible to route fat-trees without using virtual channels for deadlock avoidance; *(b)* inherent fault-tolerance, the existence of multiple paths between individual source destination pairs makes it easier to handle network faults; *(c)* full bisection bandwidth, the network can sustain full speed communication between the two halves of the network.

Fat-trees are innately fault-tolerant due to path diversity, that is, multiple paths connecting each source with each destination. Sem-Jacobsen proposed various mechanisms that improve fault-tolerance in fat-trees [40–46]. The path diversity of fat-trees was also exploited in multiple works addressing scalable data center designs [47–51]. The fault-tolerant aspects of the work presented

[3]The recommendation in [36] is the flattened butterfly.

in this thesis use and expand the groundwork that was established by Sem-Jacobsen.

2.1.2.1 Fat-Tree Variants

The fat-tree topology was introduced by Leiserson in [35]. Since then, fat-trees have been extensively studied beginning with Distributed Random Access Machines [52, 53] through the seminal paper on the CM-5 supercomputer [54] and until they have become a common topology in HPC and attracted increasing attention in commercial data center networks [47, 48, 55]. The fat-tree is a layered network topology with equal link capacity at every tier[4], and is commonly implemented by building a tree with multiple roots. The fat-trees can be built using several definitions, of which the most popular are the m-port n-tree definition [56] and the k-ary n-tree definition [57]. The k-ary n-tree definition - similarly to the k-ary n-cubes and k-ary n-flies [58] - describes a subclass of regular fat-trees that can be constructed by varying two parameters k and n. A topology built using this definition will have a recursive structure and will consist of k^n end nodes and nk^{n-1} $2k$-port switches. On the other hand, an m-port n-tree consists of $2(m/2)^n$ end nodes, $(2n-1)(m/2)^{n-1}$ m-port switches and has a height of $n+1$. Both of these definitions, however, cannot describe many of the fat-tree topologies that are used in real systems because they are unable to represent multiple links connecting two switches nor different number of connections at each level.

GFT and XGFT notations presented by Öhring [59] describe generalized fat-trees. $GFT(h, m, w)$ describes a fat-tree of height h and consisting of m^h end nodes. Each non-root switch and end node has w parent switches and each switch has m children. $XGFT(h; m_1, ...m_h; w_1, ...w_h)$ extends GFT by allowing to choose m and w parameters independently for each fat-tree stage, therefore, varying the number of switches and links between different fat-tree stages. m_L is the number of different nodes at level $L-1$ connected to the nodes at level L and w_L is the number of different nodes at level L connected to nodes at level $L-1$. With this notation, most of the simple fat-trees can be described. Nevertheless, real systems often use multiple parallel connections between two nodes to preserve the Cross Bisectional Bandwidth (CBB) and XGFT is unsuitable to describe such systems because it allows only a single connection between a pair of nodes.

Another fat-tree variant was introduced by Valerio et al. in [60] and it aimed

[4]This feature applies only to balanced pure fat-trees.

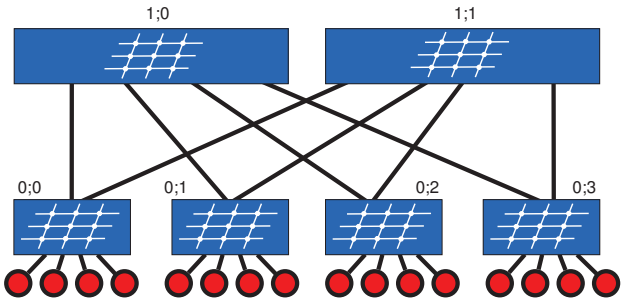
at maximizing the number of leaf connections in the topology. However, this variant removed one of the main fat-tree characteristics - multiple paths - and so an extension alleviating this issue was proposed in [61]. Nevertheless, none of the above notations could capture all real-life fat-trees, Zahavi [1] proposed Parallel ports Generalized Fat-Tree (PGFT) and Real-Life Fat-Tree (RLFT) notations to describe practical fat-tree used in today's HPC systems.

$PGFT(h; m_1, ...m_h; w_1, ...w_h; p_1, ...p_h)$ is defined similarly to XGFT with the additional p_L parameter, which is the number of parallel links connecting two nodes at levels L and $L-1$. RLFT is a notation that is derived from PGFT to describe real-life fat-trees. All RLFT have full CBB, which means that the number of input ports on each node is equal to the number output ports on the same node: $m_L w_L = m_{L+1} w_{L+1}$. Next restriction is that the number of ports on each switch is a constant value, that is, the same switch model is used at all levels of the fat-tree. The last restriction is that the end-ports are connected with a single link to the fat-tree, that is: $w_1 = p_1 = 1$. Fig. 2.3 shows an example of a XGFT(2;4,4;1,2) and PGFT(2;4,4;1,2;1,2) (which is also a RLFT). We can notice that using XGFT notation it is not possible to represent two links between a pair of switches and the requirement for full CBB cannot be met.

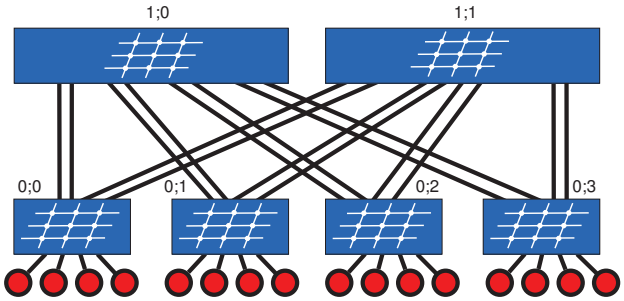
2.1.2.2 Construction Cost

There were numerous studies that took into account the cost of constructing a fat-tree and compared it to other network topologies [35, 47, 62, 63]. The general agreement is that the cost of constructing a fat-tree can be lowered by using high-radix switches. It is also well-established that the most cost-effective topology solutions are meshes and 3D-Tori [64]. The general agreement is that building fat-trees with CBB yields more performance but at an often prohibitive cost, therefore, real fat-tree fabrics are very often *oversubscribed*. By oversubscription we mean that the network bandwidth at different fat-tree stages is not equal. An oversubscription ratio of 2:1 is claimed to reduce the overall cost by up to 50% depending on the size of the fabric [65]. When a fabric is oversubscribed, the input nodes do not achieve the full bandwidth of the fabric, but the low latency is maintained, which is a desirable trade-off for applications that require low latency and moderate bandwidth. Despite the high cost, fat-trees are *universal* topologies [35], that is, for a given physical volume of hardware, no other topology will be much better[5] than a fat-tree. Even though other topologies such fat-pyramids [66, 67] or fat-stacks [68] can simulate other topologies

[5]Leiserson in his seminal work [35] proves that the experienced slowdown is at most polylogarithmic for any topology of comparable physical volume that is simulated using a fat-tree.



(a) XGFT(2;4,4;1,2)



(b) PGFT(2;4,4;1,2;1,2)

Figure 2.3: XGFT notation cannot be used to describe full CBB [1].

with less slowdown, the fat-tree is still the most popular universal network due to its lower construction cost. Furthermore, by increasing application locality, the cost of constructing a fat-tree can be optimized by introducing a thin-tree topology [69], which has reduced bandwidth at upper tree stages.

2.1.3 Switching Techniques and Flow Control

The majority of interconnection networks are labelled as lossless, that is, packet drops (losses) under normal network conditions are not allowed. The special conditions that allow packet drops usually occur when: a deadlock-recovery scheme is in effect (further described in Section 2.1.4.4, a host or link integrity error, an excessive buffer overrun or a flow control update error happen resulting in a corrupted packet. Also, selective packet retransmission and out-of-order

packet delivery are usually not supported. In a lossless network, if a retransmission is required due to one of the above errors occurring, the missing packet along with all the subsequent packets is retransmitted by the source node.

Due to these restrictions, lossless networks implement flow control mechanisms that manage the number of packets a link between two nodes can receive at any time in such a way as to minimize resource conflicts that would otherwise lead to network errors. The resources that are controlled by the flow control mechanisms are the *buffers* that hold the *flits* (flow control digits - the smallest unit of information that is recognized by the flow control mechanism), the bandwidth of channels that transport them and the channel state.

The goal of a flow control strategy is to make sure that the receiving node always has available buffer space when a packet arrives. In other words, flow control strategy decides how to allocate the network resources so that the network is efficient. Typically, the channels and the buffers are coupled with each other. It means that when a particular buffer is allocated to a packet, only this packet can use the associated channel. This implies that such a packet can sometimes block other packets that need to use the same channel.

A switching technique, on the other hand, controls how a packet or a flit internally pass through a node from the node's ingress port to its egress port. There are two main families of switching techniques, *circuit switching* and *packet switching*, each of which having its own advantages and disadvantages [70].

2.1.3.1 Circuit Switching

The idea of circuit switching can be traced to the early telecommunication networks. During a telephone call, electro-mechanical crossbar switches in the telephone exchanges established and reserved a continuous path between the two telephone nodes and that path was deallocated only when the telephone call ended. The procedure is similar in modern data networks where the necessary resources across the network are allocated before data communication between the two nodes starts, which means that buffers are not required. Circuit switching has the advantage of being very simple to implement [70], however, due to efficiency issues, strict requirements for resource availability, and the delays caused by the circuit setup phase, this switching technique is not often found in modern interconnection networks.

2.1.3.2 Packet Switching

Packet switching was invented by Paul Baran [71] in the 1960s. The fundamental difference between circuit switching and packet switching is that for packet switching, all transmitted data is grouped into sized blocks called packets where each packet is independently forwarded through the network. Such an approach allows to take independent routing and forwarding decisions in a per hop-by-hop fashion [70]. Packet switching techniques can be divided into three main groups: *Store-And-Forward* (SAF) [28, p. 356], *Virtual Cut-Through* (VCT) [72] and *wormhole flow control* [73, 74].

SAF is a technique where the whole packet is received (stored) before it is forwarded towards the next node. SAF has the disadvantage of an end-to-end delay caused by accumulating the whole packet in the node's memory before it is forwarded to the next-hop node. This end-to-end delay increases proportionally with each subsequent hop on the path between the source and the destination nodes. Another disadvantage of SAF is that it puts an upper limit on the maximum packet length because each packet must be completely stored in the available buffer space of a switch. However, SAF is easy to implement and has the advantage of easily detecting damaged packets through computing and comparing the cyclic redundancy check (CRC) packet field. SAF was often used in Ethernet switches, however, in the last few years it lost ground to the more efficient cut-through switching.

On the other hand, when performing VCT switching, the node does not wait until the whole packet is received before it starts transmitting it to the next node. Each data packet contains routing information in the packet header that allows the intermediate node to select the egress port and start sending the packet as soon as that information is received and the destination address has been looked up. This approach allows to reduce the latency since the node does not wait to receive the whole packet, and is currently used in all the ultra-low latency technologies like InfiniBand or Data Center Ethernet.

In some congestion scenarios, the VCT switching may still require the packet to be completely stored in the buffer before acting on it. This occurs when the selected egress port is busy with transmitting packets coming from other ingress interfaces. In such a situation, the node needs to buffer the packet on which the routing decision has already been made, therefore, lowering the performance to a level similar to SAF. This problem is addressed by the wormhole flow control that operates similarly to VCT, but allocates buffers not in packet units but in flit units. This means that one packet can occupy buffers in several nodes as it is forwarded across the network. In wormhole flow control the buffer space required

is much smaller than for VCT and it is used much more efficiently. Wormhole flow control has the disadvantage of complicating the *deadlock* problem because a single packet spans across multiple nodes and flits do not contain routing information, so buffer allocation cannot be restricted [75, 76]. Furthermore, if no flit buffers are available to hold a flit, wormhole flow control may block a channel in mid-packet [26].

2.1.3.3 Virtual Channels

The concept of virtual channel[6]. (VC) flow control was introduced by Dally in the late eighties [77]. In interconnection networks that use a credit-based flow control mechanism the main advantage of using VCs is to decouple the buffer allocation from the channel allocation. Normally, the receiving node keeps record of the available buffer space for each of its links by decreasing the credit counter when buffer space is allocated and increasing it when the buffer space is deallocated. The sending node, on the other hand, records the amount of credits it is allowed to send. At regular intervals, the receiving node informs the sending node how many credits it is allowed to send. This is done to synchronize the credit counters, which may have different values if a packet is lost due to CRC errors. VCs offer an improvement to flow control mechanisms by allowing to logically split the link's physical resources into several virtual resources. This means that each such virtual link has its own buffer space and flow control resources.

Fig. 2.4 presents an example of VC credit-based flow-control. VC0 runs out of credits when it consumes the credit after the first cycle (depicted by a bold D). It has to wait until a new credit is available, which happens at cycle 9 (the credit is depicted by a bold C). Other VCs are unaffected by the lack of credits at VC0. Furthermore, the physical share of the bandwidth that was used by VC0 is used by VC1 and VC2. Adding a VC to a link does not add physical bandwidth to that link, but it allows for a more efficient use of the available resources. It has been demonstrated that VCs improve the overall network performance [15,78] and, in our case, we also used the concept of VCs to address *RQ2* in Paper II [6]. Furthermore, the concept of VCs makes it possible to build virtual networks on top of a physical topology. These virtual networks can be exploited for multiple purposes such as efficient routing, deadlock avoidance, fault-tolerance and service differentiation.

[6]In this thesis, we will often use the term *virtual lanes* (VLs) to describe VCs.

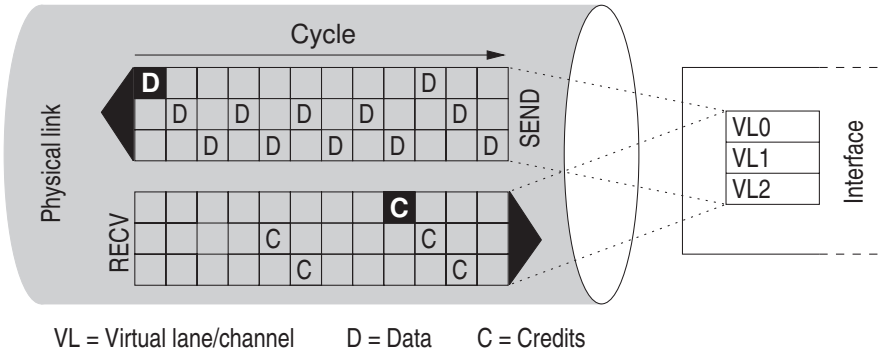


Figure 2.4: Virtual channel flow control.

2.1.4 Routing

The goal of a routing is to select a path along a network for packets flowing from a source node to a destination node. Looking at any network presented on Fig. 2.1 and Fig. 2.2, one can observe that there are multiple possible paths between a pair of nodes. The responsibility of a routing algorithm is to select a path from such a set of possible paths. Routing algorithms differ in many respects and they can be classified using various metrics such as: the timing of the routing decision and its location, the number of supported destinations, implementation (table lookup or finite-state machine), how many alternative paths are provided and how a specific path is chosen. A good routing algorithm tries to balance the traffic across multiple links and the more balanced is the usage of links in the topology, the better is the network throughput.

2.1.4.1 Routing Function

Routing function (or a routing engine) controls how the forwarding table in a switch or a router is constructed. Routing function can be defined in three fundamental ways [26, p. 163], but in this thesis, we will only consider two of them because they directly relate to our research in InfiniBand networks.

The first one is $R : N \times N \rightarrow \varphi(C)$ where N is any node and C is a channel. A routing function defined in such a way is called an *incremental* routing function. Such a function is executed at every node x to which a packet with a destination y arrives and it returns a set of output channels towards the destination node y. Formally, the relation is defined as $D = R(x, y)$ where $D \subseteq C_{x_{output}}$. This

routing function is most commonly used in TCP/IP and InfiniBand networks.

The second relation is $R : C \times N \rightarrow \varphi(C)$ where N is any node and C is a channel. It is also an incremental routing function that is executed at every intermediate node, but instead of taking the current node x as one of the parameters, it takes an input channel c_x. Again, the formal relation returning a set of channels can be defined in the following manner: $D = R(c_{input_x}, y)$ where $D \subseteq C_{x_{output}}$. The set of channels returned by this routing function provides additional information to the routing algorithm and allows to decouple the dependencies between the input and the output channels, which is exploited to avoid deadlocks [26, p. 164]. Furthermore, this relation can be extended to take into account any kind of routing information into consideration when choosing the output channel: $D = R(z, y)$ where $z \subseteq N_{src_{addr}}$ or z is the set of nodes traversed so far. Such a routing function is used for the ISSR routing algorithm – one of the inter-subnet routing algorithms we propose in Paper V [9].

2.1.4.2 Taxonomy of Routing Algorithms

There exist multiple classifications of routing algorithms [28, 79–85], however, the most complete one was presented by Duato [25, p. 140]. It extends the previous work by Gaughan [85] and classifies the algorithms using the following four principal features.

First, routing algorithms are classified based on the number of destinations they support. *Unicast* routing algorithms support packets that have only a single destination and *multicast* routing algorithms support packets that can reach multiple destinations. Examples of unicast routing algorithms for InfiniBand are fat-tree routing [14], LAyered SHortest path routing (LASH) [86] or Deadlock-Free Single-Source Shortest Path (DFSSSP) routing [87]. There are various multicast routing algorithms with each differing as to how build the delivery tree - the collection of nodes and links that the multicast packets traverse, however, the most popular is the Protocol Independent Multicast - Sparse Mode [88] protocol. In this thesis, we will only consider unicast routing algorithms.

Second, the path for the packet can be computed in a *centralized* manner during the network configuration (as it happens in most InfiniBand systems) or it can be computed in a *distributed* manner by network devices that use network or link state information to find the next-hop router (as it happens in most IP networks). Another technique is to calculate the path when injecting the packet at the source, which is called *source routing*. All of these can be combined into

a scheme that is called *multiphase routing*.

Third, routing algorithms can use a pre-calculated routing tables (*table lookup*) or run a *finite-state machine* in software or hardware that calculates the output port for each packet. Route computation can use the policy-based routing concepts where control lists are applied to the packet's header. If the header fields match the specified criteria, the output port from the router is obtained.

Last, if a routing algorithm always provides the same single path across the network between a particular source and a particular destination the routing algorithm is called *deterministic*[7]. However, if a routing algorithm allows to select different paths based on the current or historical network conditions, we call such an algorithm to be *adaptive*. Adaptive routing algorithms can be further classified based on the progressiveness, minimality and the number of paths they return towards a specific destination. In this thesis, adaptive routing algorithms will not be considered.

2.1.4.3 Fat-Tree Routing

For fat-trees, as for most other network topologies, the routing algorithm is essential in order to exploit the available network resources. In fat-trees, the routing consists of two distinct phases: the upward phase in which the packet is forwarded from a source in the direction of one of the root switches, and the downward phase when the packet is forwarded downwards to the destination. The transition between these two phases occurs at the lowest common ancestor, which is a switch that can reach both the source and the destination through its downward ports. Such an implementation ensures deadlock freedom.

The popularity of fat-trees led to many efforts trying to improve their routing performance. This includes the current approach that the OpenFabrics Enterprise Distribution [12], the de facto standard for InfiniBand system software, is based on [13, 14], and other proposals that did not gain widespread popularity, such as RANDOM algorithm [89] or MLID routing scheme [56]. Valiant's algorithm [90] could be used to route messages in a fat-tree by randomly choosing a top switch [53], which would result in a good average load, but could immediately create an imbalance if many end nodes choose the same switch [26]. All these proposals, however, have several limitations when it comes to flexibility and scalability. One problem is the static routing used by IB technology that limits the exploitation of the path diversity in fat-trees as pointed out

[7] In fact, deterministic routing algorithms are a special case of oblivious routing algorithms that route packets without considering the state of the network.

by Hoefler et al. in [91]. Another problem with the current routing are its shortcomings when routing oversubscribed fat-trees as addressed by Rodriguez et al. in [92]. Next, due to multiple links connecting each source with each destination, a good routing algorithm would exploit this feature by distributing traffic between all sources and all destinations over all possible links. By keeping the packet size minimal, the distribution would be optimal. An algorithm based on such an idea was proposed by Yuan et al. [93]. The authors show that their algorithm outperforms any single-path routing algorithm for any uniform or cluster traffic pattern. However, such an algorithm is not used in today's systems due to out-of-order delivery, the need for packet reordering and packet fragmentation. Furthermore, even though the link load is balanced perfectly, other practical issues related to buffering and congestion, such as head-of-line blocking and hotspots, are not taken into account. Any hotspot traffic pattern would break the basic assumptions of the non-blocking network and by completely distributing all traffic to all destinations over the entire network would lead to severe congestion since most of the buffer queues would be filled up with hotspot-destined traffic [94].

Despite these improvements, the fat-tree routing algorithm was not mature enough to be used in enterprise systems. Also, other proposals to utilize the fat-tree topology, such as the one to use it in Ethernet-based data centers [47] did not gain widespread usage so far.

OFED enhances the fat-tree routing by ensuring that every path to a given destination converges towards the same root node in the fat-tree, causing all packets with the same destination to follow a single path in the downward phase. The inspiration for this path distribution was [56], where the authors present the m-port n-tree definition and the associated MLID routing algorithm which performs a good distribution of the initial paths. Another implementation of a similar routing/balancing algorithm is given in [13]. In a fat-tree that is not oversubscribed, this allows for a dedicated downward path towards every destination that does not share traffic with any other destination. Gomez et al. [13] show that this path distribution gives very good performance for uniform traffic patterns. By having dedicated downward paths for every destination, contention in the downward phase is effectively removed (moved to the upward phase), so that packets for different destinations have to contend for output ports in only half of the switches on their paths.

The implementation presented in [14] also ensures that every path towards the same destination converges at the same root (top) switch such that all packets toward that destination follow a single dedicated path in the downward direction. By having a dedicated downward path for every destination, contention

in the downward phase is effectively removed (moved to the upward stage), so that packets for different destinations have to contend for output ports in only half of the switches on their path. In oversubscribed fat-trees, the downward path is not dedicated and is shared by several destinations.

The fabric discovery complexity for the IB implementation of the fat-tree routing algorithm is given by $\mathcal{O}(m+n)$ where m is the number of edges (links) and n is the number of vertices (nodes). The routing complexity is $\mathcal{O}(k \cdot n)$, where k is the number of end nodes and n is the number of switches.

2.1.4.4 Deadlock

A deadlock, by analogy, is a situation similar to a traffic gridlock. Cars (packets) are waiting for the traffic lanes (links) to be freed so they can proceed, however, the cars that are supposed to free the lanes cannot progress because other cars are blocking other lanes. Similar situation occurs in lossless interconnection networks where flow control mechanisms enforcing lossless communication prevent packets from moving forward until enough buffer space becomes available on the receiving end of a channel. If a sequence of packets waiting for channels forms a cycle then the network is in a deadlocked state and will remain in that state until deadlock recovery is initiated.

Because deadlock is a catastrophic event for the whole network and its performance, there are mechanisms whose goal is to prevent, avoid and recover from a deadlock [95]. Deadlock prevention is a technique that for each packet transmission reserves all the required resources before starting the transmission. Because this technique may be inefficient and impractical in distributed systems where the accurate knowledge about the network state and resource availability is limited, it is rarely used today. Deadlock avoidance technique only allows granting a resource to a packet when the resulting global system state is safe. Deadlocks can be also avoided by properly routing the fabric in such a way that a deadlock cycle is never formed as has been shown in Paper III [7], which addresses *RQ1* and *RQ2*. A deadlock recovery technique grants resources to packets without any verification, however, a deadlock detection mechanism must be present to detect a deadlock in the first place and then a deadlock resolution mechanism is required that usually deallocates the blocked network resources (breaking the cycle) and re-establishes normal network operations. Deadlock recovery technique is applied when deadlocks are rare and their influence on the network can be tolerated [25, p. 85].

From the system-wide perspective of an interconnection network, deadlock freedom is a crucial requirement. The necessary condition for a deadlock to

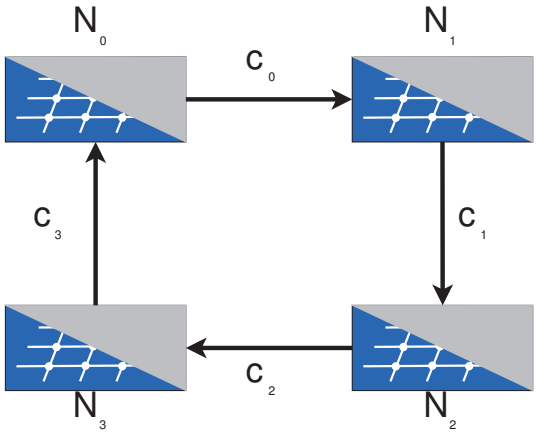


Figure 2.5: Unidirectional ring with four nodes.

happen is the creation of a cyclic channel dependency [25, 77]. Formally, a channel dependency[8] is defined in *Definition* 2.1.1 and *Definition* 2.1.2:

Definition 2.1.1. *A channel dependency between channel c_i and channel c_j occurs when a packet holding channel c_i requests the use of channel c_j.*

Definition 2.1.2. *A channel dependency graph $G = (V, E)$ is a directed graph where the vertices V are the channels of the network N, and the edges E are the pairs of channels (c_i , c_j) such that there exists a (channel) dependency from c_i to c_j.*

Using the ring network from Fig. 2.5 that consists of four nodes N_i where $i = 0, 1, 2, 3$ and a unidirectional channel that connects a pair of adjacent nodes a simple deadlock scenario can be presented. If a packet $p_{0\to2}$ is sent from node N_0 to node N_2, a packet $p_{1\to3}$ is sent from node N_1 to node N_3, a packet $p_{2\to0}$ is sent from node N_2 to node N_0, and a packet $p_{3\to1}$ is sent from node N_3 to node N_1 a cyclic channel dependency forms in which packet $p_{0\to2}$ reserves channel c_0 and requests channel c_1, packet $p_{1\to3}$ reserves channel c_1 and requests channel c_2, packet $p_{2\to0}$ reserves channel c_2 and requests channel c_3, and, finally, packet $p_{3\to1}$ reserves channel c_3 and requests channel c_0. In such a configuration and for such a routing scheme the network will deadlock because each packet

[8] A *cyclic* channel dependency is a cycle of channel dependencies.

reserves a channel and waits for a channel that is reserved by another packet. The following theorem states the sufficient condition for deadlock freedom of a routing function [25]:

Theorem 2.1.3. *A deterministic routing function R for network N is deadlock free if and only if there are no cycles in the channel dependency graph G.*

In case of the network presented in Fig. 2.5 removing the cycles can be achieved by introducing VCs and logically dividing each physical link into two virtual links. Next, the routing function has to be modified so that packets processed by node N_i that are destined to node N_j use the output channel c_{0i} if $i > j$, else use channel c_{1i}. If $i == j$, the packet is consumed[9].

2.2 InfiniBand Architecture

The InfiniBand Architecture (IBA) was first standardized in October 2000 [2], as a combination of two older technologies called Future I/O and Next Generation I/O. As with most other recent interconnection networks, InfiniBand is a serial point-to-point full-duplex technology. It is scalable beyond ten-thousand nodes with multiple CPU cores per node and efficient utilization of host side processing resources. The current trend is that IB is replacing proprietary or lower performance solutions in the high performance computing domain, where high bandwidth and low latency are the key requirements.

The de facto system software for IB is an open-source software stack developed by the OpenFabrics Alliance called OpenFabrics Enterprise Distribution (OFED) [12], which is available for GNU/Linux, Microsoft Windows, Solaris and BSD systems.

InfiniBand networks are referred to as subnets, where a subnet consists of a set of channel adapters interconnected using switches, routers and point-to-point links. An IB fabric consists of one or more subnets, which can be interconnected together using routers. A simple IB fabric is presented on Fig. 2.6. Channel adapters are intelligent network interface cards that consume and generate IB packets. They are used by computation and I/O hosts to attach them to the fabric. Switches and routers relay IB packets from one link to another. Channel adapters switches and routers within a subnet are addressed using local identifiers (LIDs) and a single subnet is limited to 48k LIDs (nodes, switches and local router ports). A LID is a 16 bit value where the first 49151 values are

[9]This example was presented in [25] and [96].

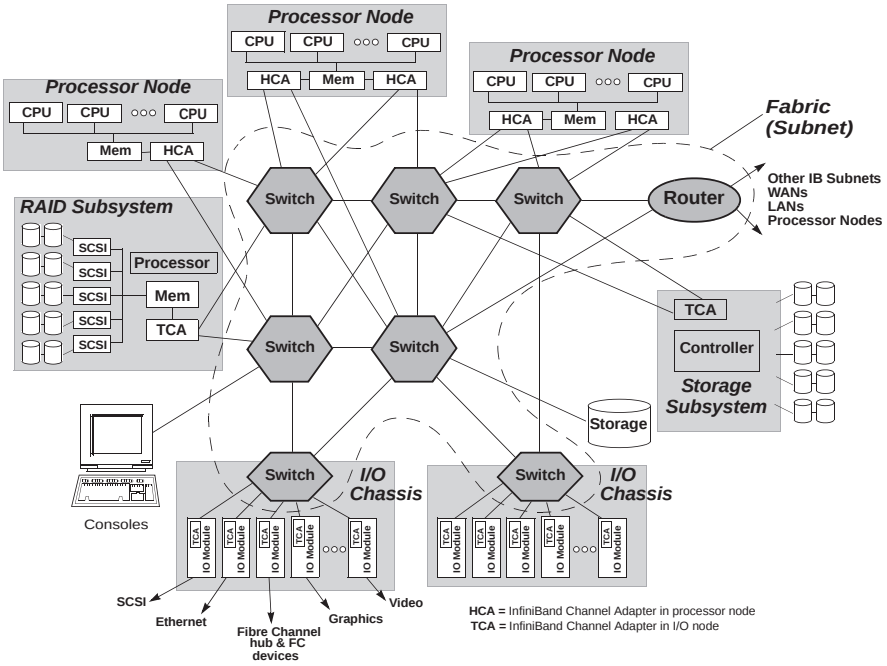


Figure 2.6: IBA System Area Network. Courtesy of IBTA [p. 89] [2].

reserved for unicast addresses and the rest is reserved for multicast. LIDs are local addresses valid only within a subnet, but each IB device also has a 64-bit *global unique identifier* (GUID) burned into its non-volatile memory. A GUID is used to form a GID - an IB layer-3 address. A GID is created by concatenating a 64-bit subnet ID with the 64-bit GUID to form an IPv6-like 128-bit address and such GID addresses are used by IB-IB routers to send packets between IB subnets.

An IB subnet requires one or more subnet managers (SM), which are responsible for initializing and bringing up the network, including the configuration of all the switches, routers, host channel adapters (HCAs) and target channel adapters (TCAs) in the subnet. At the time of initialization the SM starts in the *discovering state* where it does a heavy sweep of the network in order to discover all switches, routers and hosts. When this phase is complete the SM enters the *master state*. In this state, it proceeds with LID assignment, switch

configuration, routing table calculations, and port configuration. If successful, it signals that the *subnet is up*, and then all devices consider the subnet ready for use. After the subnet has been configured, the SM is responsible for monitoring the network for changes.

During normal operation the SM performs periodic *light sweeps* of the network to check for topology changes. If a change is discovered during a light sweep or if a message (trap) signalling a network change is received by the SM, it will reconfigure the network according to the changes discovered. The reconfiguration includes the steps used during initialization. Whenever the network changes (e.g. a link goes down, a device is added or a link is removed) the SM must reconfigure the network accordingly.

2.2.1 InfiniBand Enterprise Systems

In a series of white papers [97], Mellanox Technologies, one of the largest IB hardware manufacturers, has argued that today's data centers need an agile and intelligent infrastructure that can be only delivered by IB. HPC applications aim to make many nodes function like a single computer. IB is an excellent choice for this objective as has been demonstrated by the growing number of IB systems in the Top500 list [11]. However, enterprise systems tend to have other requirements: BigData, heavily virtualized data centers, cloud computing or scale-out web applications. The growing role of IB in enterprise data centers means that IB is able satisfy these demands [98].

Enterprise systems pose a number of challenges to the subnet manager and the routing algorithm. First, in HPC systems, the traffic pattern can be predicted at the time of job scheduling, but data center networks run multiple applications simultaneously, making the traffic pattern impossible to predict [99]. Second, compute nodes have different roles so large traffic imbalances may be created in the network, which means that the routing algorithm has to distinguish between different types of nodes. Third, heavily virtualized networks with flat addressing (where each virtual machine has a layer-2 address) will quickly exhaust LID address space, so address reuse is a highly beneficial feature. Next, studies have demonstrated that utilization of data networks is low [100], so the always-on cabling unnecessarily adds to the power bill. IB adapters consume extremely low power of less than 0.1 watt per gigabit, and IB switches less than 0.03 watts per gigabit, which is less than even IEEE 802.3az Energy Efficient-Ethernet [101] and largest data center operators such as Google have already endorsed IB for its energy savings [99]. However, it is still highly advantageous to deliver high performance for power saving clusters where end nodes are pow-

ered down when not in use and to support fast rerouting on network changes.

2.2.2 Routing in InfiniBand

A major part of the SM's responsibility are the routing table calculations. Routing of the network must be performed in order to guarantee full connectivity, deadlock freedom, and proper load balancing between all source and destination pairs. Routing tables must be calculated at network initialization time and this process must be repeated whenever the topology changes in order to update the routing tables and ensure optimal performance.

The InfiniBand Architecture (IBA) specification clearly distinguishes [2, p. 225] between *routing* and *forwarding*. Forwarding (another term used in the IBA specification is *switching*) is defined as the process of moving a packet from one link to another within a subnet. Routing, on the other hand, is the process of moving a packet from one subnet to another and is accomplished using routers. These definitions match the well-established terms used in TCP/IP networking.

In the scientific community that works with interconnection networks the distinction between *routing* and *switching* is not that evident and those terms are often used interchangeably. The similar lack of distinction is also visible in OFED where the term routing is used to describe algorithms that forward packets within a single subnet. This thesis follows this naming scheme. By a *routing algorithm* we mean an algorithm that controls the selection of the path for a packet regardless whether the packet is forwarded within a subnet or routed between subnets. Furthermore, the distinction between those two terms is only of concern when addressing *RQ4*. Paper V [9], which answers this question, discusses both the intra-subnet routing (forwarding) and inter-subnet routing (routing between subnets).

Currently, IB switches support only Linear Forwarding Tables (LFTs) which map from DLIDs to egress switch ports. For LFTs the value of the DLID is implicit to the position of the entry in the table. Another possibility is to use Random Forwarding Tables (RFTs) which are conceptually the same as Content-Addressable Memory (CAM), however, RFTs are not yet supported in the existing hardware. For RFT each entry consists of a DLID/LMC (LID Control Mask) pair and the egress port associated with them. LMC is used to specify a range of DLIDs from DLID to DLID + $2^{LMC} - 1$. Such a LMC-based scheme was used for MLID fat-tree routing [56], however, it was not implemented in the Open Subnet Manager (OpenSM), the SM available in OFED.

The only routing function supported by today's IB hardware is the incremental routing function of a form $R : N \times N \rightarrow \varphi(C)$ where N is a node and C is a channel. The manner of calculation of the routing function is *centralized* because all processing is done by the SM. The routing function uses a *table-lookup* mechanism to assign an egress port to each packet. In this thesis, all discussed routing algorithms are *deterministic* because adaptive routing algorithms for IB are not yet widely available, are mostly proprietary and are also not well-understood.

OFED stack includes OpenSM, which contains a set of routing algorithms described below[10].

2.2.2.1 Min-Hop

MinHop is the default fallback routing algorithm for the OpenSM, and, as the name suggests, it finds minimal paths among all nodes. Furthermore, MinHop tries to balance the number of routes per link at the local switch. Using MinHop routing often leads to credit loops, which may deadlock the fabric [102].

The MinHop routing comprises of two steps. First, the algorithm calculates the hop distance between each port and each LID using a relaxation algorithm and stores it in a MinHop matrix. A relaxation algorithm, as it traverses through the network, at each step tries to find a shorter path between two nodes than the one it already knows. Second, the algorithm visits each switch and decides which output port should be used to reach a particular LID. This is the balancing step where each port has a counter counting the number of target LIDs routed through it. When there are multiple ports through which a particular LID can be reached, the one with the smaller counter value is selected.

The complexity of MinHop is given by $\mathcal{O}(n^2)$ where n is the number of nodes.

2.2.2.2 Up*/Down*

Up*/Down* [103] (UPDN) is also based on minimum hops to each node, but the paths are constrained by ranking rules. To compute the routing tables, a breadth-first search (BFS) algorithm is run on the graph representing the network. The algorithm, which starts its run on a root node, constructs a spanning tree and marks the links on each node with either *up* or *down* labels. The *up* links are closer to the root of the spanning tree than the *down* links.

[10] Refer to the OpenSM documentation for a detailed description of each routing algorithm. Fat-Tree routing as a whole is covered separately in Section 2.1.4.3.

Up*/Down* defines that a legal route is the one that never uses an *up* link after it has used a *down* link. Such a restriction guarantees that there will never be a deadlock in the network.

2.2.2.3 Dimension Order Routing

Dimension Order Routing (DOR) is a deadlock-free implementation of the MinHop routing algorithm for meshes and hypercubes. DOR does not perform the balancing step (apart from a situation when there are multiple links between the same two switches). Instead of the MinHop's balancing step, DOR chooses from among the available shortest paths based on an ordering of dimensions. DOR routing requires proper cabling across the whole fabric so mesh or hypercube dimension representation is consistent.

DOR builds the paths starting at the destination and continues towards the source using the lowest dimension (port) from all the available paths at each step. For hypercubes, the cabling requirement is that the same port is used throughout the fabric to represent the hypercube dimension and that port must match on both sides of the link. For meshes, the dimension should use the same pair of ports consistently.

2.2.2.4 Torus-2QoS

Torus-2QoS is a routing algorithm designed for large-scale 2D/3D tori fabrics. This algorithm, originally developed at Sandia National Laboratories to route the Red Sky supercomputer [104], is based on DOR and avoids deadlocks that normally occur on DOR-routed tori by using SLs to create a virtual fabric along each torus plane.

Torus-2QoS algorithm has many advanced features such as resiliency to failures and application-transparent self-healing. Furthermore, the remaining SL bits can be used for quality of service, which allows different traffic types to be assigned with their own dedicated network resources.

2.2.2.5 LASH

LAyered SHortest Path (LASH) [86, 105, 106] is a deterministic shortest path routing algorithm for irregular networks. All packets are routed using the minimal path, and the algorithm achieves deadlock freedom by finding and breaking cycles by using virtual lanes. LASH routing comprises of three distinct phases.

First, the shortest paths are found between all pairs of sources and destinations. Second, for each route LASH assigns Service Levels (SLs), which will

be later mapped to VLs. A route can only be assigned to a particular SL if the presence of that route on that SL will not cause a deadlock within that layer. Once a route is found whose addition to a layer would cause a deadlock, a new layer is started and the assignment is continued. Third, the balancing step is performed. It is more likely that first layers will contain more routes than the latter ones, so the number of paths per layer is averaged by moving them between the layers.

The disadvantage of LASH is that it does not balance the traffic well, which is especially evident in fat-tree fabrics. The algorithm aims at avoiding the deadlock by using the lowest possible number of VLs. Assuming no turn restrictions, the only deadlock-free logical topology within a physical fat-tree is a single-rooted tree, however, it also has the lowest performance due to the lowest path diversity.

The computing complexity for LASH is $\mathcal{O}(n^3)$ where n is the number of nodes, which makes the algorithm unusable for very large fabrics.

2.2.2.6 DFSSSP

Deadlock-Free Single-Source-Shortest-Path routing (DFSSSP) [87] is an efficient oblivious routing for arbitrary topologies developed by Domke et al. It uses virtual lanes to guarantee deadlock freedom and, in comparison to LASH, aims at not limiting the number of possible paths during the routing process. It also uses improved heuristics to reduce the number of used virtual lanes in comparison to LASH.

The problem with DFSSSP was that for switch-to-switch traffic it assumed deadlock freedom, and did not break any cycles that occurred for switch-to-node and switch-to-switch pairs[11]. The computing complexity for the offline DFSSSP is $\mathcal{O}(n^2 \cdot \log(n))$ where n is the number of nodes [87].

[11] This bug was fixed in a recent patch, but was an issue when Paper III and Paper IV were researched.

Chapter 3

Summary of Research Papers

This chapter presents the six research papers that were written as a part of this thesis. The contributions from each paper are summarized in the relevant abstract. Each of the research papers tackles a research problem listed in Chapter 1.1. Four of the papers were published at peer-reviewed international conferences, one paper was published in the ACM TACO journal, and the last paper was accepted to the 22nd Euromicro International Conference on Parallel, Distributed and Network-Based Processing.

Paper I [5] studies the problem of non-optimal routing in topologies where switch ports are not fully populated with end nodes and we propose a solution that alleviates the counter-intuitive performance drop for such fabrics.

In Paper II [6] we study and propose a cheap alternative to congestion control. By using multiple virtual lanes, we are able to remove head-of-line blocking and parking lot problems in fat-tree fabrics, which significantly improves the network performance for hotspot traffic patterns.

Paper III [7] focuses on ease-of-management and availability concepts. By making sure that each device in the fat-tree fabric is reachable in a deadlock-free manner, it became possible to run the subnet manager on the spine switches. Moreover, IB diagnostic tools relying on full connectivity for basic fabric management and monitoring could be run from any network node. Lastly, IP over IB also achieved full reachability, which was required by non-InfiniBand aware management and monitoring protocols like SNMP or SSH.

In the first three papers, we address *RQ1: How should the InfiniBand fat-tree routing be extended to support modern enterprise systems?* and *RQ2: What network resources can be used to achieve high routing performance and full reachability?*.

Paper IV [8] and Paper VI [10] both treat the topic of fault-tolerance in fat-trees. The former paper proposes extensions to the fat-tree routing algorithm to make it less prone to failure for non-standard topologies, and the latter paper presents a new routing logic that introduces a new level of fault-tolerance by making sure that the path to the same node that has two different ports never contains a single point of failure. Both Paper IV and Paper VI address *RQ3: How to make the fat-tree routing more resilient to switch failures?* and Paper VI also addresses *RQ1: How should the InfiniBand fat-tree routing be extended to support modern enterprise systems?*.

Lastly, in Paper V [9], we focus on inter-subnet routing. Very little actual work was done for native IB-IB routing and most of it concerned disaster recovery. To the best of our knowledge, we present the first two advanced inter-subnet routing algorithms for InfiniBand. Having separate subnets increases fabric scalability and allows the reuse of layer-2 addresses, which may be beneficial for heavily virtualized systems. This papers provides a unified solution for designing and routing a multi-subnet IB fabric. This paper addresses *RQ4: What are the requirements for the InfiniBand inter-subnet routing algorithms and how these algorithms can use the local routing information supplied by intra-subnet routing algorithms?*.

3.1 Paper I: Achieving Predictable High Performance in Imbalanced Fat Trees

In this paper, we studied a performance issue often encountered during cluster construction, for underpopulated clusters ready for future expansion (a common scenario), for power saving clusters where end nodes are powered down when not in use and also in enterprise systems that are limited by the number of rack units and often cannot utilize all the ports on a densely packed IB switch.

The problem is that the classic fat-tree routing algorithm designed by Zahavi [14] behaves counter-intuitively when the number of end nodes in a tree decreases. Decreasing the number of end nodes in a fat-tree while keeping the number of connections between the switches constant should decrease link contention when routing packets in the upward direction, however, due to flaws

in Zahavi's algorithm, that contention is not only not decreased, but also the downward paths from the spine switches are no longer dedicated and are shared between multiple destinations. The severity of the problem depends on the distribution of the end nodes and the size of the tree.

To address this problem, we present an extension to the fat-tree routing algorithm that completely removes this problem. First, we study the behaviour of the classic fat-tree routing algorithm to identify its weaknesses and extend it by adding a balancing step that makes the performance independent of the number and the distribution of the end nodes. The problem with Zahavi's implementation are the dummy end nodes that are introduced to ensure a correct distribution of the downward paths in the network with respect to an all-to-all shift communication pattern. While this leads to an apparently balanced network, it may lead to skew in the balance of the upward paths.

The methodology that was applied to solve this problem was to look beyond the local routing information, that is, investigate the loads of downward channels on switches at higher tiers than the one that is currently being processed and perform an additional balancing step that distributed the paths evenly across the switches.

By using a fixed set of synthetic traffic patterns combined with HPCC Benchmark [107] simulations, we show that our extensions improve performance by up to 30% depending on topology size and node distribution. Furthermore, the simulations demonstrate that our algorithm allows to achieve a high predictable throughput irrespective of the node distribution and the node number.

3.2 Paper II: vFtree - A Fat-tree Routing Algorithm using Virtual Lanes to Alleviate Congestion

In this paper, we continued the work on improving the performance of the fat-tree routing algorithm. The problem we addressed was related to network congestion occurring due to hotspots.

Even though the bisection bandwidth in a fat-tree is constant, hotspots are still possible and they will degrade performance for flows not contributing to them due to head-of-line blocking and parking lot problem. Such a situation may be alleviated through adaptive routing or congestion control, however, these methods are not yet readily available in IB technology, are often proprietary and are not well understood [108, 109].

It is a well known fact [15, 77] that multiple virtual lanes can improve performance in interconnection networks, but this knowledge has had little impact on real clusters. Currently, a large number of clusters using InfiniBand is based on fat-tree topologies that can be routed deadlock-free using only one virtual lane. Consequently, all the remaining virtual lanes are left unused.

To remedy this problem, we have implemented an enhanced fat-tree algorithm in OpenSM that distributes traffic across all available virtual lanes without any configuration needed. Our algorithm works on top of the classic fat-tree routing [14] and distributes all destinations sharing the same upward link across different virtual lanes. This means that if one of the destinations is the contributor to an endpoint hotspot, other destination flows sharing the same upward port will not affected by head-of-line blocking because they use a different virtual lane with its own buffering resources.

We evaluated the performance of the algorithm on a small cluster and did a large-scale evaluation through simulations. We demonstrated that by extending the fat-tree routing with VLs, we are able to dramatically improve the network performance in presence of hotspots. We achieved improvements from 38% for small hardware topologies to 757% for large-scale simulated clusters when compared with the conventional fat-tree routing.

3.3 Paper III: sFtree: A Fully Connected and Deadlock-Free Switch-to-Switch Routing Algorithm for Fat-Trees

In this paper, we studied the problem of full reachability in fat-trees. It is a well known fact that existing fat-tree routing algorithms fully exploit the path diversity of a fat-tree topology in the context of end node traffic, but very little research was done into the switch-to-switch traffic. In fact, the switch-to-switch communication has been ignored for a long time because the switches themselves lacked the necessary intelligence to be able to support advanced system management techniques, and originated very little traffic.

In recent years, with the general increase in system management capabilities found in modern InfiniBand switches, the lack of deadlock-free switch-to-switch communication became a problem for fat-tree based IB installations because management traffic might cause routing deadlocks that bring the whole system down. This lack of deadlock-free communication affects all system management and diagnostic tools using LID routing.

In this paper we present, to the best of our knowledge, the first fat-tree routing algorithm that supports deadlock-free and fully connected switch-to-switch routing. Our approach retains all the performance characteristics of the algorithm presented by Zahavi [14] and it is evaluated on a working prototype tested on commercially available IB technology. Furthermore, our sFtree algorithm fully supports all types of single and multi-core fat-trees commonly encountered in commercial systems.

Our solution to the deadlock-free switch-to-switch routing is based on a concept of a *subtree*. In this paper, we propose a binary tree called subtree whose root is one of the leaf switches and whose leaves are all the spines in the topology. Such a subtree allows us to localize all the normally prohibited down/up turns in a fabric to a single deadlock-free tree. Accommodating all the prohibited turns in a subtree is deadlock-free, which is formally proven in the paper, and it permits full connectivity between all switches if a subtree root is chosen properly.

We evaluate the performance of the sFtree algorithm experimentally on a small cluster and we do a large-scale evaluation through simulations. The results confirm that the sFtree routing algorithm is deadlock-free and show that the impact of switch-to-switch management traffic on the end-node traffic is negligible.

3.4 Paper IV: Discovery and Routing of Degraded Fat-Trees

In this paper, we studied the fault-resilience of the fat-tree routing algorithm and compared the performance achieved by the fat-tree routing with three other major algorithms available in OpenSM when routing a severely degraded topologies.

The main contribution of this paper is the extension of the fat-tree algorithm that liberalizes the restrictions imposed on the fat-tree discovery and routing of degraded fabrics. First, we liberalized topology validation to make fat-tree routing more versatile. Second, we proposed a switch tagging mechanism through vendor SMP attributes that can be queried via vendor specific SMPs and are used to configure the switches with specific fabric roles, which decouples topology discovery from actual routing.

The enhancements presented in this paper allowed us to solve the flipping switch anomaly that occurs for leaf switches that have no end nodes connected.

Without our fixes, such a leaf switch will be treated as a switch that is located 2 levels higher than it really is because the routing algorithm loses the sense of directions that connect to such a switch.

We compared four non-proprietary routing algorithms running on a degraded 3-stage 648-port fat-tree. By using a fixed set of synthetic traffic patterns combined with HPCC Benchmark [107] simulations, we showed that the fat-tree routing is still the preferred algorithm even if the number of failures is very large. We also demonstrated that even though the fat-tree routing is the most susceptible (with largest performance drops) algorithm to device and link failures, it still delivers the highest performance even in extreme cases for non-trivial traffic patterns.

3.5 Paper V: Making the Network Scalable: Inter-subnet Routing in InfiniBand

In this paper, we examine and solve the routing problems that occur when using native IB-IB routers, and we introduce and analyse two new routing algorithms for inter-subnet IB routing: inter-subnet source routing (ISSR) and inter-subnet fat-tree routing (ISFR).

As InfiniBand-based clusters grow in size and complexity, the need arises to segment the network into manageable sections. Up until now, InfiniBand routers were not used extensively and little research was done to accommodate them. However, the limits imposed on local addressing space, inability to logically segment the fabrics, long reconfiguration times for large fabrics in case of faults, and, finally, performance issues when interconnecting large clusters, have rekindled the industry's interest into IB-IB routers. In the context of supercomputing the path to improved network scalability does not necessarily lie in subnetting, and a single subnet approach is the preferred solution for HPC applications, for cloud computing, data warehousing, enterprise systems, in case of hybrid topologies (for seamless interconnection), or in heavily virtualized systems there is a need to partition the network into self-contained and independently administered domains.

Our algorithms are implemented in SM and require additional support from router's firmware. Namely, ISSR is a deterministic oblivious routing algorithm that uses a simple hashing mechanism to generate the output port for each source-destination pair. ISFR goes a step further and requires that the router delivers input to its local SM as to how configure the local intra-subnet routing.

This input is based on the intra-subnet routing tables in the remote subnet.

Through simulations, we show that both ISSR and ISFR clearly outperform current implementation of the inter-subnet routing available in OpenSM. By varying the number of subnets and the percentage of traffic that is sent between the subnets, we are able to observe that for congested traffic patterns, ISSR obtains slightly higher performance than ISFR, but when the number of subnets increases, ISFR delivers almost equal performance.

By laying the groundwork for layer-3 routing, we show that native IB-IB routers can make the network scalable, and that designing efficient routing algorithms is the first step towards that goal. Future work includes solving the deadlock problem in multi-subnet environment.

3.6 Paper VI: Multi-homed Fat-Tree Routing with InfiniBand

In this paper, we studied the fault-resilience of the fat-tree routing algorithm for multi-homed end nodes. We discovered that even though an end node may have two ports connected to the same fabric, there still exists a single point of failure and we decided to solve this problem by proposing our own fat-tree routing algorithm - mFtree.

One of the missing properties of the fat-tree routing algorithm is that there is no guarantee that each port on a multi-homed node is routed through redundant spines, even if these ports are connected to redundant leaves. As a consequence, in case of a spine failure, there is a small window where the node is unreachable until the subnet manager has rerouted to another spine.

The current fat-tree routing is oblivious whether ports belong to the same node or not. This makes the routing depend on the cabling and may be very misleading to the fabric administrator, especially when recabling is not possible due to a fixed cable length as often happens in enterprise IB systems. Furthermore, this requires the fabric designers to connect the end-nodes in such a way that will make the fat-tree routing algorithm route them through independent paths. Whereas this is a simple task for very small fabrics, when a fabric grows, it quickly becomes infeasible. Additionally, the scheme will break in case of any failure as usually the first thing that the maintenance does when a cable or a port does not work, is to reconnect the cable to another port, which changes the routing.

In this paper, we presented the new mFtree routing algorithm. By means

of simulations, showed that it may improve the network performance compared to the current OpenSM fat-tree routing by up to 52.6% depending on the traffic pattern. Most importantly, however, mFtree routing algorithm gives much better redundancy than classic fat-tree routing, which means that multi-homed nodes will suffer no downtime in case of switch failures.

Chapter 4

Conclusions

This chapter presents suggestions for future research and concludes the thesis. This thesis set out to explore optimized fat-tree routing for enterprise systems. We started by pointing out weaknesses in the current fat-tree routing algorithm, and we continued with providing multiple enhancements to it that solved crucial problems encountered in InfiniBand systems. Nonetheless, even a larger set of more interesting challenges lie ahead and deserve further study. These challenges are briefly presented below.

4.1 Future Directions

There were many vital research topics that were not covered in this thesis. One of such topics becomes evident when routing modern database systems. Such a system is constructed using cell storage nodes that generate the majority of the traffic and database server nodes that are the traffic consumers whose role is to process the data received from the storage nodes. Because a traffic pattern for such a system is easy to predict, a routing algorithm treating each node obliviously will not offer the maximum performance. Therefore, further work needs to be done to extend the current routing so it automatically detects the node type and properly balances the paths in the network. Such intelligent routing could be accomplished by using proprietary vendor attributes similarly to the solution presented in Paper IV [8]. However, the challenging aspect of

such a solution is to maintain fault-tolerance in active-passive systems[1] such as the ones described in Paper VI [10].

Next, as mentioned in Paper V [9], there exists the issue of generalized deadlock-free inter-subnet routing. Even though our current solution supports many different regular fabrics in a deadlock-free manner by interconnecting them using a fat-tree backbone, real scalability could be achieved by supporting transition subnets, that is, subnets which interconnect other subnets and deadlock-free inter-subnet all-to-all switch-to-switch routing algorithms based on the concept described in Paper III [7]. Furthermore, in our work we only allow fat-tree subnet topologies, however, it is essential for other popular interconnection topologies to be supported as subnets as well.

Another area that requires attention is evaluating the hardware design alternatives for native IB-IB routers. Paper V [9] covers only the area of unicast routing algorithms. However, other valid research topics include inter-subnet multicast routing, optimizing the hardware cost of content-addressable memory required for GUID to LID mappings, inter-subnet state coordination through a super subnet manager, long-haul link support or encoding local state information in the subnet prefix.

Lastly, increased virtualization poses additional challenges to the routing algorithm. Assuming that subnetting IB networks will allow a flat addressing space for virtual machines, frequent virtual machine migrations may lead to frequent and unnecessary network reconfigurations negatively influencing the performance of the subnet manager. By sacrificing the number of virtual machines to be limited to 128 at each leaf switch, LID Mask Control (LMC) could be used to make a migration within a leaf switch transparent to all the other nodes by only reconfiguring the routing table at the switch where the migration takes place. This could be later extended to support more advanced scenarios such as transparently migrating a router port and all inter-subnet paths associated with it from one router to another in such a way that no reconfiguration occurs.

[1] In such systems, a two port node has one port active and other port passive (standby). This occurs due to 8x PCI Express 2.0 limitations for Quad Data Rate (QDR) and 8x PCI Express 3.0 limitations for Enhanced Data Rate (EDR). EDR is currently the fastest IB standard.

4.2 Concluding Remarks

In this thesis, we have considered three major challenges that are encountered when routing enterprise systems: performance, scalability and fault-tolerance. To this end, we have presented six research papers in which we discussed and evaluated a number of routing enhancements for the fat-tree routing. We showed that by improving the routing algorithm we obtain large gains system-wide because we solve scalability issues as showed in Paper III and V, reliability as presented in Papers IV and VI, and performance limitations as demonstrated in Papers I and II.

The contributions to this thesis are a set of five extended routing algorithms that build upon the original fat-tree routing by Zahavi [14]. Another large contribution is, to the best of our knowledge, the first two advanced inter-subnet routing algorithms for IB. The usability of these solutions can be best illustrated by the fact that the algorithms described in Paper III, Paper IV and Paper VI are already implemented in all InfiniBand systems manufactured by Oracle Corporation. Still, given the strict requirements for enterprise systems whose performance depends upon finely tuned parameters, we believe that there is a large set of improvements that this thesis does not discuss.

Bibliography

[1] Eitan Zahavi. D-Mod-K Routing Providing Non-Blocking Traffic for Shift Permutations on Real Life Fat Trees. `http://www.technion.ac.il/~ezahavi/`, September 2010.

[2] InfiniBand™ Architecture Specification, November 2007.

[3] Charles Eames, Ray Eames, and International Business Machines Corporation. *A Computer Perspective: Background to the Computer Age*. Harvard University Press, 1973.

[4] Ahmad Chadi Aljundi, Jean-Luc Dekeyser, Tahar Kechadi, and Isaac Scherson. A Study of an Evaluation Methodology for Unbuffered Multistage Interconnection Networks. In *Proceedings of the 17th International Parallel and Distributed Processing Symposium*, pages 8 pp.–, 2003.

[5] Bartosz Bogdański, Frank Olaf Sem-Jacobsen, Sven-Arne Reinemo, Tor Skeie, Line Holen, and Lars Paul Huse. Achieving Predictable High Performance in Imbalanced Fat Trees. In *Proceedings of the 16th IEEE International Conference on Parallel and Distributed Systems*, pages 381–388. IEEE Computer Society, 2010.

[6] Wei Lin Guay, Bartosz Bogdański, Sven-Arne Reinemo, Olav Lysne, and Tor Skeie. vFtree - A Fat-Tree Routing Algorithm Using Virtual Lanes to Alleviate Congestion. In *Proceedings of the 25th International Parallel and Distributed Processing Symposium*, pages 197–208. IEEE Computer Society, 2011.

[7] Bartosz Bogdański, Sven-Arne Reinemo, Frank Olaf Sem-Jacobsen, and Ernst Gunnar Gran. sFtree: A Fully Connected and Deadlock Free

Switch-to-Switch Routing Algorithm for Fat-Trees. *ACM Transactions on Architecture and Code Optimization*, 8(4), January 2012.

[8] Bartosz Bogdański, Bjørn Dag Johnsen, Sven-Arne Reinemo, and Frank Olaf Sem-Jacobsen. Discovery and Routing of Degraded Fat-Trees. In *Proceedings of the 13th International Conference on Parallel and Distributed Computing, Applications and Technologies*, pages 689–694. IEEE Computer Society, December 2012.

[9] Bartosz Bogdański, Bjørn Dag Johnsen, Sven-Arne Reinemo, and José Flich. Making the Network Scalable: Inter-subnet Routing in InfiniBand. In *Proceedings of the 19th International Euro-Par Conference on Parallel Processing*, volume 8097 of *Lecture Notes in Computer Science*, pages 685–698. Springer Berlin Heidelberg, 2013.

[10] Bartosz Bogdański, Bjørn Dag Johnsen, and Sven-Arne Reinemo. Multihomed Fat-Tree Routing with InfiniBand. In *Accepted to Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, Euromicro PDP 2014. IEEE Computer Society Press, February 2014.

[11] Top 500 supercomputer sites. `http://top500.org/`, November 2013. Retrieved November 21, 2013.

[12] The OpenFabrics Alliance. `http://openfabrics.org/`. Retrieved July 31, 2013.

[13] Crispín Gómez, Francisco Gilabert, Maria Gómez, Pedro López, and José Duato. Deterministic versus Adaptive Routing in Fat-Trees. In *Proceedings of the 21st International Parallel and Distributed Processing Symposium*, pages 1–8, March 2007.

[14] Eitan Zahavi, Gregory Johnson, Darren Kerbyson, and Michael Lang. Optimized InfiniBand^{TM}Fat-Tree Routing for Shift All-to-All Communication Patterns. *Concurrency and Computation: Practice and Experience*, 22(2):217–231, February 2010.

[15] William Dally. Virtual-Channel Flow Control. *IEEE Transactions on Parallel and Distributed Systems*, 3(2):194–205, March 1992.

[16] José Flich, Pedro López, José Carlos Sancho, Antonio Robles, and José Duato. Improving InfiniBand Routing through Multiple Virtual Networks.

In *Proceedings of the 4th International Symposium on High Performance Computing*, volume 2327 of *Lecture Notes in Computer Science*, pages 49–63. Springer Berlin Heidelberg, January 2002.

[17] Robert Stebbins. *Exploratory Research in the Social Sciences*. Qualitative Research Methods. SAGE Publications, 2001.

[18] Gordana Dodig Crnkovic. Constructive Research and Info-computational Knowledge Generation. In *Model-Based Reasoning in Science and Technology*, volume 314 of *Studies in Computational Intelligence*, pages 359–380. Springer Berlin Heidelberg, 2010.

[19] Voltaire infiniband fabric simulator - ibsim. `https://github.com/jgunthorpe/ibsim`, October 2011. Retrieved August 5, 2013.

[20] Infiniband fabric management utilitie - ibutils. `http://www.openfabrics.org/`, April 2013. Retrieved August 5, 2013.

[21] UML Resource Page. `http://www.uml.org/`, August 2013. Retrieved August 5, 2013.

[22] Andras Varga. Parameterized Topologies for Simulation Programs. In *WMC'98: Western Multiconference on Simulation, CNDS'98: Communication Networks and Distributed Systems*, 1998.

[23] Ernst Gunnar Gran and Sven-Arne Reinemo. InfiniBand Congestion Control, Modelling and validation. In *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, 2011.

[24] Charles Spurgeon. *Ethernet: The Definitive Guide*. O'Reilly and Associates, Inc., 2000.

[25] José Duato, Sudhakar Yalamanchili, and Ni Lionel. *Interconnection Networks: An Engineering Approach*. Morgan Kaufmann Publishers Inc., 2003.

[26] William Dally and Brian Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., 2003.

[27] Sven-Arne Reinemo. *Quality of Service in Interconnection Networks*. PhD thesis, University of Oslo, September 2007.

[28] Andrew Tanenbaum. *Computer Networks*. Prentice Hall Professional Technical Reference, 5th edition, October 2010.

[29] William Dally. Performance Analysis of k-ary n-cube Interconnection Networks. *IEEE Transactions on Computers*, 39(6):775–785, June 1990.

[30] Ankit Singla, Chi-Yao Hong, Lucian Popa, and P. Brighten Godfrey. Jellyfish: Networking Data Centers Randomly. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, pages 17–17. USENIX Association, 2012.

[31] Charles Clos. A Study of Non-blocking Switching Networks. *Bell Systems Technical Journal*, 32:406–424, 1953.

[32] Václav Beneš. *Mathematical Theory of Connecting Networks and Telephone Traffic*. Mathematics in science and engineering : a series of monographs and textbooks. Academic Press, 1965.

[33] J.H. Patel. Performance of Processor-Memory Interconnections for Multiprocessors. *IEEE Transactions on Computers*, C-30(10):771–780, 1981.

[34] John Kim, William Dally, and Dennis Abts. Flattened Butterfly: A Cost-Efficient Topology for High-Radix Networks. In *Proceedings of the 34th Annual International Symposium on Computer Architecture*, pages 126–137. ACM, 2007.

[35] Charles Leiserson. Fat-trees: Universal Networks for Hardware-Efficient Supercomputing. *IEEE Transactions on Computers*, C-34(10):892–901, 1985.

[36] John Kim, William Dally, Steve Scott, and Dennis Abts. Cost-Efficient Dragonfly Topology for Large-Scale Systems. *IEEE Micro*, 29(1):33–40, 2009.

[37] Ted H. Szymanski. "Hypermeshes": Optical Intercomnnection Network for Parallel Computing. *Journal of Parallel and Distributed Computing*, 26(1):1–23, 1995.

[38] Oracle engineered systems. `http://www.oracle.com/us/products/engineered-systems/index.html`, 2013. Retrieved September 19, 2013.

[39] Yuuichirou Ajima, Shinji Sumimoto, and Toshiyuki Shimizu. Tofu: A 6D Mesh/Torus Interconnect for Exascale Computers. *Computer*, 42(11):36–40, 2009.

[40] Frank Olaf Sem-Jacobsen, Tor Skeie, Olav Lysne, and José Duato. Dynamic Fault Tolerance in Fat Trees. *IEEE Transactions on Computers*, 60(4):508–525, 2011.

[41] Frank Olaf Sem-Jacobsen and Olav Lysne. Fault Tolerance with Shortest Paths in Regular and Irregular Networks. In *Proceedings of the 22nd International Parallel and Distributed Processing Symposium*. IEEE Computer Society, April 2008.

[42] Frank Olaf Sem-Jacobsen, Tor Skeie, Olav Lysne, Ola Tørudbakken, Eivind Rongved, and Bjørn Dag Johnsen. Siamese-Twin: A Dynamically Fault-Tolerant Fat Tree. In *Proceedings of the 19th International Parallel and Distributed Processing Symposium*. IEEE Computer Society, 2005.

[43] Frank Olaf Sem-Jacobsen, Tor Skeie, and Olav Lysne. A Dynamic Fault-tolerant Routing Algorithm for Fat-trees. In *Proceedings of the 11th International Conference on Parallel and Distributed Processing Techniques and Applications*, pages 318–324. CSREA Press, June 2005.

[44] Frank Olaf Sem-Jacobsen, Tor Skeie, Olav Lysne, and José Duato. Dynamic Fault Tolerance with Misrouting in Fat Trees. In *Proceedings of the 35th International Conference on Parallel Processing*, pages 33–45. IEEE Computer Society, August 2006.

[45] Frank Olaf Sem-Jacobsen, Olav Lysne, and Tor Skeie. Combining Source Routing and Dynamic Fault Tolerance. In *Proceedings of the 18th International Symposium on Computer Architecture and High Performance Computing*, pages 151–158. IEEE Computer Society, October 2006.

[46] Frank Olaf Sem-Jacobsen, Åshild Grønstad Solheim, Olav Lysne, Tor Skeie, and Thomas Sødring. Efficient and Contention-Free Virtualisation of Fat-Trees. In *Proceedings of the 25th International Parallel and Distributed Processing Symposium*, pages 754–760. IEEE Computer Society, May 2011.

[47] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A Scalable, Commodity Data Center Network Architecture. In *Proceedings of the*

ACM SIGCOMM 2008 Conference on Data Communication, SIGCOMM, pages 63–74. ACM, 2008.

[48] Radhika Niranjan Mysore, Andreas Pamboris, Nathan Farrington, Nelson Huang, Pardis Miri, Sivasankar Radhakrishnan, Vikram Subramanya, and Amin Vahdat. PortLand: A Scalable Fault-Tolerant Layer 2 Data Center Network Fabric. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, SIGCOMM, pages 39–50. ACM, 2009.

[49] Albert Greenberg, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. Towards a Next Generation Data Center Architecture: Scalability and Commoditization. In *Proceedings of the ACM Workshop on Programmable Routers for Extensible Services of Tomorrow*, pages 57–62. ACM, 2008.

[50] Brent Stephens, Alan Cox, Wes Felter, Colin Dixon, and John Carter. PAST: Scalable Ethernet for Data Centers. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pages 49–60. ACM, 2012.

[51] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. VL2: A Scalable and Flexible Data Center Network. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, SIGCOMM, pages 51–62. ACM, 2009.

[52] Charles Leiserson and Bruce Maggs. Communication-Efficient Parallel Algorithms for Distributed Random-access Machines. *Algorithmica*, 3(1-4):53–77, 1988.

[53] Tom Leighton, Bruce Maggs, and Satish Rao. Universal Packet Routing Algorithms. In *29th Annual Symposium on Foundations of Computer Science*, pages 256–269, 1988.

[54] Charles Leiserson, Zahiand Abuhamdeh, David Douglas, Carl Feynman, Mahesh Ganmukhi, Jeffrey Hill, Daniel Hillis, Bradley Kuszmaul, Margaret St. Pierre, David Wells, Monica Wong-chan, Shaw-wen Yang, and Robert Zak. The Network Architecture of the Connection Machine CM-5. *Journal of Parallel and Distributed Computing*, pages 272–285, 1992.

[55] Nathan Farrington, Erik Rubow, and Amin Vahdat. Data Center Switch Architecture in the Age of Merchant Silicon. In *Proceedings of the 17th IEEE Symposium on High-Performance Interconnects*, August 2009.

[56] Xuan-Yi Lin, Yeh-Ching Chung, and Tai-Yi Huang. A Multiple LID Routing Scheme for Fat-Tree-based InfiniBand Networks. In *Proceedings of the 18th International Parallel and Distributed Processing Symposium*, pages 11–, 2004.

[57] Fabrizio Petrini and Marco Vanneschi. k-ary n-trees: High Performance Networks for Massively Parallel Architectures. In *Proceedings of the 11th International Parallel Processing Symposium*, pages 87–93, 1997.

[58] Frank Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. Morgan Kaufmann Publishers, 1992.

[59] Sabine Öhring, Maximilian Ibel, Sajal Das, and Mohan Kumar. On Generalized Fat Trees. In *Proceedings of 9th International Parallel Processing Symposium*, pages 37–44, 1995.

[60] Marcos Valerio, Louise Moser, and Michael Melliar-Smith. Recursively Scalable Fat-Trees as Interconnection Networks. In *IEEE 13th Annual International Phoenix Conference on Computers and Communications*, pages 40–, 1994.

[61] Marcos Valerio, Louise Moser, and Michael Melliar-Smith. Fault-Tolerant Orthogonal Fat-Trees as Interconnection Networks. In *Proceedings of the 1st International Conference on Algorithms and Architectures for Parallel Processing*, volume 2, pages 749–754 vol.2, April 1995.

[62] Cyriel Minkenberg, Ronald Luijten, and Germán Rodríguez. On the Optimum Switch Radix in Fat Tree Networks. In *High Performance Switching and Routing (HPSR), 2011 IEEE 12th International Conference on*, pages 44–51, 2011.

[63] Sven-Arne Reinemo, Frank Olaf Sem-Jacobsen, and Tor Skeie. Fat-trees and Dragonflies - A Perspective on Topologies. Contributed talk at the HPC Advisory Council Switzerland Workshop, Lugano, Switzerland., March 2012.

[64] Gilad Shainer. Networks: Topologies - How to Design? Contributed talk at the HPC Advisory Council Switzerland Workshop, Lugano, Switzerland., March 2011.

[65] HPC Fabric Analysis - Designed for Reduced Costs and Improved Performance. `http://www.qlogic.com/OEMPartnerships/HP/Documents/hpc/wp_4AA2-3765ENW.pdf`, July 2010. Retrieved November 08, 2013.

[66] Ronald Greenberg. The Fat-Pyramid: A Robust Network for Parallel Computation. In *Advanced Research in VLSI: Proceedings of the Sixth MIT Conference*, pages 195–213. MIT Press, 1990.

[67] Ronald Greenberg. The Fat-Pyramid and Universal Parallel Computation Independent of Wire Delay. *IEEE Transactions on Computers*, 43:1358–1364, 1994.

[68] Kevin Chen and Edwin Sha. The Fat-stack and Universal Routing in Interconnection Networks. *Journal of Parallel and Distributed Computing*, 66(5):705–715, May 2006.

[69] Javier Navaridas, Jose Miguel-Alonso, Francisco Javier Ridruejo, and Wolfgang Denzel. Reducing Complexity in Tree-Like Computer Interconnection Networks. *Parallel Computing*, 36(2-3):71–85, February 2010.

[70] Randy Bush and David Meyer. Some Internet Architectural Guidelines and Philosophy. RFC 3439, December 2002.

[71] Paul Baran. On Distributed Communications. August 1964.

[72] Parviz Kermani and Leonard Kleinrock. Virtual Cut-Through: A New Computer Communication Switching Technique. *Computer Networks*, 3:267–286, 1979.

[73] William Dally and Charles Seitz. Deadlock-Free Message Routing in Multiprocessor Interconnection Networks. *IEEE Transactions on Computers*, C-36(5):547–553, 1987.

[74] Seitz Charles et al. Wormhole Chip Project Report, 1985.

[75] José Duato. A New theory of Deadlock-Free Adaptive Routing in Wormhole Networks. *IEEE Transactions on Parallel and Distributed Systems*, 4(12):1320–1331, 1993.

[76] José Duato. A Necessary and Sufficient Condition for Deadlock-Free Adaptive Routing in Wormhole Networks. *IEEE Transactions on Parallel and Distributed Systems*, 6(10):1055–1067, 1995.

[77] William Dally and Charles Seitz. Deadlock-Free Message Routing in Multiprocessor Interconnection Networks. *IEEE Transactions on Computers*, 36(5):547–553, May 1987.

[78] Wei Lin Guay, Sven-Arne Reinemo, Olav Lysne, and Tor Skeie. dFtree: A Fat-Tree Routing Algorithm Using Dynamic Allocation of Virtual Lanes to Alleviate Congestion in InfiniBand Networks. In *Proceedings of the 1st International Workshop on Network-Aware Data Management*, NDM '11, pages 1–10. ACM, 2011.

[79] Mischa Schwartz and Thomas Stern. Routing Techniques Used in Computer Communication Networks. *IEEE Transactions on Communications*, 28(4):539–552, 1980.

[80] Paul Bell and Kamal Jabbour. Review of Point-to-Point Network Routing Algorithms. *IEEE Communications Magazine*, 24(1):34–38, 1986.

[81] Harry Rudin. On Routing and Delta Routing: A Taxonomy and Performance Comparison of Techniques for Packet-Switched Networks. *IEEE Transactions on Communications*, 24(1):43–59, 1976.

[82] Wai Sum Lai. Packet forwarding. *IEEE Communications Magazine*, 26(7):8–17, 1988.

[83] Narsingh Deo and Chi-Yin Pang. Shortest-path Algorithms: Taxonomy and Annotation. *Networks*, 14(2):275–323, 1984.

[84] Mischa Schwartz and Thomas Stern. *Routing Protocols*. Applications of Communications Theory. Springer US, 1982.

[85] Patrick Gaughan and Sudhakar Yalamanchili. Adaptive Routing Protocols for Hypercube Interconnection Networks. *Computer*, 26(5):12–23, May 1993.

[86] Tor Skeie, Olav Lysne, and Ingebjørg Theiss. Layered Shortest Path (LASH) Routing in Irregular System Area Networks. In *Proceedings of the 16th International of the Parallel and Distributed Processing Symposium*, pages 8 pp–, 2002.

[87] Jens Domke, Torsten Hoefler, and Wolfgang Nagel. Deadlock-Free Oblivious Routing for Arbitrary Topologies. In *Proceedings of the 25th IEEE International Parallel and Distributed Processing Symposium*, pages 613–624. IEEE Computer Society, May 2011.

[88] Bill Fenner, Mark Handley, Hugh Holbrook, and Isidor Kouvelas. Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification

(Revised). RFC 4601 (Proposed Standard), August 2006. Updated by RFCs 5059, 5796, 6226.

[89] Ronald Greenberg and Charles Leiserson. Randomized Routing on Fat-Trees. In *Advances in Computing Research*, pages 345–374. JAI Press, 1996.

[90] Leslie Valiant and Gordon Brebner. Universal Schemes for Parallel Communication. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing*, STOC '81, pages 263–277. ACM, 1981.

[91] Torsten Hoefler, Timo Schneider, and Andrew Lumsdaine. Multistage Switches are not Crossbars: Effects of Static Routing in High-Performance Networks. In *Proceedings of the IEEE International Conference on Cluster Computing*, pages 116–125, September 2008.

[92] Germán Rodríguez, Cyriel Minkenberg, Ramón Beivide, Ronald Luijten, Jesús Labarta, and Mateo Valero. Oblivious Routing Schemes in Extended Generalized Fat Tree Networks. In *Proceedings of the IEEE International Conference on Cluster Computing*, pages 1–8, 2009.

[93] Xin Yuan, Wickus Nienaber, Zhenhai Duan, and Rami Melhem. Oblivious Routing for Fat-tree Based System Area Networks with Uncertain Traffic Demands. In *Proceedings of the 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS, pages 337–348. ACM, 2007.

[94] Frank Olaf Sem-Jacobsen. *Towards a Unified Interconnect Architecture: Combining Dynamic Fault Tolerance with Quality of Service, Community Separation, and Power Saving.* PhD thesis, University of Oslo, August 2008.

[95] Mukesh Singhal. Deadlock Detection in Distributed Systems. *Computer*, 22(11):37–48, 1989.

[96] William Dally and Charles Seitz. The Torus Routing Chip. *Journal of Distributed Computing*, 1(3):187–196, 1986.

[97] Mellanox Technologies. InfiniBand White Papers. `http://www.mellanox.com/page/white_papers`. Retrieved November 21, 2013.

[98] Taneja Group. InfiniBand's Data Centers March. `https://cw.infinibandta.org/document/dl/7269`, July 2012.

[99] Dennis Abts, Michael Marty, Philip Wells, Peter Klausler, and Hong Liu. Energy Proportional Datacenter Networks. In *Proceedings of the 37th Annual International Symposium on Computer Architecture*, pages 338–347. ACM, 2010.

[100] Andrew Odlyzko. Data Networks Are Mostly Empty and for Good Reason. *IT Professional*, 1(2):67–69, March 1999.

[101] IEEE P802.3az. `http://grouper.ieee.org/groups/802/3/az`, 2010.

[102] Wei Lin Guay, Sven-Arne Reinemo, Olav Lysne, Tor Skeie, Bjørn Dag Johnsen, and Line Holen. Host Side Dynamic Reconfiguration with InfiniBand. In *IEEE International Conference on Cluster Computing*, pages 126–135. IEEE Computer Society, September 2010.

[103] Michael Schroeder, Andrew Birrell, Michael Burrows, Hal Murray, Roger Needham, Thomas Rodeheffer, Edwin Satterthwaite, and Charles Thacker. Autonet: A High-Speed, Self-Configuring Local Area Network Using Point-to-point Links. *IEEE Journal on Selected Areas in Communications*, 9(8):1318–1335, 1991.

[104] Marcus Epperson, John Naegle, Jim Schutt, Matthew Bohnsack, Steve Monk, Mahesh Rajan, and Doug Doerfler. HPC Top 10 InfiniBand Machine... A 3D Torus InfiniBand Interconnect on Red Sky. Contributed talk to OpenFabrics International Workshop, March 2010.

[105] Åshild Grønstad Solheim, Olav Lysne, Tor Skeie, Thomas Sødring, and Ingebjørg Theiss. Routing for the ASI Fabric Manager. *IEEE Communications Magazine*, 44(7):39–44, 2006.

[106] Olav Lysne, Tor Skeie, Sven-Arne Reinemo, and Ingebjørg Theiss. Layered Routing in Irregular Networks. *IEEE Transactions on Parallel and Distributed Systems*, 17(1):51–65, 2006.

[107] HPC Challenge Benchmark. `http://icl.cs.utk.edu/hpcc/`, 2013. Retrieved October 25, 2013.

[108] Ernst Gunnar Gran, Magne Eimot, Sven-Arne Reinemo, Tor Skeie, Olav Lysne, Lars Paul Huse, and Gilad Shainer. First Experiences with Congestion Control in InfiniBand Hardware. In *Proceedings of the 24th International Parallel and Distributed Processing Symposium*, pages 1–12, 2010.

[109] Ernst Gunnar Gran, Sven-Arne Reinemo, Olav Lysne, Tor Skeie, Eitan Zahavi, and Gilad Shainer. Exploring the Scope of the InfiniBand Congestion Control Mechanism. In *Proceedings of the 26th IEEE International Parallel and Distributed Processing Symposium*, pages 1131–1143. IEEE Computer Society, 2012.

Published Works

Paper I

Achieving Predictable High Performance in Imbalanced Fat Trees

Bartosz Bogdański, Frank Olaf Sem-Jacobsen, Sven-Arne Reinemo, Tor Skeie, Line Holen, Lars Paul Huse

2010 16th International Conference on Parallel and Distributed Systems

Achieving Predictable High Performance in Imbalanced Fat Trees

Bartosz Bogdanski*, Frank Olaf Sem-Jacobsen*, Sven-Arne Reinemo*, Tor Skeie*, Line Holen†, Lars Paul Huse†
*Simula Research Laboratory
P.O. Box 134,
NO-1325, Lysaker, Norway
E-mail:{bartoszb, frankose, svenar, tskeie}@simula.no
†Oracle Corporation, Oslo, Norway
E-mail: {Line.Holen, Lars.Paul.Huse}@oracle.com

***Abstract*—The fat-tree topology has become a popular choice for InfiniBand fabrics due to its inherent deadlock freedom, fault-tolerance and full bisection bandwidth. InfiniBand is used by more than 40% of the systems on the latest Top 500 list, and many of these systems are based on a fat-tree topology. However, the current InfiniBand fat-tree routing algorithm suffers from flaws that reduce its scalability and flexibility. Counter-intuitively, the achievable throughput per node deteriorates both when the number of nodes in a tree decreases or when the node distribution among leaves is nonuniform.**

In this paper, we identify the weaknesses of the current enhanced fat-tree routing algorithm in OpenFabrics Enterprise Distribution and we propose extensions to it that alleviate all performance problems related to node distribution. The new algorithm is implemented in OpenSM for real world evaluation and for future contribution to the OpenFabrics community. We demonstrate that our solution allows to achieve a predictable high throughput regardless of the number of nodes and their distribution. Furthermore, the simulations show that our extensions improve throughput up to 30% depending on topology size and node distribution.

***Keywords*-routing; fat-trees; interconnection networks; InfiniBand;**

I. Introduction

The fat-tree topology is one of the most common topologies for high performance computing clusters today, and for clusters based on InfiniBand (IB) technology the fat-tree is the dominating topology. This includes large installations such as LANL Roadrunner, TACC Ranger, and FZJ-JSC JuRoPA [1]. There are three properties that make fat-trees the topology of choice for high performance interconnects: deadlock-free routing, the use of a tree structure makes it possible to route fat-trees without special considerations for deadlock avoidance; inherent fault-tolerance, the existence of multiple paths between individual source-destination pairs makes it easier to handle network faults; full bisection bandwidth, a balanced fat-tree can sustain full speed communication between the two halves of the network.

The InfiniBand Architecture was first standardized in 2000 [2], as a combination of two older technologies called Future I/O and Next Generation I/O. As with most other recent interconnection networks, IB is a serial point-to-point full-duplex technology. Due to efficient utilization of host side processing resources it is scalable beyond ten-thousand nodes each with multiple CPU cores. The current trend is that IB is replacing proprietary or lower performance solutions in the high performance computing domain [1], where high bandwidth and low latency are the key requirements. For fat-trees, as with most other topologies, the routing algorithm is crucial for efficient use of the underlying topology. The popularity of fat-trees in the last decade led to many efforts trying to improve their routing performance. This includes the current approach that the OpenFabrics Enterprise Distribution (OFED) [3], the de facto standard for IB system software, is based on [4], [5].

Nevertheless, the proposals to improve the routing performance have several limitations when it comes to flexibility and scalability. One problem is the static routing used by IB technology that limits the exploitation of the path diversity in fat-trees as pointed out by Hoefler et al. in [6]. Another problem with the current routing are its shortcomings when routing oversubscribed fat-trees as addressed by Rodriguez et al. in [7]. A third problem, and the one that we are addressing in this paper, is that performance is reduced when the number of compute nodes connected to the tree is reduced. For large fabrics where the leaf switches are not fully-populated with end nodes the throughput per node is lower than for the case when the leaf switches are fully-populated. This means that keeping the switching capacity constant while reducing the number of nodes in the network leads to reduced performance (less throughput per node). This counter-intuitive behavior is a problem in situations where the fat-tree is not fully populated. Such a situation often occurs during cluster construction, for underpopulated clusters ready for future expansion (a common scenario), and for power saving clusters where end nodes are powered down when not in use. The severity of the problem depends on the distribution of the end nodes and the size of the tree.

In this paper, we analyze the performance of the fat-tree routing algorithm for IB with various partially populated trees and with various distributions of the end nodes. We show through simulations how the performance of the current algorithm is reduced when the number of end nodes

1521-9097/10 $26.00 © 2010 IEEE
DOI 10.1109/ICPADS.2010.94

is reduced, and how various distributions of the end nodes further affect network throughput. We then extend the algorithm by adding a balancing step that makes the performance independent of the number and distribution of the end nodes.

The rest of this paper is organized as follows: we introduce the fat-tree routing and our optimizations in Section II. We describe the experimental setup in Section III followed by the experimental analysis in Section IV. Finally, we conclude in Section V.

II. Fat-Tree Routing

The fat-tree is an efficient interconnect topology that is well-suited for general purpose high-performance computing. It was first introduced by C. Leiserson in 1985 [8], and gained a large installation base among the current Top 500 supercomputers [1]. A balanced fat-tree is a tree topology where the link capacity at every tier is the same. This is commonly implemented by building a tree with multiple roots, often following the m-port n-tree definition [9]. In general, fat-tree routing consists of an upward phase from the source of a packet, and a downward phase towards the destination. The transition from the upward phase to the downward phase occurs in the least common ancestor, a switch which reaches both the source and the destination through different downward links. This algorithm ensures deadlock-free connectivity between all source-destination pairs, but it does not guarantee any sort of balancing of network traffic. Exactly how the different paths are distributed in the network is subject to specific implementations of the algorithm. In this section we discuss the fat-tree routing algorithm implementation found in OFED. One of the packages distributed with OFED is OpenSM, the subnet manager (SM) for IB subnets. An IB subnet requires one or more subnet managers, which are responsible for initializing and bringing up the network. A major part of the SMs responsibility are routing table calculations. Routing of the network aims at obtaining full connectivity, deadlock freedom, and proper load balancing between all source and destination pairs. Routing tables must be calculated at network initialization time and this process must be repeated whenever the topology changes in order to update the routing tables and ensure optimal performance.

OFED enhances the fat-tree routing by ensuring that every path to a given destination converges towards the same root node in the fat-tree, causing all packets with the same destination to follow a single path in the downward phase. The inspiration for this path distribution was [9], where the authors present the m-port n-tree definition and an associated multipath routing algorithm which performs a good distribution of the initial paths. Another implementation of a similar routing/balancing algorithm is given in [4]. In a fat-tree that is not oversubscribed, this allows for a dedicated downward path towards every destination that does not share traffic with any other destination. Gomez et al. [4] show that this path

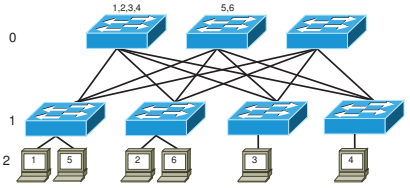


Figure 1: With the current algorithm the selection of the root switch depends on the node distribution which might result in poorly balanced paths.

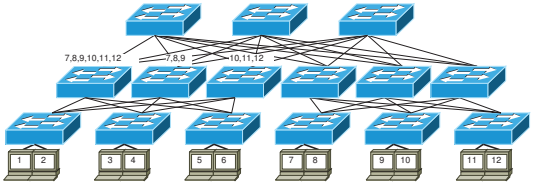


Figure 2: Several rack switches show preference for the same upward leaf switch, resulting in a poorly balanced routing algorithm.

distribution gives very good performance for uniform traffic patterns. By having dedicated downward paths for every destination, contention in the downward phase is effectively removed (moved to the upward phase), so that packets for different destinations have to contend for output ports in only half of the switches on their paths.

Specifically, such balancing is implemented in OpenSM 3.3.5, the SM distributed with OFED 1.5.1, by setting up the path in the reverse direction, from the destination towards the source. In the reverse downward direction from the destination d (at tier n, see Fig. 1) connected to switch s at tier $n-1$ towards the root (at tier 0), the algorithm will always choose the switch s at tier $n-1$ which is connected to d at tier n and has the lowest number of destinations with the connecting link as output. Then, the upward port from s (connected to switches at tier $n-2$) which has the lowest load of downward paths is chosen, etc.

None of the routing algorithms we described consider the effect of the number and the distribution of the end nodes, and they implicitly assume that the network is fully-populated by end nodes in terms of balancing. It became evident from a number of real systems that prompted this study that the number and the distribution of the end nodes have a significant impact on the performance of the system, and, moreover, this results from the manner in which the routing algorithm was implemented. In the next section, we explain why having a dedicated downward path is not sufficient to achieve good performance, and describe how this causes performance degradation.

A. Balancing Issues

There are two fat-tree configurations that we find in current systems. The first is a regular fat-tree where the end nodes are connected directly to the leaf switches of the tree. The second consists of one or more fat-trees with

an extra tier of rack switches at the bottom, connected to all of the fat-trees. Both these configurations experience issues with poor balancing of the network traffic in the case where the leaves/rack switches are not fully-populated with end nodes, both with an uniform and nonuniform node distribution. In the following paragraphs we explain why this imbalance occurs, before we outline in the next sections how we modified the algorithm to tolerate arbitrary end node connection patterns without a significant performance degradation.

The main concept behind the routing algorithms we described is that every destination has a dedicated downward path from a specific root in the fat-tree. For fully-populated topologies this is sufficient. However, as the connectivity pattern of the end nodes becomes more irregular, a specific artifact of the algorithm becomes evident. In addition to minimizing contention, it is also important to spread the traffic evenly throughout the network. This aspect was not sufficiently considered in the work presented in the previous section. For the interested reader, note also that algorithm 4, and the solution for the missing compute nodes in section 3.2 in [5] are insufficient for completely balancing the fat-tree. Namely, the problem are the dummy end nodes that are introduced to ensure a correct distribution of the downward paths in the network with respect to an all-to-all shift communication pattern. While this leads to an apparently balanced network, it may lead to skew in the balance of the upward paths.

We first consider the case for a regular fat-tree network where the end nodes are connected directly to the leaves of the fat-tree. The original condition that every destination should have a unique path from a root in the topology to the destination, which is not shared by any path to any other destination, is valid regardless of the manner in which end nodes are connected to the network. Any imbalance must therefore lie in the manner in which the root nodes for these paths are selected.

Consider two end nodes that are connected to different leaf switches such that their least common ancestors are at the root tier of the topology. Since the algorithm only checks the load of the path coming from the switch above when constructing the downward path, it is entirely possible that the downward path to these two destinations will originate from the same root switch. Consequently, given 24 end nodes in a fat-tree consisting of 24-port switches, connected such that the least common ancestors of every possible pair is at the root tier of the fat-tree (this is possible with up to and including 24 nodes), a legal outcome of the algorithm is that one single root tier switch is the origin of the downward path to all 24 destinations (the actual outcome will depend on the sequence in which the possible output ports are evaluated). As even more end nodes are added to the topology, traffic towards these 24 destinations will be heavily aggregated towards the single root switch,

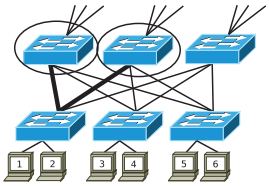


Figure 3: Our proposed approach of balancing without rack switches by considering the number of downward paths routed through the switches at the stage above (circled), in addition to the number of paths on each of the downward links (bold).

thus yielding much lower performance than if each of the 24 destinations was routed through different root switches. It is thus not sufficient to only consider the load of the downward links from the stage above when choosing the downward path in the routing algorithm. An example of this is presented in Fig. 1.

The second case which may cause an imbalance in the amount of traffic that is transmitted through the different root switches (or network switches in general) is when there is an additional layer of switches (called in general "rack switches") between the end nodes and the fat-tree topology. For the topologies considered in this paper, the rack switches have multiple connections to the fat-tree topology (topologies), allowing traffic from a given source to enter the fat-tree topology through one of several possible leaf switches (see Fig. 2). Which of the possible paths are actually chosen is somewhat arbitrary in the current algorithm, it is the path that first reaches the rack switches as the routing algorithm traverses all possible paths. This will, in most cases, cause traffic from the rack switches to favor certain leaf switches in the fat-tree, causing the paths from the switches to be overloaded, while underutilizing other alternative paths. Similar to the first case, it is clear that simply considering the load of a single link is insufficient to achieve a well-balanced fat-tree routing function.

B. Balancing Solutions

Both problems outlined in the previous section can be solved to a large extent by the same mechanism, namely considering the load of downward channels and switches at higher tiers than the one directly above. Ideally, the complete load of all possible downward paths from the root tier to the current switch should be considered, which would guarantee the ideal balance with regards to the number of communicating pairs that share each link (see Fig. 3). However, this would give the routing algorithm an exponential complexity, and the evaluations we present below indicate that considering only the downward paths in the directly connected links and the total number of downward paths in the switches to which they are connected at the tier above (as opposed to only the directly connected links in the original algorithm) is sufficient to achieve very good balance.

For the first imbalance case this is sufficient to greatly improve the performance of the routing algorithm for topologies that do not include rack switches. However, for the second case we must add an additional mechanism to ensure good balancing. For a regular fat-tree, every leaf switch will only have a single path to the chosen root switch for a given destination. The rack switches, however, will have multiple. Therefore, a complete solution to the second case involves correctly choosing which fat-tree leaf switch to forward traffic to for a given destination. This requires a two-step algorithm. First, we must go through all the leaf switches that are part of the possible path to a given destination from a given rack switch and determine whether one of these already carries traffic from another rack switch towards the given destination. If such a leaf switch is found, choose it as the next hop from the rack switch to the destination. If such a switch is not found, choose the leaf switch that already forwards traffic from rack switches (or directly connected end nodes) to the lowest number of destinations. Aggregating all traffic for a single destination as early as possible as done with this algorithm will decrease network contention, and thus increase performance in addition to improving the balance of source-destination pairs for the individual links.

III. Experiment Setup

To validate our claims in this paper and perform large-scale evaluations, we used an IB model for the OMNeT++ simulator. The model contains an implementation of HCAs and switches with routing tables. The topology and routing tables are generated using OpenSM and converted into OMNeT++ readable format in order to simulate real-world systems. For our simulations we used the following two traffic patterns for performance analysis:

1) uniform traffic pattern: for all topologies we are sending series of 2 kB IB packets and the network load varies from 0% to 100%. The link speed was set to SDR (1GB/s) for all the simulations. For each message each source is choosing the destination randomly from a list of all possible destinations.

2) ping pong traffic pattern: which was used to run the HPC Challenge Benchmark tests in the simulator. For the bandwidth tests we used a message size of 1954 kB and the network load was set constant at 100%. The bandwidth tests were performed on 31 ring patterns: one natural-ordered ring and 30 random-ordered rings from which the minimum, maximum and average results were taken. For this measurement, each node sends a message to its left neighbor in the ring and receives a message from its right neighbor. Next, it sends a message back to its right neighbor and receives a return message from its left neighbor.

A. Topologies

The simulations were run on five different fat-tree topologies modeled after real systems. In particular, we have used commercially available switch designs with 648 ports with and without additional rack switches and a switch design with 3456 ports. A 648-port switch is the largest possible 2-stage fat-tree switch that can be constructed using 36-port switch elements and a 3456-port switch is the largest possible 3-stage fat-tree switch that can be constructed using 24-port switch elements. To construct the former, 54 36-port switch elements are needed (18 roots and 36 leaves) and to construct the latter, 720 24-port switch elements are required (144 roots, 288 intermediate switches and 288 leaves). By "rack switches" we mean external switches that are connected to the leaves in the chassis and they usually have multiple upward connections to different leaves. Computing nodes are connected to these switches, where applicable.

Moreover, we have connected multiple 648-port switches and 3456-port switches to simulate supercomputer systems similar to JuRoPA [10] and Ranger [11]. These supercomputers are constructed using multiple fat-trees that are interlinked through rack switches. Consequently, the JuRoPA topology consists of eight 648-port switches with additional cross-linked rack switches (explained in the next section) and the Ranger topology is built from two 3456-port switches with an additional switching stage consisting of rack switches at the bottom tier. The 648-port systems are currently a popular product and many vendors offer their own systems [12]–[14] while a 3456-port switch is only available from Oracle/Sun [15].

The exact node distribution and the total number of nodes for each topology is presented in Table I. By a node distribution of X:Y we mean that X nodes are connected to the even leaf switches and Y nodes are connected to the odd ones. For topologies with cross-linked rack switches, such a notation implies that X nodes in total are connected to the first two cross-linked rack switches, and Y nodes in total are connected to the next two.

IV. Performance Evaluation

Network performance depends on the physical topology, the chosen routing algorithm and the applied traffic pattern. To properly understand the influence of these factors, we have performed numerous simulations on different network topologies with various node distributions using the most recent version of the routing algorithm available for IB, the *enhanced fat-tree* (EFT) algorithm, and the *balanced enhanced fat-tree* (BEFT) algorithm proposed in this paper. We use the achieved average *throughput per end node* relative to the link capacity as the metric for measuring performance and comparing the two algorithms.

A. Uniform Traffic

This traffic pattern is likely to cause light congestion in the fabric because the destinations are chosen randomly by each source (i.e. two or more sources may send traffic to a

Table I: Node distribution for the simulated topologies.

Even leaf switches	Odd leaf switches	Total no. of nodes
9	9	324
11	7	324
13	5	324
17	1	324
18	18	648

(a) 648-port switch

Even leaf switches	Odd leaf switches	Total no. of nodes
6	6	1728
7	5	1728
9	3	1728
11	1	1728
12	12	3456

(b) 3456-port switch

Even rack switches	Odd rack switches	Total no. of nodes
12	12	324
22	1	321
24	24	648

(c) 648-port switch with cross-linked rack switches

Even rack switches	Odd rack switches	Total no. of nodes
12	12	2592
23	1	2592
24	24	5184

(d) JuRoPA-like supercomputer

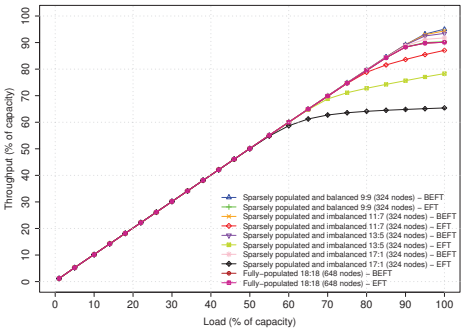


Figure 4: Simulation results for 648-port topologies.

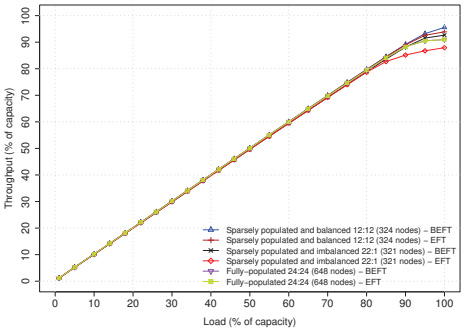


Figure 5: Simulation results for 648-port topologies with rack switches.

single destination). In the following section we discuss the performance results for each of the five topologies we have studied.

1) 648-port fat-tree: We have routed and performed simulations for the configurations listed in Table Ia. Fig. 4 shows the results for all configurations. The main results are that for all configurations the BEFT algorithm outperforms the EFT algorithm and the best case for the EFT algorithm equals to the worst case for the BEFT algorithm. This happens for the fully-populated configuration where balancing is not an issue (Table Ia row five). Going into more detail, we observe that, in the case of a skewed node distribution, the EFT algorithm fails to choose the root switches in a balanced manner, and consequently there is a drop in throughput. This is especially visible in case of a node distribution of 17:1, where the average throughput per node for the BEFT algorithm is over 90% compared to approximately 65% for the EFT algorithm. Furthermore, the experiment shows that with uniformly distributed end nodes (9:9), the difference between the BEFT and EFT algorithm is approximately 5% in favor of the BEFT algorithm.

2) 648-port fat-tree with cross-linked rack switches: In this configuration we treat two cross-linked rack switches as one large rack switch when connecting the nodes (e.g. a 22:1 node distribution means that 22 nodes are connected to first two cross-linked rack switches and only 1 node is connected to the other two cross-linked rack switches). Moreover, we used a design available from Oracle/Sun [16] where in each cross-linked rack switch, two switch elements share four upward leaf switches and each of the rack switches uses a three-port port group to connect to a single upward leaf switch.

With this configuration the observed differences between the two algorithms are at most 6%, always in favor of our BEFT algorithm (see Fig. 5). The explanation is that multiple connections to the upward switches and the use of cross-links reduce the contention on the upward ports of the rack switches. Path distribution across the root switches is more balanced, therefore there is only a small performance drop with the EFT algorithm.

3) 3456-port fat-tree: The simulation results, shown on Fig. 6, confirm the existence of the same balancing issues as in the case of 648-port switch. Even if the nodes are uniformly distributed (6:6), the difference in the average throughput per node is visible with 95.42% achieved by BEFT and 89.85% obtained by EFT algorithm. By further increasing the node skew, we observe that the throughput for fabrics routed using EFT decreases dramatically. This is not the case for BEFT which maintains a generally constant throughput regardless of the node distribution.

4) JuRoPA-like topology: JuRoPA supercomputer is a fat-tree built from eight 648-port switches. These eight separate switches are interlinked at the bottom tier using cross-linked rack switches, so all-to-all connectivity is possible. The

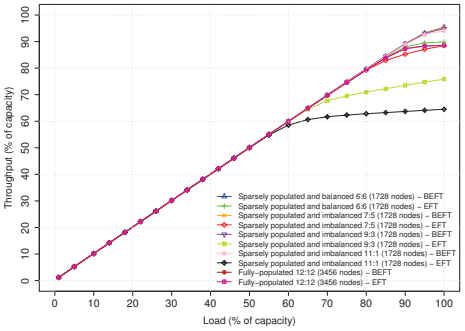


Figure 6: Simulation results for 3456-port topologies.

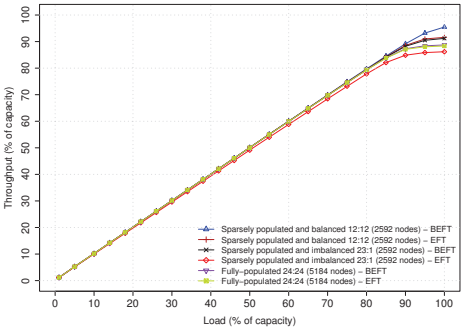


Figure 7: Simulation results for the JuRoPA-like topology.

results presented on Fig. 7 confirm that the introduction of cross-linked rack switches reduces the balancing issues for EFT. Nevertheless, by introducing a large enough skew (23:1), the balancing issues become evident and the difference in achieved throughput is 5%. Moreover, for every node distribution apart from the fully-populated state (Table Id row three), the throughput in a fabric routed by BEFT was at least a few percent larger than in the case of EFT. While the above numbers show that BEFT provides better path balancing than EFT, the important practical observation from the point of view of such a supercomputer as JuRoPA is that the proper choice of a routing algorithm may boost applications' performance.

5) Ranger-like fat-tree: Ranger, a dual-tree topology, is built from two 3456-port switches interlinked with rack switches at the bottom tier. We performed the simulations for the same node distributions as listed in Table Ib, however, we are not presenting the graph due to space limitations. We observed that, in this case, the differences between both algorithms are minimal (in a range of 1-2%), but still in favor of BEFT for every performed experiment apart from the fully-populated scenario (Table Ib, row five), in which the results were identical. This shows that BEFT balances the paths through the roots better than EFT.

B. HPC Challenge Benchmark

The implementation of this traffic pattern follows the freely available algorithm for performing the HPCC tests on real clusters [17]. Table II lists the bandwidth tests results for a 648-port switch and a 648-port switch with cross-linked rack switches. For fat-trees, all the natural-ordered ring bandwidth results will be the same (no contention). However, when we compare the average or minimum bandwidths for random-ordered rings for any topology (apart from the fully-populated one for 648-port switch), we observe that by using BEFT, we gain a relative improvement in bandwidth ranging from 3.63% to 22.31% for 648-port switch and from 3.1% to 57% for 648-port switch with cross-linked rack switches. An interesting observation is the fact that even for a fully-populated 648-port switch with cross-linked rack switches there is a noticeable performance gain when using BEFT, which is explained by the fact that the LID-to-output port mapping for EFT is different than for BEFT, and that the particular traffic patterns caused more congestion.

Table II: The HPC Challenge Benchmark results.

Measurement	BEFT (MB/s)	EFT (MB/s)	Improvement
648-port fat-tree with 9:9 node distribution			
NOR BW[1]	1523.43	1523.43	-
ROR BW MIN[2]	284.06	251.29	13%
ROR BW MAX[2]	1523.73	1523.73	-
ROR BW AVG[2]	962.87	787.24	22.3%
648-port fat-tree with 17:1 node distribution			
NOR BW AVG	1523.55	1523.55	-
ROR BW MIN	255.96	225.74	13.4%
ROR BW MAX	1523.73	1523.73	-
ROR BW AVG	829.28	800.20	3.6%
648-port fat-tree with 18:18 node distribution			
NOR BW	1523.56	1523.56	-
ROR BW MIN	176.10	176.10	-
ROR BW MAX	1523.75	1523.75	-
ROR BW AVG	769.90	769.89	-

(a) 648-port fat-tree switch

Measurement	BEFT (MB/s)	EFT (MB/s)	Improvement
648-port fat-tree (racks cross-linked) with 12:12 node distribution			
NOR BW	1523.36	1523.36	-
ROR BW MIN	390.14	267.34	45.9%
ROR BW MAX	1522.92	1399.33	8.8%
ROR BW AVG	855.61	784.89	9%
648-port fat-tree (racks cross-linked) with 22:1 node distribution			
NOR BW AVG	1523.43	1523.43	-
ROR BW MIN	363.92	231.79	57%
ROR BW MAX	1523.73	1523.73	-
ROR BW AVG	798.39	724.98	10.1%
648-port fat-tree (racks cross-linked) 24:24 node distribution			
NOR BW	1523.50	1523.50	-
ROR BW MIN	267.81	170.68	56.9%
ROR BW MAX	1523.73	1523.73	-
ROR BW AVG	676.56	656.22	3.1%

(b) 648-port fat-tree switch with additional rack switches

[1] Natural-ordered ring bandwidth;
[2] Random-ordered ring bandwidth (minimum, maximum and average);

Table III: Routing execution time for different topologies in seconds.

Topology	BEFT	EFT
648-port fat-tree switch (s)	0.03	0.02
648-port fat-tree with rack switches (s)	0.54	0.07
3456-port fat-tree switch (s)	1.89	4.71
JuRoPA-like supercomputer (s)	46.02	9.52
Ranger-like supercomputer (s)	145.3	36.72

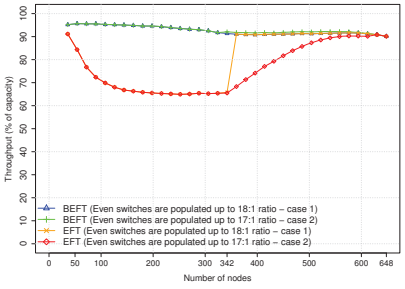


Figure 8: Relationship between the node distribution (increasing imbalance) and the achieved throughput for an 648-port network topology.

C. Node number vs. throughput

An important question is how the achieved throughput is related to the number of nodes in a particular topology. Specifically, the purpose of the experiment presented in this section was not only to gain an overall comparison of both algorithms but also to determine what is the predictability of the achieved throughput for each of the algorithms at full (100%) load under uniform traffic when the number of nodes varies. The simulations were carried out for two fat-tree topologies: a 648-port fat-tree switch and a 3456-port switch.

Furthermore, to properly understand the influence of the dummy nodes (described in Section II-A and Section 3.2 of [5]), we chose two *border case* node distributions for each fat-tree topology. For the 648-port fat-tree, the number of nodes was increased from 36 (1 node per each leaf switch) up to 324 (17 nodes per even leaf switches and 1 per odd ones). Next, we introduced the two border cases: *first*, we added one more node to all even leaf switches (18:1) to fully

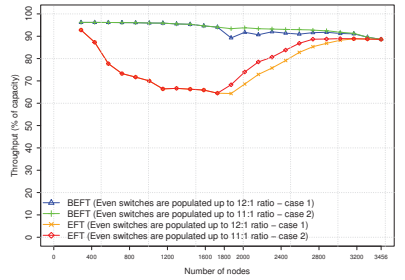


Figure 9: Relationship between the node distribution (increasing imbalance) and the achieved throughput for an 3456-port network topology.

populate them and then we started increasing the number of nodes on odd leaf switches until we reached a fully-populated network (18:2, 18:3,...,18:18); *second*, at 17:1 node distribution, we started adding nodes on the odd leaf switches (17:2, 17:3,..., 17:18), and, ultimately, we added the final missing nodes on even leaf switches, again reaching the fully-populated state (18:18). The results are presented on Fig. 8 and a more detailed analysis shows that even with a ratio of 5:1, the achieved throughput in a fabric routed by EFT declines to 70% of capacity and reaches a minimum of 64.9% for 13:1 ratio. Moreover, as visible in the second border case, when no leaf switches are fully-populated with nodes (i.e. no leaf has 18 nodes attached), and the node distribution becomes less skewed, the throughput increases gradually, but it is still much lower than for BEFT. The dummy nodes activate properly only when the fabric reaches 18:1 ratio, meaning that at least some of the switches have to be fully-populated. This behavior is marked by the large vertical jump from 65% to over 90% throughput on Fig. 8. However, the dummy nodes do not function correctly unless at least one of the switches reaches a fully-populated state (shown by the second border case), or, as shown on Fig. 9, not function at all in case of a more complex topology. The important observation is that, regardless of the number of nodes and their distribution, BEFT achieves a *predictable* high throughput at a level of 90% of capacity. Additionally, in case we do not have fully-populated switches in the network (second border case), the throughput is slightly higher because the paths are balanced between all of the root switches and a smaller number of nodes makes the contention at the root level smaller.

Fig. 9 shows the results of a similar experiment performed on a 3456-port fat-tree topology. Again, we introduce two border cases. Having started adding the nodes at a ratio of 1:1 (288 nodes), we introduced the first case at a ratio 11:1 (1728 nodes) when we fully populated the even leaf switches with 12 nodes (12:1) and started populating the odd leaf switches (12:2, 12:3,...12:12) until reaching a distribution of 12:12. Next, we introduced the second case where we did not populate any of the switches fully until the very last steps (11:2, 11:3..., 11:11, 11:12, 12:12). Note that even at a ratio of 1:1, EFT does not choose the root switches in a balanced manner as the achieved throughput is lower than for BEFT. Moreover, at a ratio of 7:1 for EFT, the achieved throughput is lower than 70% and the minimum is achieved at 12:1 ratio (64.3%). However, the most striking feature is the fact that dummy nodes do not activate when 12:1 distribution is reached. This is explained by the greater complexity of the 3456-port fabric and the lack of scalability of the dummy node implementation. A noticeable aspect of BEFT is that the throughput suddenly drops by 4.8% when 12:1 ratio is reached. This is a general artifact in the fat-tree routing algorithm, which assigns the upward output ports on a switch in a sequential manner, and, for some node distributions, it is

impossible to have a perfectly equal balance on every single port. Even though this artifact influences BEFT's throughput, we observe that it is much more stable and predictable than EFT irrespective of the node distribution and the achieved throughput for BEFT is always in the range between 88.5 and 96.2%.

D. Execution time

The execution time of the fat-tree routing algorithm depends on the overall size of the network and is mainly influenced by the number of computing nodes and switches in the network. Because BEFT chooses the next hop switch in a more sophisticated manner, its execution time is generally greater than that of EFT. However, this is only true for networks for which the special routing in leaves is executed (described in Section II-B), which is not necessary for simpler networks like 648-port or 3456-port fat-tree switches. To decrease the execution time on these topologies, we introduced a command line parameter $-Z$ (off by default), which, when supplied to the routing engine, executes the time-consuming logic allowing the algorithm to properly balance the paths for complex topologies. Table III shows the execution time of both algorithms for a set of the fully populated topologies considered in this paper. For less complex topologies, like 648-port or 3456-port fat-trees, special leaf routing is not necessary, so BEFT execution time is comparable to or faster than for EFT because the leaf balancing does not need to be invoked. For more complex topologies (e.g. 648-port switch with cross-linked racks, JuRoPA-like or Ranger-like) there is a trade-off between a longer routing time and better balancing (thus, better performance).

V. Conclusions and Future Work

In this paper, we identified a flaw in the existing fat-tree routing algorithm for IB networks, where the achievable throughput per node deteriorates both when the number of nodes in a tree decreases and when the node distribution among the leaf switches is nonuniform. With this insight, we proposed an extension to the existing algorithm that alleviates all performance problems related to node distribution. To evaluate the algorithm, we simulated several fat-tree topologies based on common switching products and real installations. Simulation results show that our extensions improve performance by 30% depending on topology size and node distribution. Furthermore, the simulations demonstrate that BEFT allows to achieve a high predictable throughput irrespective of the node distribution and the node number.

In the future, we plan to expand this work to also cover fat-trees with nonuniform distribution of switches, and to contribute our changes to OpenFabrics community. Looking further ahead, we will also look into extensions that will reduce the amount of network congestion by using nonproprietary and widely available techniques.

Acknowledgments

This work is in part financed by Oracle Corporation.

References

[1] "Top 500 supercomputer sites," http://top500.org/, June 2010.

[2] *Infiniband architecture specification*, 1st ed., InfiniBand Trade Association, November 2007.

[3] "The OpenFabrics Alliance," http://openfabrics.org/.

[4] C. Gomez, F. Gilabert, M. E. Gomez, P. Lopez, and J. Duato, "Deterministic versus Adaptive Routing in Fat-Trees," in *Workshop on Communication Architecture on Clusters, IPDPS*, 2007.

[5] E. Zahavi, G. Johnson, D. J. Kerbyson, and M. Lang, "Optimized Infiniband fat-tree routing for shift all-to-all communication patterns," in *Concurrency and Computation: Practice and Experience*, 2009.

[6] T. Hoefler, T. Schneider, and A. Lumsdaine, "Multistage switches are not crossbars: Effects of static routing in high-performance networks," in *IEEE International Conference on Cluster Computing*, 2008.

[7] G. Rodriguez, C. Minkenberg, R. Beivide, and R. P. Luijten, "Oblivious Routing Schemes in Extended Generalized Fat Tree Networks," *IEEE International Conference on Cluster Computing and Workshops*, 2009.

[8] C. E. Leiserson, "Fat-Trees: Universal Networks for Hardware-Efficient Supercomputing," *IEEE Transactions on Computers*, 1985.

[9] X.-Y. Lin, Y.-C. Chung, and T.-Y. Huang, "A Multiple LID Routing Scheme for Fat-Tree-Based Infiniband Networks," *Proceedings of IEEE International Parallel and Distributed Processing Symposiums*, 2004.

[10] "Julich Supercomputing Centre," http://www.fz-juelich.de/jsc/juropa/configuration/.

[11] "Texas Advanced Computing Center," http://www.tacc.utexas.edu/.

[12] "IS5600 - 648-port InfiniBand Chassis Switch," Mellanox Technologies, http://www.mellanox.com/related-docs/prod_ib_switch_systems/IS5600.pdf.

[13] "Sun Datacenter InfiniBand Switch 648," http://www.oracle.com/us/products/servers-storage/networking/infiniband/034537.htm.

[14] "Voltaire QDR InfiniBand Grid Director 4700," http://www.voltaire.com/Products/InfiniBand/Grid_Director_Switches/Voltaire_Grid_Director_4700.

[15] "Sun Datacenter Switch 3456," http://www.oracle.com/us/products/servers-storage/networking/infiniband/031556.htm.

[16] "Sun Blade 6048 InfiniBand QDR Switched NEM," http://www.oracle.com/us/products/servers-storage/servers/blades/031095.htm.

[17] "HPC Challenge Benchmark," http://icl.cs.utk.edu/hpcc/.

Paper II

vFtree - A Fat-tree Routing Algorithm using Virtual Lanes to Alleviate Congestion

Wei Lin Guay, Bartosz Bogdański, Sven-Arne Reinemo, Olav Lysne, Tor Skeie

2011 IEEE International Parallel & Distributed Processing Symposium

vFtree - A Fat-tree Routing Algorithm using Virtual Lanes to Alleviate Congestion

Wei Lin Guay, Bartosz Bogdanski, Sven-Arne Reinemo, Olav Lysne, Tor Skeie
Simula Research Laboratory
P.O. Box 134,
NO-1325, Lysaker, Norway
E-mail:{weilin, bartoszb, svenar, olavly, tskeie}@simula.no

***Abstract*—It is a well known fact that multiple virtual lanes can improve performance in interconnection networks, but this knowledge has had little impact on real clusters. Currently, a large number of clusters using InfiniBand is based on fat-tree topologies that can be routed deadlock-free using only one virtual lane. Consequently, all the remaining virtual lanes are left unused.**

In this paper we suggest an enhancement to the fat-tree algorithm that utilizes virtual lanes to improve performance when hot-spots are present. Even though the bisection bandwidth in a fat-tree is constant, hot-spots are still possible and they will degrade performance for flows not contributing to them due to head-of-line blocking. Such a situation may be alleviated through adaptive routing or congestion control, however, these methods are not yet readily available in InfiniBand technology. To remedy this problem, we have implemented an enhanced fat-tree algorithm in OpenSM that distributes traffic across all available virtual lanes without any configuration needed. We evaluated the performance of the algorithm on a small cluster and did a large-scale evaluation through simulations. In a congested environment, results show that we are able to achieve throughput increases up to 38% on a small cluster and from 221% to 757% depending on the hot-spot scenario for a 648-port simulated cluster.

I. Introduction

The fat-tree topology is one of the most common topologies for high performance computing clusters today, and for clusters based on InfiniBand (IB) technology the fat-tree is the dominating topology. This includes large installations such as the Roadrunner, Ranger, and JuRoPa [1]. There are three properties that make fat-trees the topology of choice for high performance interconnects: deadlock freedom, the use of a tree structure makes it possible to route fat-trees without using virtual lanes for deadlock avoidance; inherent fault-tolerance, the existence of multiple paths between individual source destination pairs makes it easier to handle network faults; full bisection bandwidth, the network can sustain full speed communication between the two halves of the network.

For fat-trees, as with most other topologies, the routing algorithm is crucial for efficient use of the underlying topology. The popularity of fat-trees in the last decade led to many efforts trying to improve the routing performance. This includes the current approach that the OpenFabrics Enterprise Distribution (OFED) [2], the de facto standard for IB system software, is based on [3], [4]. These proposals, however, have several limitations when it comes to flexibility and scalability. One problem is the static routing used by IB technology that limits the exploitation of the path diversity in fat-trees as pointed out by Hoefler et al. in [5]. Another problem with the current routing are its shortcomings when routing oversubscribed fat-trees as addressed by Rodriguez et al. in [6]. A third problem is that performance is reduced when the number of compute nodes connected to the tree is reduced as addressed by Bogdanski et al. in [7]. And finally we have the problem of reducing the negative impact of congestion due to head-of-line blocking (HOL) [8]. This is not a routing problem per se as this should be handled by a congestion control mechanism, e.g. the mechanism found in IB [9], [10]. This mechanism, however, has its own set of challenges; one being that it is not generally available for existing IB hardware, another being that it is not yet understood how to configure congestion control for large networks [11]. Therefore, it is important to minimize the problem by other means. We suggest to do this using a combination of efficient routing and virtual lanes in an implementation that can be directly applied to IB or other technologies supporting multiple virtual channels.

Virtual lanes (or channels) were first introduced by Dally in the late eighties [12]. The intention at the time was to alleviate the restriction on routing flexibility that was imposed by deadlock considerations. In 1992 he published an analysis on the effect that virtual channels could have on network performance [13]. In spite of his positive findings, the usage of virtual channels has been confined to flexible routing and service differentiation, both in academia and in the industry. This is partly due to the fact that the analysis in the 1992 paper was based on assumptions that were not true for real technologies - in particular that a source was free to decide virtual lanes at the packet level, and not at the stream level. More recent works have addressed the congestion issue in several other ways. A recent proposal by Rodriguez et al. [14] also addresses this from a routing perspective, but in an application-specific manner and without using virtual lanes. Another approach using a combination of multipath routing and bandwidth estimation was proposed by Vishnu et al. in [15], but this is significantly more complex to implement than our proposal. A third proposal by Escudero-

1530-2075/11 $26.00 © 2011 IEEE
DOI 10.1109/IPDPS.2011.28

Sahuquillo et al. [16] uses multiple queues at the input ports in the switches to avoid HOL, but this is not compatible with any existing network technology and requires new hardware to be built.

In this paper, we present the first results that indicate the gain of adding virtual lanes on a real commercial technology. These results deviate from Dally's in two respects. Firstly, they indicate that the performance gain is significantly bigger than he reported. Secondly,

that most of the gain can be realized by only 2 VLs, making it an obvious, readily available and potent improvement for all existing InfiniBand clusters. To be specific, we analyze the performance of the fat-tree routing algorithm in OpenSM in a hot-spot scenario and suggest a new routing algorithm, vFtree, that improves performance when hot-spots are present by using virtual lanes. Through a prototype implementation in OpenSM we demonstrate, using a small cluster, how virtual lanes can be used to achieve the same effect as IB congestion control. Then we generalize this into a new fat-tree routing algorithm that we evaluate for performance on a small cluster and for performance and scalability through simulations.

The rest of this paper is organized as follows: we introduce the InfiniBand Architecture in Section II followed by a description of fat-tree topologies and routing in Section III and a motivation for our proposal in Section IV. The algorithm is described in Section V. Then we describe the experimental setup in Section VI followed by the performance analysis of the experimental and simulated results in Section VII. Finally, we conclude in Section VIII.

II. The InfiniBand Architecture

InfiniBand is a serial point-to-point full-duplex technology, and the InfiniBand Architecture was first standardized in October 2000 [9]. Due to efficient utilization of host side processing resources, IB is scalable beyond ten thousand nodes with each having multiple CPU cores. The current trend is that IB is replacing proprietary or low-performance solutions in the high performance computing domain [1], where high bandwidth and low latency are the key requirements.

The de facto system software for IB is OFED developed by dedicated professionals and maintained by the OpenFabrics Alliance [2]. OFED is open source and is available for both GNU/Linux and Microsoft Windows. The improved vFtree algorithm that we propose in this paper was implemented and evaluated in a development version of OpenSM, which is a subnet manager distributed together with OFED.

A. Subnet Management

InfiniBand networks are referred to as subnets, where a subnet consists of a set of hosts interconnected using switches and point-to-point links. An IB fabric constitutes of one or more subnets, which can be interconnected together using routers. Hosts and switches within a subnet are addressed using local identifiers (LIDs) and a single subnet is limited to 48151 LIDs.

An IB subnet requires at least one subnet manager (SM), which is responsible for initializing and bringing up the network, including the configuration of all the IB ports residing on switches, routers and host channel adapters (HCAs) in the subnet. At the time of initialization the SM starts in the *discovering state* where it does a sweep of the network in order to discover all switches and hosts. During this phase it will also discover any other SMs present and negotiate who should be the master SM. When this phase is complete the SM enters the *master state*. In this state, it proceeds with LID assignment, switch configuration, routing table calculations and deployment, and port configuration. At this point the subnet is up and ready for use. After the subnet has been configured, the SM is responsible for monitoring the network for changes.

A major part of the SMs responsibility are routing table calculations. Routing of the network aims at obtaining full connectivity, deadlock freedom, and proper load balancing between all source and destination pairs. Routing tables must be calculated at network initialization time and this process must be repeated whenever the topology changes in order to update the routing tables and ensure optimal performance.

B. Virtual Lanes

InfiniBand is a lossless networking technology, where flow-control is performed per *virtual lane* (VL) [13]. The concept of virtual lanes is shown in Fig. 1. VLs are logical channels on the same physical link, but with separate buffering, flow-control, and congestion management resources. Fig. 2 shows an example of per VL credit-based flow-control where VL 0 runs out of credits after cycle 1 (depicted by a bold D) and is unable to transmit until credit arrives in cycle 9 (depicted by a bold C). As the other lanes have sufficient credit, they are unaffected and are able to use the slot that VL 0 would otherwise use. Transmission resumes for VL 0 when credit arrives.

The concept of virtual lanes makes it possible to build virtual networks on top of a physical topology. These virtual networks, or layers, can be used for various purposes such as efficient routing, deadlock avoidance, fault-tolerance and service differentiation. Our proposal exploits VLs for improved routing and network performance.

The VL architecture in IB consists of four mechanisms: *service levels*, *virtual lanes*, *virtual lane weighting*, and *virtual lane priorities*. A service level (SL) is a 4-bit field in the packet header that denotes what type of service a packet shall receive as it travels toward its destination. This is supplemented by up to sixteen virtual lanes. A minimum of two VLs must be supported: VL 0 as the default data lane and VL 15 as the subnet management traffic lane. By

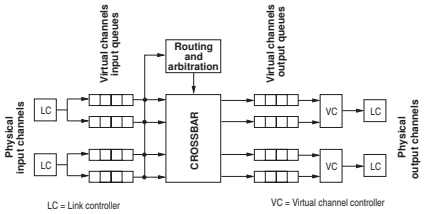


Figure 1. The canonical virtual channel architecture.

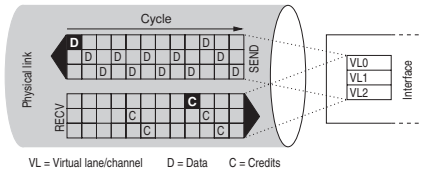


Figure 2. Virtual lane flow control in InfiniBand.

default, the sixteen SLs are mapped to the corresponding VL by the SL number, i.e. SL_i is mapped to VL_i. If a direct SL to VL mapping is not possible, the SL will be degraded according to a SL to VL mapping table. In the worst case only one data VL is supported and all SLs will be mapped to VL 0. In current IB hardware it is common to support nine VLs: one for management and eight for data.

Each VL can be configured with a weight and a priority, where the weight is the proportion of link bandwidth a given VL is allowed to use and the priority is either high or low. Our contribution in this paper will not make use of the weight and priority features in IB, we will use a direct SL to VL mapping and equal priority for all VLs. For more details about the weight and priority mechanisms consult [9], [17].

III. Fat-Trees

The fat-tree topology was introduced by C. Leiserson in 1985 [18], and has since then become a common topology in high performance computing (HPC). The fat-tree is a layered network topology with link capacity equal at every tier (applies for balanced fat-trees), and is commonly implemented by building a tree with multiple roots, often following the m-port n-tree definition [19] or the k-ary n-tree definition [20].

With the introduction of IB the fat-tree became the topology of choice due to its inherent deadlock freedom, fault tolerance, and full bisection bandwidth properties. It is used in many of the IB installations in the Top500 List, including supercomputers such as Los Alamos National Laboratory's Roadrunner, Texas Advanced Computing Center's Ranger, and Forschungszentrum Juelich's JuRoPa [1]. The Roadrunner differs from the two other examples in that it uses an oversubscribed fat-tree [21]. By carefully designing an oversubscribed fabric the implementation costs of an HPC cluster can be significantly reduced with only a limited loss in the overall application performance [22].

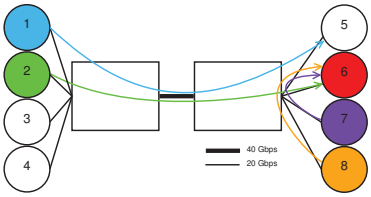


Figure 3. A simple congestion control experiment.

For fat-trees, as for most other network topologies, the routing algorithm is essential in order to exploit the available network resources. In fat-trees the routing algorithm consists of two distinct stages: the upward stage in which the packet is forwarded from the source, and the downward phase when the packet is forwarded toward the destination. The transition between those two stages occurs at the least common ancestor, which is a switch that can reach both the source and the destination through its downward ports. The algorithm ensures deadlock-freedom, and the IB implementation available in OpenSM also ensures that every path toward the same destination converges at the same root node, which causes all packets toward that destination to follow a single dedicated path in the downward direction [4]. By having dedicated downward paths for every destination, contention in the downward stage is effectively removed (moved to the upward stage), so that packets for different destinations have to contend for output ports in only half of the switches on their paths. In oversubscribed fat-trees, the downward path is not dedicated and is shared by several destinations.

Since the fat-tree routing algorithm only requires a single VL, the remaining virtual lanes are available for other purposes such as quality of service or for reducing the negative impact of congestion induced by the head-of-line blocking as we will do in this paper.

IV. Motivation

Algorithmic predictability of network traffic patterns is reduced with the introduction of virtualization and many-cores systems. When multiple virtualized clients reside on the same physical hardware, the network traffic becomes an overlay of multiple traffic patterns that might lead to hot-spots in the network. A *hot-spot* occurs if multiple flows are destined toward a single endpoint. Common sources for hot-spots include complex traffic patterns due to virtualization, migration of virtual machine images, checkpoint and restore mechanisms for fault tolerance, and storage and I/O traffic.

When a hot-spot exists in a network, the flows designated for the hot-spot might reduce the performance for other flows, called *victim flows*, not designated to the hot-spot. This is due to the head-of-line (HOL) blocking phenomena created by the congested hot-spot [8]. One way to avoid this problem is to use a congestion control mechanism such

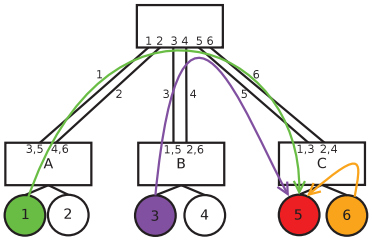
as the one recently specified and implemented for IB [9]. This mechanism was evaluated in hardware and shown to be working by Gran et al. [11], however this solution is not yet generally available. Furthermore, the selection of the appropriate CC parameters highly depends on the topology, is poorly understood, and incorrect parameters might lead to performance degradation.

It is a well known fact that using virtual lanes is one of the ways to alleviate network congestion [12], [13]. To demonstrate this as a plausible solution to alleviate congestion, we performed three simple hardware experiments on the topology presented in Fig. 3. In all the three experiments we used the following synthetic communication pattern: 1-5, 2-6, 8-6 and 7-6 which creates a hot-spot on endpoint 6. In Fig. 5a we show the per flow throughput for the first experiment where we observe that the victim flow (1-5) is affected by the congestion to the same degree as the flows contributing to congestion, even though the flow is not destined to the congested endpoint. This is caused by two main reasons. Firstly, the victim flow (1-5) is sharing the 32 Gb/s (effective bandwidth) link with the contributors to the congestion. Secondly, the HOL blocking reduced its bandwidth to the same share of switch-to-switch link bandwidth as the contributor, which is approximately 4.5 Gb/s. In Fig. 5b we present the result from the second experiment where the IB congestion control mechanism was turned on and configured using the parameters given by Gran et al. [11]. Now we observe that the victim flow is not deteriorated by HOL blocking and it manages to get about 13 Gb/s independent of other traffic flows.

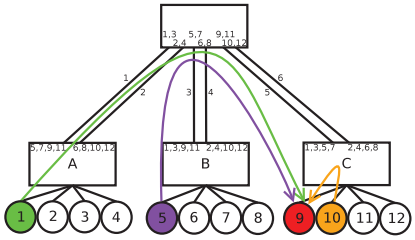
However, some oscillations occur among the flows due to the fact that the congestion control mechanism is dynamically adjusting the injection rate of the senders. In Fig. 5c we show the result of the third experiment where the victim flow (1-5) was manually assigned to a separate virtual lane. The measured result is similar to the experiment with IB CC turned on. In fact, the total network throughput shown in Fig. 6 is slightly better because we do not see the oscillations of IB CC mechanism. However, the traffic patterns used in this scenario are artificial and the victim flow is known in advance. Since we are unable to adapt to the congestion with this approach, we must rely on the statistical probability of improvements occurring when we distribute the traffic across a set of virtual lanes.

V. Design of Congestion Avoidance Mechanism Based on Routing

In this section, we propose a generic and simple routing-based congestion avoidance mechanism, which may easily be applied to high-performance network technologies supporting VLs. Our proposed routing algorithm uses the available virtual lanes to reduce the hot-spot problem by distributing *source destination* pairs across all available VLs. The approach is generic and can be applied to any



(a) A hot-spot scenario in a non-oversubscribed fat-tree.



(b) A hot-spot scenario in a oversubscribed fat-tree.

Figure 4. Experiment scenarios for the hardware testbed.

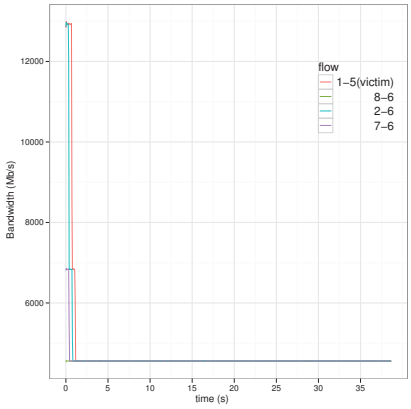
topology, but in this paper we focus on fat-trees because their regular structure makes it straightforward to distribute the destinations across VLs and they do not require VLs to be routed deadlock-free.

A. vFtree - Fat-tree Routing using Virtual Lanes

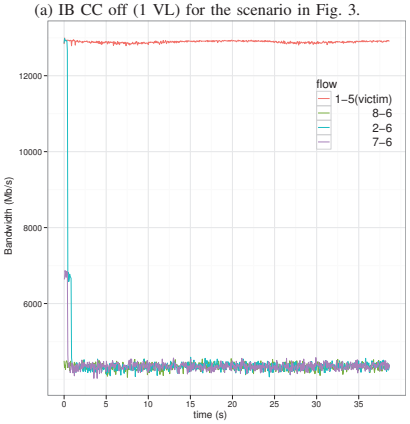
Routing in IB is a table lookup algorithm where each switch holds a forwarding table containing a linear list of LIDs (destination addresses) and the corresponding output ports. Our proposed algorithm is an extension of the current fat-tree routing algorithm that is available in OpenSM 3.2.5. The main feature of the new algorithm is to isolate the possible endpoint congestion flows by distributing *source-destination* pairs across virtual lanes. Our simple experiments in Section IV showed that virtual lanes can be used as a mechanism to alleviate endpoint congestion if the victim flow is identified. Unfortunately, as we have no reliable way of identifying victim flows, we propose to evenly distribute all *source-destination* pairs that share the same link in the upward direction across the available VLs.

The current fat-tree routing algorithm proposed by Zahavi et al. [4] is already optimized to avoid the contention for shift all-to-all communication traffic patterns. In a fully populated and non-oversubscribed fat-tree, this algorithm is equalizing the load on each port and always selects a different upwards link for destinations located on the same leaf switch. For example, in Fig. 4a destination 3 is reached from leaf switch A using link 1, while destination 4 is reached using link 2. It means that the destinations sharing the same upstream link are always located on different leaf switches.

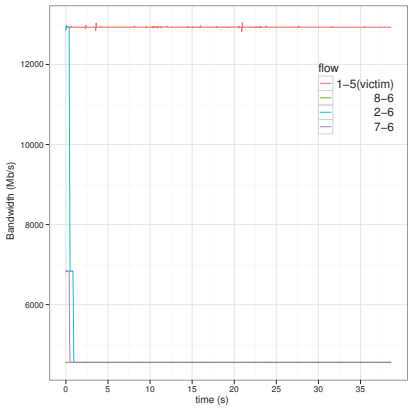
Our algorithm works on top of this and distributes all



(a) IB CC off (1 VL) for the scenario in Fig. 3.



(b) IB CC on for the scenario in Fig. 3.



(c) IB CC off (2 VLs) for the scenario in Fig. 3.

Figure 5. Per flow throughput comparison for IB CC experiment on Fig. 3.

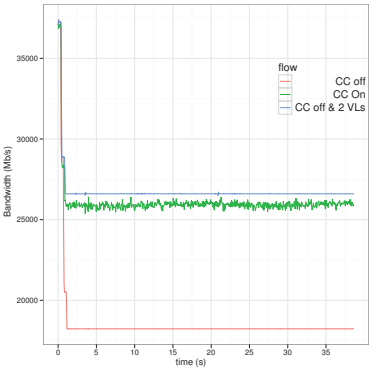


Figure 6. The total network throughput for experiment in Fig. 3.

destinations sharing the same upstream link across different virtual lanes. In detail, this means that from switch A we reach destination 3 using link 1 and VL 0, while destination 5 also uses link 1, but has VL 1 assigned. Consequently, if one of the designated destinations is the contributor to an endpoint hot-spot, the other destination flow (victim flow) sharing the same upstream port is not affected by HOL blocking because it uses a different virtual lane with its own buffering resources.

Ideally, the number of VLs required by our algorithm depends on the number of destinations that share the same upstream link, which is equivalent to the $N-1$ where N is the number of leaf switches. The $N-1$ virtual lanes cover all the traffic routed to destinations connected to other leaf switch except those destinations that are connected to the same leaf switch as the traffic source. In the implementation, however, the number of virtual lanes required is higher due to a requirement in the IB specification. The specification and the current implementation requires that the VL used from A to B is symmetric, which means that the communication from A to B and from B to A must be able to use the same VL.

The core functionality of the vFtree routing is divided into two algorithms as presented in Algo. 1 and Algo. 2. The former one distributes leaf $< sw_{src}, sw_{dst} >$ pairs across the available virtual lanes.

The outer ***for*** loop iterates through all the source leaf switches and the inner ***for*** loop iterates through all the destination leaf switches. In the inner ***for*** loop we check whether a VL has been assigned to a $< sw_{src}, sw_{dst} >$ pair, and, if not, we assign a VL accordingly. The requirement is that the VL assigned to a $< sw_{src}, sw_{dst} >$ must be the same as the VL assigned to $< sw_{dst}, sw_{src} >$ pair. The first ***if...else*** block starting at line 2 determines the initial *vl* value used for the inner ***for*** loop so the overlapping of VLs is minimized. The max_{vl} variable is an input argument for OpenSM that provides flexibility to the cluster administrator

who may wish to reserve some VLs for quality of service (QoS).

When a connection (Queue Pair) is being established in IB, the source node will query the path record and then Algo. 2 is executed. The arguments passed to this function are the source and destination addresses. Using these values, the source node obtains the VL from the array generated in Algo. 1 to be used for communication with the destination node.

Algorithm 1 Assign Virtual Lanes

Require: Routing table has been generated.
Ensure: Symmetrical VL for every <src, dst> pair.

```
for sw_src = 0 to max_leaf_switches do
  if odd(sw_src × 2/max_vl) then
    vl = ((sw_src × 2) % max_vl) + 1
  else
    vl = (sw_src × 2) % max_vl
  end if
  for sw_dst = 0 to max_leaf_switches do
    if sw_dst > sw_src then
      if VL[sw_src][sw_dst].set = FALSE then
        VL[sw_src][sw_dst].vl = vl
        VL[sw_src][sw_dst].set = TRUE
      end if
      if VL[sw_dst][sw_src].set = FALSE then
        VL[sw_dst][sw_src].vl = vl
        VL[sw_dst][sw_src].set = TRUE
      end if
      vl = incr(vl) % max_vl
    else if sw_dst == sw_src then
      VL[sw_dst][sw_src].vl = 0
      VL[sw_dst][sw_src].set = TRUE
    end if
  end for
end for
```

However, in OFED one major difference between vFtree and the conventional fat-tree routing is that for an application to acquire the correct SL in a topology routed using vFtree, a communication manager (CM) needs to be queried. The reason for that is only the CM can return the SL that was set up by the SM, and this SL implies which VL should be used. Otherwise, the default VL would be used, which is VL 0.

Algorithm 2 Get Virtual Lanes (LID_{src},LID_{dst})

```
dst_id = get_leaf_switch_id(LID_dst)
src_id = get_leaf_switch_id(LID_src)
return VL[src_id][dst_id]
```

B. Limitations

The main limitation of our approach is related to the number of VLs used. The IB specification defines 16 VLs, however, the actual implementation in today's hardware is limited to 8 VLs. This is insufficient to cover all the possibilities of endpoint congestion in a large-scale cluster which requires $N - 1$ VLs where N is the number of leaf switches. But as our results in Section VII show, large improvements are still possible with only two VLs. Another limitation is related to the use of VLs for other purposes such as QoS. QoS can be used together with vFtree routing, but then the number of SLs is reduced from 8 to 4 because for each SL two VLs are consumed by vFtree routing. For topologies other than fat-tree, which might use VLs for deadlock-free routing, the number of available SLs may be further reduced.

Furthermore, the assumption made for the vFtree algorithm is that the end node distribution is uniform. This is because the current version of the fat-tree routing algorithm has limitations when it comes to properly balancing the paths when the end node distribution is nonuniform as mentioned in [7].

VI. Experiment Setup

To evaluate our proposal we have used a combination of simulations and measurements on a small IB cluster. In the following subsections, we present the hardware and software configuration used in our experiments.

A. Experimental Test Bed

Our test bed consists of twelve nodes and four switches. Each node is a Sun Fire X2200 M2 server that has a dual port Mellanox ConnectX DDR HCA with an 8x PCIe 1.1 interface, one dual core AMD Opteron 2210 CPU, and 2GB of RAM. The switches are two 24-port Infiniscale-III DDR switches and two 36-port Infiniscale-IV QDR switches which we used to construct the topologies illustrated in Fig. 3 and Fig. 4. All the hosts have Ubuntu Linux 8.04 x86_64 installed with kernel version 2.6.24-24-generic and the subnet is managed by a modified version of OpenSM 3.2.5 that contains the implementation of the vFtree routing algorithm. Our *Perftest* [23] was also modified to support regular bandwidth reporting and continuous sending of traffic at full link capacity. The modified *Perftest* is used to generate the hot-spots shown in Fig. 4a and Fig. 4b.

B. Simulation Test Bed

To perform large-scale evaluations and verify the scalability of our proposal, we developed an InfiniBand model for the OMNeT++ simulator. The model contains an implementation of HCAs and switches with routing tables and virtual lanes. The network topology and the routing tables were generated using OpenSM and converted into OMNeT++ readable format in order to simulate real-world

systems. In the simulator, every source-destination pair has a VL assigned according to Algo. 1.

The simulations were performed on a 648-port fat-tree topology, which is the largest possible 2-stage fat-tree topology that can be constructed using 36-port switch elements. When fully populated this topology consists of 18 root switches and 36 leaf switches. Additionally, we performed the simulations of a 648-port topology that had an oversubscription ratio of 2:1. This was achieved by removing half of the root switches from the topology (9 switches). We chose a 648-port fabric because it is a common configuration used by switch vendors in their own 648-port systems [24], [25], [26]. Additionally, such switches are often connected together to form larger installations like JuRoPa. For the simulations we used a nonuniform traffic pattern, where 5% of all packets generated by a computing node was sent to a predefined hot-spot and the rest of the traffic was sent to a randomly chosen node. Additionally, we used multiple localized hot-spots by partitioning the network into three or nine segments, which corresponded to the physical features of the 648-port switch built in such a way that four leaf switch elements are placed on a single modular card.

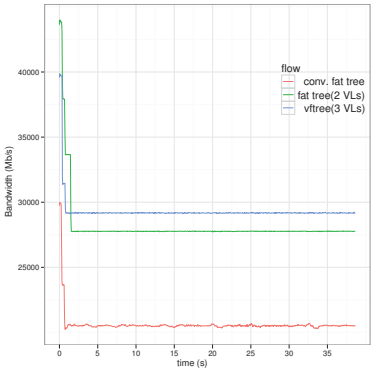
Each simulation run was repeated eight times with different seeds and the average of all simulation runs was taken. The packet size was 2 kB for every simulation. Furthermore, we have tuned the simulator to the hardware so we could observe the same trends when performing the data analysis. The results obtained through simulations exhibited the same trends as the results obtained from the IB hardware, with a maximum difference of 12% between the hardware and simulations.

VII. Performance Evaluation

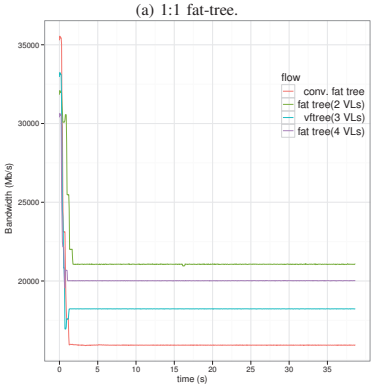
Our performance evaluation consists of measurements on an experimental cluster and simulations of large-scale topologies. For the cluster measurements we use the *per flow throughput* and *the total network throughput* as our main metrics to compare the performance of our proposed vFtree algorithm and the existing fat-tree algorithm. Additionally we use the results from the HPCC benchmark on certain scenarios to show how the algorithm impacts application traffic. For the simulations we use the *achieved average throughput per end node* as the metric for measuring the performance of the vFtree algorithm on the simulated 648-port topology. In both experimental cluster and simulations, all traffic flows are started at the same time and they are based on transport layer of the IB stack.

A. Experimental results

We carried out two different experiments on two different configurations which are a *non-oversubscribed fat-tree* as shown in Fig. 4a and *2:1 oversubscribed fat-tree* as shown in Fig. 4b. In the first experiment for the first configuration, a collection of synthetic traffic patterns ({1-5, 2-3, 3-5, 4-1, 6-5}) was selected to generate a hot-spot. In Fig. 4a node 5 is the hot-spot and the nodes 1, 3 and 6 are the contributors to the hot-spot. The flows 2-3 and 4-1 represent the victim flows. The purpose of the first experiment is to illustrate the negative impact the HOL blocking has on the victim flows. Additionally, it also shows how the vFtree routing algorithm avoids the negative effects of endpoint congestion. In the second experiment, we replaced the victim flows with the *HPC challenge* benchmark [27]. Our HPCC benchmark b_eff test suite was modified to generate 5000 random traffic patterns in order to obtain a more accurate result for randomly ordered ring bandwidth test. Even though the congested flows are still synthetically generated, this scenario resembles the network environment that an application could experience during congestion.



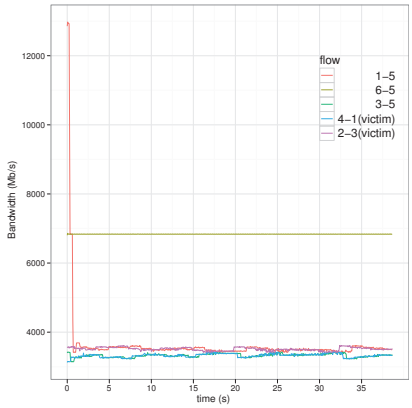
(a) 1:1 fat-tree.



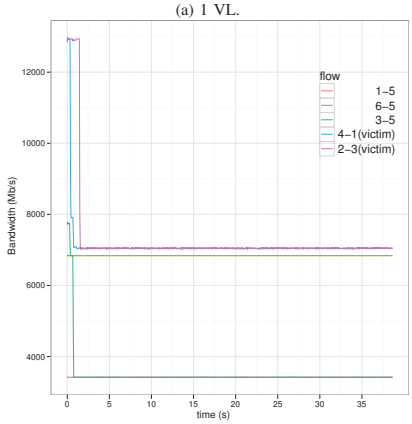
(b) 2:1 oversubscribed fat-tree.

Figure 7. Total network throughput using 1 to 3 VLs for the non-oversubscribed fat-tree and 1 to 4 VLs for the oversubscribed one (Fig. 4).

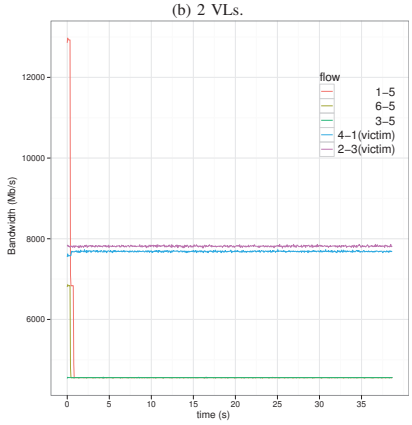
On the second configuration, shown in Fig. 4b, we repeated both of the experiments. A collection of synthetic traffic patterns {1-9, 6-9, 8-11, 10-9} was used for the first experiment. In this case, the hot-spot was at node 9 and the contributors were the nodes 1, 6 and 10. The flow 8-11



(a) 1 VL.



(b) 2 VLs.



(c) 3 VLs.

Figure 8. Per flow throughput for the fat-tree in Fig. 4a using 1 to 3 VLs.

represents the victim flow.

The first experiment (synthetic traffic) is described in Sections VII-A1 for non-oversubscribed fat-tree and VII-A2 for the oversubscribed one, and the second experiment (HPCC benchmark) is described in Section VII-A3 for both fat-tree topologies.

1) Non-oversubscribed fat-tree: Fig. 8 shows the per flow throughput of the first experiment when using 1 to 3 VLs. Fig. 8a shows the per flow throughput for the conventional fat-tree routing algorithm using one VL. The congestion towards node 5 blocks the traffic on physical link 1 and 3, and makes flows 4-1 and 2-3 victim flows. For these flows the throughput is less than half of the bandwidth that is available in the network, but due to the HOL blocking the bandwidth of flow 2-3 is reduced to the bandwidth that the congested flow 1-5 achieves across link 1. For the same reason flow 4-1 is reduced to the bandwidth of the congested flow 3-5. Furthermore, we also observe that, owing to the parking lot problem [28], flow 6-5 gets a higher share of the bandwidth toward destination 5 than flow 1-5 and 3-5.

If we manually assign the victim flows to a different VL, the situation improves as shown in Fig. 8b. The victim flows are able to avoid the HOL blocking, giving each of them an effective throughput of approximately 7 Gb/s, which corresponds to the actual available bandwidth in the network. The parking lot problem, however, is still present and we can see unfairness among the flows toward the endpoint hot-spot.

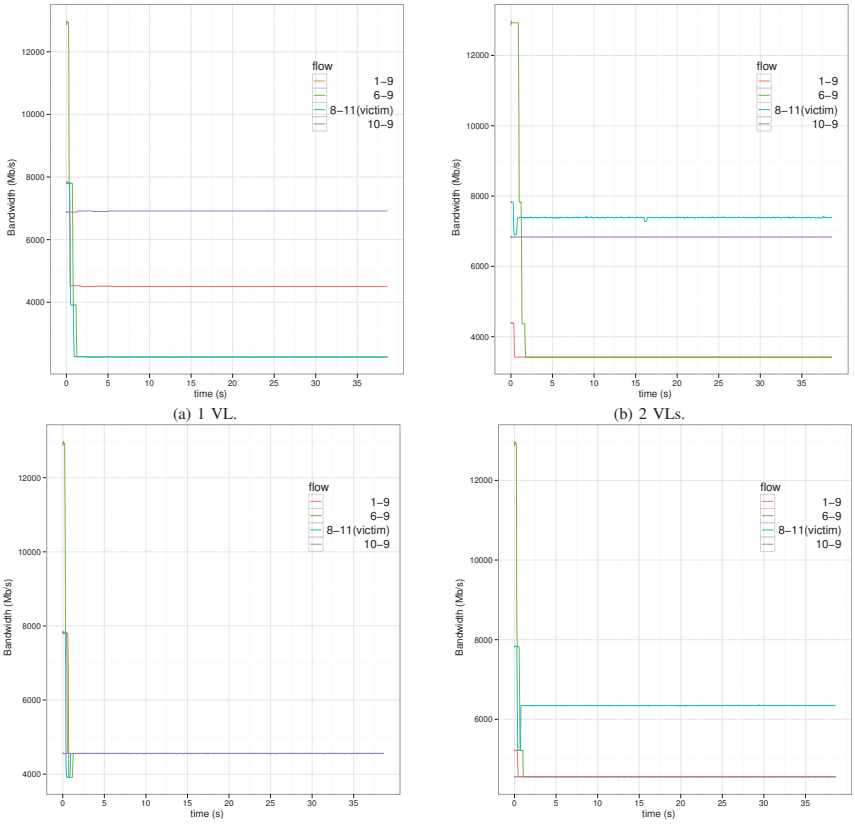
Fig. 8c shows the results of the repeated experiment with 3 VLs where both the HOL blocking of the victim flows and the parking lot problem toward the congested endpoint are solved. The reason for that is that the vFtree algorithm placed the routes for the victim flows in their own separate VLs, which solves the HOL blocking. Additionally, it placed the flows 1-5 and 3-5 on different VLs making the link arbitration between the sources 1,3, and 6 fair at switch C.

To summarize, we showed that the vFtree algorithm reduced both the HOL blocking and the parking lot problem when applied to fat-tree networks. The overall increase in total network throughput is approximately 38% when compared to the original fat-tree routing algorithm as shown in Fig. 7a.

2) 2:1 Oversubscribed network: In an oversubscribed network, the victim flows may suffer from HOL blocking in two different ways.

The first case is similar to the non-oversubscribed network where the performance reduction of the victim flow is due to a shared upstream link with the contributors to congestion.

In the second case, the victim flow shares both the upstream and the downstream link toward the hot-spot with the contributors, even though the victim flow is eventually routed to a different destination. In this section, we focus on the latter case because for the former case the results will be similar to the non-oversubscribed network scenario from Section VII-A1.



(a) 1 VL. (b) 2 VLs.

(c) 3 VLs. (d) 4 VLs.

Figure 9. Per flow throughput for the fat-tree in Fig. 4b using 1 to 4 VLs.

Fig. 9a shows the per flow throughput for the conventional fat-tree routing algorithm where the congestion in the network (Fig. 4b) blocked the victim flow (8-11). Among the congested flows, flow 6-9 obtains the lowest bandwidth because it also shares the upstream/downstream path with the victim flow. As a result, this also affects the victim flow (8-11) because it receives the same bandwidth across link 5 as flow 6-9 due to the HOL blocking, approximately 2.1 Gb/s.

If we manually assign the victim flow to a different VL, the situation improves as Fig. 9b shows. The results show that the victim flow (8-11) is able to avoid the HOL blocking and obtains approximately 7.5 Gb/s, the actual effective bandwidth that is available for this flow. As previously, unfairness exists among the contributing flows (1-9, 6-9 and 10-9) due to the parking lot problem.

As shown in Fig. 9d, the parking lot problem is resolved with 4 VLs where each of the contributors of the congestion manages to obtain 1/3 of the effective link bandwidth at approximately 4.5 Gb/s. Another observation is that the victim flow is getting a slightly lower bandwidth. This is due to the fact that the victim flow shares the downstream link with the rest of the congestion contributors. With a separate VL for each congestion contributor, flows 1-9 and 6-9 have an increased link bandwidth and, consequently, reduce the victim flow's bandwidth as shown in Fig. 9d.

In an oversubscribed fat-tree, utilizing more virtual lanes does not necessarily mean increasing the performance for certain types of traffic patterns. As shown in Fig. 7b, the total network bandwidth is higher with only 2 VLs when compared to the bandwidth obtained with 4 VLs. This is because with separate VL for each congestion contributor, the parking lot problem is resolved but the share of the link bandwidth given to the victim flow is reduced. Furthermore, if we would like to consider both cases of victim flow occurrence, it would require more VLs. Thus, in order to make our algorithm more predictable, we have decided to support only the first case of the victim flow as discussed in

Table I
RESULTS FROM THE HPC CHALLENGE BENCHMARK WITH CONVENTIONAL FAT-TREE ROUTING AND VFTREE.

Network latency and throughput	a) conventional Ftree	b) vFtree	c) Improvement
Min Ping Pong Lat. (ms)	0.002116	0.002116	0.0%
Avg Ping Pong Lat. (ms)	0.022898	0.013477	41.14%
Max Ping Pong Lat. (ms)	0.050500	0.043005	14.84%
Naturally Ordered Ring Lat. (ms)	0.021791	0.014591	33.04%
Randomly Ordered Ring Lat. (ms)	0.024262	0.015826	34.77%
Min Ping Pong BW (MB/s)	94.868	345.993	264.71%
Avg Ping Pong BW (MB/s)	573.993	830.909	44.75%
Max Ping Pong BW (MB/s)	1593.127	1594.338	0.07%
Naturally Ordered Ring BW (MB/s)	388.969246	454.236253	16.78%
Randomly Ordered Ring BW (MB/s)	331.847978	438.604531	32.17%

Table II
RESULTS FROM THE HPC CHALLENGE BENCHMARK WITH CONVENTIONAL FAT-TREE ROUTING AND VFTREE IN AN OVERSUBSCRIBED NETWORK.

Network latency and throughput	a) conventional Ftree	b) vFtree	c) Improvement
Min Ping Pong Lat. (ms)	0.002176	0.002176	0.0%
Avg Ping Pong Lat. (ms)	0.015350	0.009491	38.17%
Max Ping Pong Lat. (ms)	0.050634	0.043496	14.10%
Naturally Ordered Ring Lat. (ms)	0.021601	0.015616	27.70%
Randomly Ordered Ring Lat. (ms)	0.023509	0.016893	28.14%
Min Ping Pong BW (MB/s)	126.135	342.553	171.58%
Avg Ping Pong BW (MB/s)	825.874	1031.098	24.85%
Max Ping Pong BW (MB/s)	1594.186	1594.338	0.01%
Naturally Ordered Ring BW (MB/s)	369.021995	436.588321	18.31%
Randomly Ordered Ring BW (MB/s)	254.737276	355.412454	39.52%

earlier in this section. Nevertheless, we still manage to get about 20% improvement with vFtree routing algorithm that uses 3 VLs when compare with conventional fat-tree routing as illustrated in Fig. 7b. The reason behind this is that the parking lot problem is solved when each of the contributors to the hot-spot has a fair share of link bandwidth, but HOL blocking for the victim flow is not avoided. As observed on Fig. 9c, each of the flows (including the victim flow) obtains approximately 4.5 Gb/s of effective link bandwidth.

3) HPC challenge Benchmark: The second experiment is a combination of the I/O traffic generated by Perftest [23] and the application traffic generated by the HPCC benchmark [27]. The endpoint hot-spot is created using Perftest by running the traffic pattern presented in Fig. 4a for the non-oversubscribed configuration and in Fig. 4b for the 2:1 oversubscribed configuration). Simultaneously, we are running the HPCC benchmark in order to study the impact of congestion on the traffic generated by the HPCC benchmark.

Table I shows the comparison of the HPCC b_eff results between the conventional fat-tree routing and our vFtree routing algorithm in the presence of congestion in a non-oversubscribed network. The most interesting observation is that the randomly ordered ring bandwidth increased by 32.1% with our vFtree routing algorithm which uses only 3 VLs. We can see the improvement for all the latency and bandwidth tests, which is expected, as they correspond to the synthetic traffic patterns experiment that was carried out in the previous section. The results for the oversubscribed network are presented in Table II and the same trends are visible as for the non-oversubscribed network. We also managed to achieve an improvement in most of the latency and bandwidth tests when using our vFtree routing algorithm.

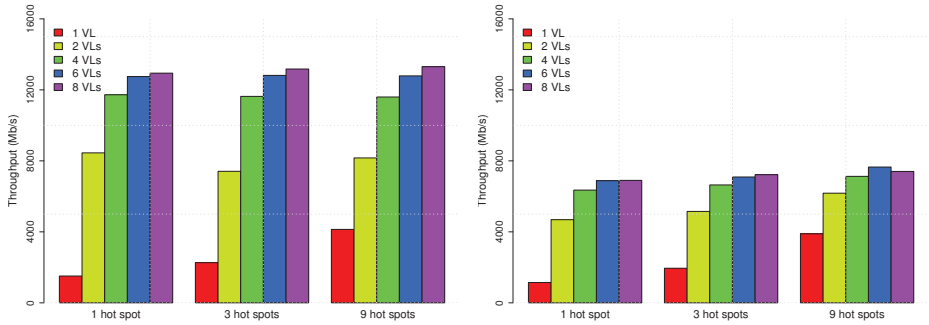
These results clearly illustrate the performance gain with our vFtree routing algorithm from the application traffic pattern's point of view.

B. Simulation results

An important question is how well the presented algorithm scales. Specifically, the purpose of the simulations was to show that the same trends exist when the number of nodes is large and the network topology corresponds to real systems. We performed the simulations on a 648-port switch without oversubscription and with 2:1 oversubscription.

1) Non-oversubscribed network: For a single hot-spot scenario, node 1 connected to switch 1 on modular card 1 (see Section VI-B for modular card definition) was the hot spot, and all the other nodes in the fabric were the contributors to the hot-spot. In case of three hot-spots, nodes 1 (modular card 1, switch 1), 217 (modular card 4, switch 1), and 433 (modular card 7, switch 1) were the hot-spots and the contributors were the nodes connected to modular cards 1-3 for node 1, modular cards 4-6 for node 217, and modular cards 7-9 for node 433. For a nine hot-spot scenario, the hot-spots were nodes 1, 73, 145, 217, 289, 361, 433, 505 and 577 connected to switch 1 at each modular card, and the contributors for each hot-spot were all the the other nodes connected to the same modular card as the hot-spot. In each scenario, the contributors sent 5% of their overall traffic to the hot-spot and 95% of other traffic to any other randomly chosen node in the fabric (it could also be any of the predefined hot-spots).

For the case presented on Fig. 10a, we observe that a single hot-spot dramatically decreases the average throughput per node because of the large number of victim flows. If more hot-spots are added, but the contributor traffic is localized (i.e. less victim flows), we observe that the



(a) Simulation results for 648-port switch with no oversubscription. (b) Simulation results for 648-port switch with 2:1 oversubscription.

Figure 10. Simulation results for 648-port switch.

throughput per node increases. The most important observation is the fact that every additional VL for data also reduces the number of victim flows, therefore increasing the network performance. The largest relative increase in the performance is obtained when adding a second VL. The relative improvement when compared with 1 VL is: 459% for 2 VLs, 676% for 4 VLs, 744% for 6 VLs and 757% for 8 VLs. The improvements when hot-spots are localized are smaller because of the fewer victim flows, which is best illustrated by the example from nine hot-spots case when comparing 1 VL to 8 VLs scenarios where we see an improvement of 221%. It also needs to be mentioned that the difference in average throughput per node between 6 VLs and 8 VLs is in a range of 400 Mb/s, so every additional VL provides a smaller throughput increase. Furthermore, it has to be noted that due to the randomness of the traffic there may exist more hot-spots in the network, and these hot-spots are not necessarily localized, which would explain the drops in average throughput per node for 2 VL scenario (yellow bars).

2) Oversubscribed network: Fig. 10b shows the results of a similar experiment performed on a 648-port network with 2:1 oversubscription. For every scenario, the hot-spots were chosen in the exact same manner as for the non-oversubscribed network and the same traffic patterns were used. We observe that the average throughput per node is generally halved when compared to the previous experiment with a non-oversubscribed topology. This is caused by the fact that the downward paths in the tree are shared by two destinations. The improvements when using 8 VLs compared to 1 VL are 503%, 270% and 90% for one, three or nine hot-spots respectively. Even though the result for nine hot-spots with 6 VLs was better (97% gain when compared with 1 VL) than with 8 VLs, we may assume this was a result of the randomness of the traffic as described in the previous section. This shows that the presented algorithm also reduced the number of the victim flows in an oversubscribed tree scenario, which makes it usable not only for small topologies, but also for real-world fabric examples.

To summarize, the large differences between the hardware results and the simulation results can be attributed to the fact that the simulated topologies are much larger in size than the hardware topologies we were able to construct. In the hardware the 38% improvement is visible for only two contributors sending to a single hot-spot. In the simulation, the worst case is if 5% of all 648 nodes are sending to a single hot-spot (plus any of the 95% of other nodes with a probability of $1/648$). It means we may safely assume that at any point in time at least 32 nodes are the contributors to the hot-spot. Therefore, every additional VL improves the network performance by reducing the number of victim flows, and because there are so many contributors and many more victim flows, the improvement is much larger for large-scale scenarios than for smaller topologies.

VIII. Conclusions and Future Work

In this paper, we demonstrated that by extending the fat-tree routing with VLs, we are able to dramatically improve the network performance in presence of hot-spots. Our solution is not only inexpensive, scalable, and readily available, but also does not require any additional configuration. By implementing the vFtree algorithm in OpenSM, we have shown that it can be used with the current state-of-the-art technology, and that the achieved improvements vary from 38% for small hardware topologies to 757% for large-scale simulated clusters when compared with the conventional fat-tree routing. Furthermore, the ideas from our proposal can be ported to other types of routing algorithms and similar improvements would be expected. Moreover, the solution is not restricted to InfiniBand technology only, and the concept can be applied to any other interconnects that support VLs.

In future, we plan to expand this solution to be able

to dynamically reconfigure the balancing of the network in case of faults, and to contribute our modifications to OpenFabrics community. Looking further ahead, we will also propose extensions to better support oversubscribed fat-trees by distributing the VLs in the downward direction.

REFERENCES

[1] "Top 500 supercomputer sites," http://www.top500.org/, Jun. 2010.

[2] "The OpenFabrics Alliance," http://openfabrics.org/, Sep. 2010.

[3] C. Gómez *et al.*, "Deterministic versus Adaptive Routing in Fat-Trees," in *Proceedings of the IEEE International Parallel and Distributed Processing Symposium*. IEEE CS, 2007. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.96.5710

[4] E. Zahavi *et al.*, "Optimized InfiniBand TM fat-tree routing for shift all-to-all communication patterns," *Concurrency and Computation: Practice and Experience*, vol. 22, no. 2, pp. 217–231, 2009. [Online]. Available: http://www3.interscience.wiley.com/journal/122677542/abstract

[5] T. Hoefler *et al.*, "Multistage switches are not crossbars: Effects of static routing in high-performance networks," in *Cluster Computing, 2008 IEEE International Conference on*, 2008, pp. 116–125.

[6] G. Rodriguez *et al.*, "Oblivious Routing Schemes in Extended Generalized Fat Tree Networks," *IEEE International Conference on Cluster Computing and Workshops, 2009. CLUSTER '09.*, pp. 1–8, 2009.

[7] B. Bogdanski *et al.*, "Achieving Predictable High Performance in Imbalanced Fat Trees," in *Proceedings of the 16th International Conference on Parallel and Distributed Systems (ICPADS'10) - to appear*, 2010.

[8] G. F. Pfister and A. Norton, ""Hot Spot" Contention and Combining in Multistage Interconnection Networks," *IEEE Transactions on Computers*, vol. C-34, no. 10, pp. 943–948, 1985.

[9] *InfiniBand architecture specification*, 1st ed., InfiniBand Trade Association, November 2007.

[10] G. Pfister *et al.*, "Solving Hot Spot Contention Using InfiniBand Architecture Congestion Control," Jul. 2005. [Online]. Available: http://www.cercs.gatech.edu/hpidc2005/presentations/GregPfister.pdf

[11] E. G. Gran *et al.*, "First Experiences with Congestion Control in InfiniBand Hardware," in *Proceeding of the 24th IEEE International Parallel & Distributed Processing Symposium*, 2010.

[12] W. J. Dally and C. L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Transactions on Computers*, vol. 36, no. 5, pp. 547–553, May 1987.

[13] W. J. Dally, "Virtual-Channel Flow Control," *IEEE Transactions on Parallel and Distributed Systems*, vol. 3, pp. 194–205, Mar. 1992.

[14] G. Rodriguez *et al.*, "Exploring pattern-aware routing in generalized fat tree networks," in *Proceedings of the 23rd international conference on Supercomputing*. New York: ACM, 2009, pp. 276–285. [Online]. Available: http://portal.acm.org/citation.cfm?id=1542275.1542316

[15] A. Vishnu *et al.*, "Topology agnostic hot-spot avoidance with InfiniBand," *Concurrency and Computation: Practice and Experience*, vol. 21, no. 3, pp. 301–319, 2009.

[16] J. Escudero-Sahuquillo *et al.*, "An Efficient Strategy for Reducing Head-of-Line Blocking in Fat-Trees," in *Lecture Notes in Computer Science*, D'Ambra, Pasqua And Guarracino, Mario And Talia, Domenico, Ed., vol. 6272. Springer Berlin / Heidelberg, 2010, pp. 413–427.

[17] S.-A. Reinemo *et al.*, "An overview of QoS capabilities in InfiniBand, Advanced Switching Interconnect, and Ethernet," *IEEE Communication Magazine*, vol. 44, Jul. 2006.

[18] C. E. Leiserson, "Fat-Trees: Universal Networks for Hardware-Efficient Supercomputing," *IEEE Transactions on Computers*, vol. C-34, pp. 892–901, 1985.

[19] X.-Y. Lin *et al.*, "A multiple lid routing scheme for fat-tree-based infiniband networks," *Parallel and Distributed Processing Symposium, International*, vol. 1, p. 11a, 2004.

[20] F. Petrini *et al.*, "K-ary N-trees: High Performance Networks for Massively Parallel Architectures," Dipartimento di Informatica, Universita di of Pisa, Tech. Rep., 1995.

[21] K. J. Barker *et al.*, "Entering the petaflop era: the architecture and performance of Roadrunner," *SC Conference*, vol. 0, pp. 1–11, 2008.

[22] "HPC Fabric Analysis - Designed for Reduced Costs and Improved Performance," QLogic Corporation, 2010.

[23] "PerfTest - Performance Tests suite that bundle with OFED," Sep. 2009.

[24] "Sun Datacenter InfiniBand Switch 648," Oracle Corporation, http://www.oracle.com/us/products/servers-storage/networking/infiniband/034537.htm.

[25] "Voltaire QDR InfiniBand Grid Director 4700," Voltaire Inc., http://www.voltaire.com/Products/InfiniBand/Grid_Director_Switches/Voltaire_Grid_Director_4700.

[26] "IS5600 - 648-port InfiniBand Chassis Switch," Mellanox Technologies, http://www.mellanox.com/related-docs/prod_ib_switch_systems/IS5600.pdf.

[27] "HPC Challenge Benchmark," http://icl.cs.utk.edu/hpcc/.

[28] W. J. Dally and B. Towles, *Principles and practices of interconnection networks*. Morgan Kaufmann, 2004, ch. 15.4.1, pp. 294–295.

Paper III

sFtree: A Fully Connected and Deadlock-Free Switch-to-Switch Routing Algorithm for Fat-Trees

Bartosz Bogdański, Sven-Arne Reinemo, Frank Olaf Sem-Jacobsen, Ernst Gunnar Gran

sFtree: A Fully Connected and Deadlock-Free Switch-to-Switch Routing Algorithm for Fat-Trees

BARTOSZ BOGDANSKI, SVEN-ARNE REINEMO, FRANK OLAF SEM-JACOBSEN, and ERNST GUNNAR GRAN, Simula Research Laboratory

Existing fat-tree routing algorithms fully exploit the path diversity of a fat-tree topology in the context of compute node traffic, but they lack support for deadlock-free and fully connected switch-to-switch communication. Such support is crucial for efficient system management, for example, in InfiniBand (IB) systems. With the general increase in system management capabilities found in modern InfiniBand switches, the lack of deadlock-free switch-to-switch communication is a problem for fat-tree-based IB installations because management traffic might cause routing deadlocks that bring the whole system down. This lack of deadlock-free communication affects all system management and diagnostic tools using LID routing.

In this paper, we propose the sFtree routing algorithm that guarantees deadlock-free and fully connected switch-to-switch communication in fat-trees while maintaining the properties of the current fat-tree algorithm. We prove that the algorithm is deadlock free and we implement it in OpenSM for evaluation. We evaluate the performance of the sFtree algorithm experimentally on a small cluster and we do a large-scale evaluation through simulations. The results confirm that the sFtree routing algorithm is deadlock-free and show that the impact of switch-to-switch management traffic on the end-node traffic is negligible.

Categories and Subject Descriptors: C.2.2 [**Computer Communications Network**]: Network Protocol—*Routing protocols*

General Terms: Algorithms

Additional Key Words and Phrases: Routing, fat-trees, interconnection networks, InfiniBand, switches, deadlock

ACM Reference Format:
Bogdanski, B., Reinemo, S.-A., Sem-Jacobsen, F. O., and Gran, E. G. 2012. sFtree: A fully connected and deadlock-free switch-to-switch routing algorithm for fat-trees. ACM Trans. Architec. Code Optim. 8, 4, Article 55 (January 2012), 20 pages.
DOI = 10.1145/2086696.2086734 http://doi.acm.org/10.1145/2086696.2086734

1. INTRODUCTION

The fat-tree topology is one of the most common topologies for high performance computing (HPC) clusters today, and for clusters based on InfiniBand (IB) technology, the fat-tree is the dominating topology. This includes large installations such as the Roadrunner, Ranger, and JuRoPa [Top 500 2011]. There are three properties that make fat-trees the topology of choice for high performance interconnects: (a) deadlock freedom, the use of a tree structure makes it possible to route fat-trees without using virtual lanes for deadlock avoidance; (b) inherent fault-tolerance, the existence of multiple paths between individual source destination pairs makes it easier to handle network faults; (c) full bisection bandwidth, the network can sustain full speed communication between the two halves of the network.

Author's address: B. Bogdanski, Simula Research Laboratory, Norway; email: bartoszb@simula.no.

DOI 10.1145/2086696.2086734 http://doi.acm.org/10.1145/2086696.2086734

For fat-trees, as with most other topologies, the routing algorithm is crucial for efficient use of the network resources. The popularity of fat-trees in the last decade led to many efforts trying to improve the routing performance. This includes the current approach that the OpenFabrics Enterprise Distribution (OFED) [OpenFabrics Alliance 2011], the de facto standard for IB system software, is based on. That approach is presented in works by Gómez et al. [2007], Lin et al. [2004] and Zahavi et al. [2009]. Additionally, there exist several performance optimizations to this approach [Rodriguez et al. 2009; Bogdanski et al. 2010; Guay et al. 2011].

All the previous work, however, has one severe limitation when it comes to *switch-to-switch* communication. None of them support deadlock-free and fully connected switch-to-switch communication which is a requirement for efficient system management where all switches in the fabric can communicate with every other switch in a deadlock free manner. This is crucial because current IB switches support advanced capabilities for fabric management that rely on IP over IB (IPoIB). IPoIB relies on deadlock-free and fully connected IB routing tables. Without such support, features like the Simple Network Management Protocol (SNMP) for management and monitoring, Secure SHell (SSH) for arbitrary switch access, or any other type of IP traffic or applications using LID routing between the switches, will not work properly. The routing tables will only be fully connected and deadlock free from the point-of-view of the leaf switches.

There are algorithms that manage to obtain full connectivity on fat-tree topologies, but using them means sacrificing either performance or deadlock freedom. First of all, there is minhop that simply does shortest-path routing between all the nodes. As the default fallback algorithm implemented in the OpenSM subnet manager, it is not optimized for fat-tree topologies and, furthermore, it is not deadlock free. The alternatives include using a different routing algorithm, like layered shortest-path routing (LASH) [Skeie et al. 2002] or Up*/Down* [Schroeder et al. 1991]. LASH uses virtual lanes (VL) for deadlock avoidance and ensures full connectivity between every pair of nodes, but, like minhop, it is not optimized for fat-trees, which leads to suboptimal performance and longer route calculation times. Up*/Down* does not use VLs, but otherwise has the same drawbacks as LASH. Finally, there is a deadlock-free single-source shortest-path (DFSSSP) routing algorithm based on Dijkstra's algorithm [Domke et al. 2011]. However, it assumes that switch traffic will not cause a deadlock and uses VLs for deadlock avoidance for end-node traffic only.

In this paper we present, to the best of our knowledge, the first fat-tree routing algorithm that supports deadlock-free and fully connected switch-to-switch routing. Our approach retains all the performance characteristics of the algorithm presented by Zahavi [2009], and it is evaluated on a working prototype tested on commercially available IB technology. Our sFtree algorithm fully supports all types of single and multi-core fat-trees commonly encountered in commercial systems.

The rest of this paper is organized as follows. We introduce the InfiniBand Architecture in Section 2, followed by a description of fat-tree topologies and routing in Section 3. A description of the sFtree algorithm is given in Section 4 and in Section 5 we prove that it is deadlock free. Then we describe the setup of our experiments in Section 6, followed by a performance analysis of the result from the experiments and simulations in Section 7. Finally, we conclude in Section 8.

2. THE INFINIBAND ARCHITECTURE

InfiniBand is a lossless serial point-to-point full-duplex interconnect network technology that was first standardized in October 2000 [IBTA 2007]. The current trend is that IB is replacing proprietary or low-performance solutions in the high-performance computing domain [Top 500 2011], where high bandwidth and low latency are the key requirements.

The de facto system software for IB is OFED developed by dedicated professionals and maintained by the OpenFabrics Alliance [OpenFabrics Alliance 2011]. The sFtree algorithm that we propose in this paper was implemented and evaluated in a development version of OpenSM, which is the subnet manager distributed together with OFED.

2.1. Subnet Management

InfiniBand networks are referred to as *subnets*, where a subnet consists of a set of hosts interconnected using switches and point-to-point links. An IB fabric constitutes one or more subnets, which can be interconnected using routers. Hosts and switches within a subnet are addressed using local identifiers (LIDs) and a single subnet is limited to 49151 LIDs.

An IB subnet requires at least one subnet manager (SM), which is responsible for initializing and bringing up the network, including the configuration of all the IB ports residing on switches, routers, and host channel adapters (HCAs) in the subnet. At the time of initialization the SM starts in the *discovering state* where it does a sweep of the network in order to discover all switches and hosts. During this phase it will also discover any other SMs present and negotiate who should be the master SM. When this phase is complete the SM enters the *master state*. In this state, it proceeds with LID assignment, switch configuration, routing table calculations and deployment, and port configuration. When this is done, the subnet is up and ready for use. After the subnet has been configured, the SM is responsible for monitoring the network for changes.

A major part of the SM's responsibility is to calculate routing tables that maintain full connectivity, deadlock freedom, and proper load balancing between all source and destination pairs. Routing tables must be calculated at network initialization time and this process must be repeated whenever the topology changes in order to update the routing tables and ensure optimal performance.

During normal operation the SM performs periodic *light sweeps* of the network to check for topology changes (e.g., a link goes down, a device is added, or a link is removed). If a change is discovered during a light sweep or if a message (trap) signaling a network change is received by the SM, it will reconfigure the network according to the changes discovered. This reconfiguration also includes the steps used during initialization.

IB is a lossless networking technology where flow-control is performed per *virtual lane* (VL) [Dally 1992]. VLs are logical channels on the same physical link, but with separate buffering, flow-control, and congestion management resources. The concept of VLs makes it possible to build virtual networks on top of a physical topology. These virtual networks, or layers, can be used for various purposes such as efficient routing, deadlock avoidance, fault-tolerance and service differentiation.

Our contribution in this paper will not make use of the service differentiation features in IB, but we will use VLs to show how switch-to-switch traffic influences end-node traffic when both traffic types are sharing one VL and when they are separated into different VLs. For more details about service differentiation mechanisms refer to IBTA [2007] and Reinemo et al. [2006].

3. FAT-TREE ROUTING

The fat-tree topology was introduced by Leiserson [1985] and has since become a common topology in HPC. The fat-tree is a layered network topology with equal link capacity at every tier (applies for balanced fat-trees) and is commonly implemented by building a tree with multiple roots, often following the m-port n-tree definition [Lin et al. 2004] or the k-ary n-tree definition [Petrini and Vanneschi 1995]. An XGFT notation is also used to describe fat-trees and was presented by Öhring [1995].

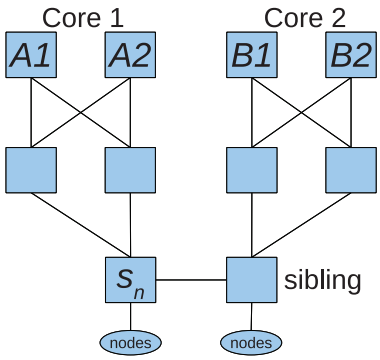


Fig. 1. A multi-core tree showing the s_n switch and its neighbor (sibling).

To construct larger topologies, the industry has found it to be more convenient to connect several fat-trees together rather than building a single large fat-tree. Such a fat-tree built from several single fat-trees is called a multi-core fat-tree. An example illustrating the concept is presented in Figure 1. Multi-core fat-trees may be interconnected through the leaf switches using horizontal links [Jülich Supercomputing Centre 2011] or by using an additional layer of switches at the bottom of the fat-tree and every such switch is connected to all the fat-trees composing the multi-core fat-tree [TACC 2011].

For fat-trees, as for most other network topologies, the routing algorithm is essential in order to exploit the available network resources. In fat-trees, the routing consists of two distinct phases: the upward phase in which the packet is forwarded from the source in the direction of one of the root switches and the downward phase when the packet is forwarded downwards to the destination. The transition between these two phases occurs at the lowest common ancestor, which is a switch that can reach both the source and the destination through its downward ports. Such an implementation ensures deadlock freedom, and the implementation presented in Zahavi et al. [2009] also ensures that every path towards the same destination converges at the same root (top) switch such that all packets toward that destination follow a single dedicated path in the downward direction. By having a dedicated downward path for every destination, contention in the downward phase is effectively removed (moved to the upward stage) so that packets for different destinations have to contend for output ports in only half of the switches on their path. In oversubscribed fat-trees, the downward path is not dedicated and is shared by several destinations.

3.1. Switch-to-Switch Routing

The fat-tree routing described in the previous section does not include switch-to-switch communication. In fact, the switch-to-switch communication has been ignored for a long time because the switches themselves lacked the necessary intelligence to be able to support advanced system management techniques, and originated very little traffic. In other words, IB switches were considered to be transparent devices that could be safely ignored from the point of view of the routing algorithm. Even today, switch-to-switch paths are treated as secondary paths (not balanced across ports), and are routed in the same manner as compute node paths. It means that to find a path between any two switches, the lowest common ancestor must be found, and the traffic is always forwarded through the first available port. Moreover, such a routing

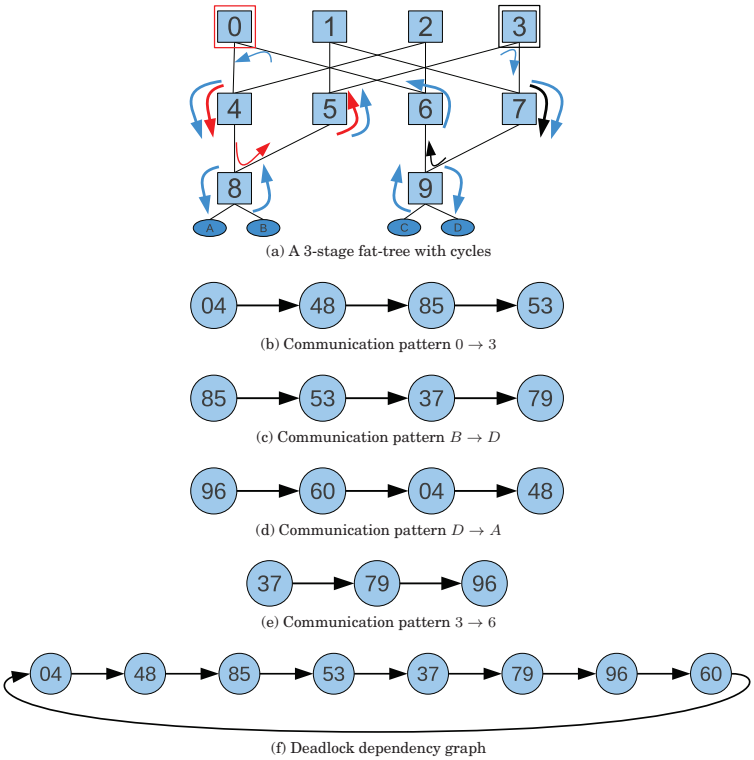
scheme does not provide connectivity between those switches which do not have a lowest common ancestor. These are usually the root switches in any fat-tree topology, but the problem can also manifest itself for all non-leaf switches in multi-core fat-trees or in an ordinary fat-tree with a rank (the number of stages in a tree) greater than two depending on the cabling as discussed in Section 4.1.

Today, full connectivity between all the nodes—both the end-nodes and the switches—is a requirement. IB diagnostic tools rely on LID routing and need full connectivity for basic fabric management and monitoring. More advanced tools like *perftest* used for benchmarking employ IPoIB. Moreover, IPoIB is also required by non-InfiniBand-aware management and monitoring protocols and applications like SNMP or SSH. Being an encapsulation method that allows running TCP/IP traffic over an IB network, IPoIB relies on the underlying LID routing. In the past, there was no requirement for connectivity between the switches because they lacked the capability to run many of the above mentioned tools and protocols. Today, however, switches are able to generate arbitrary traffic, can be accessed like any other end-node, and often contain an embedded SM. Therefore, the requirement for full connectivity between all the switches in a fabric is essential.

From the system-wide perspective of an interconnection network, deadlock freedom is a crucial requirement. Deadlocks occur because network resources such as buffers or channels are shared and because IB is a lossless network technology, i.e., packet drops are usually not allowed. The necessary condition for a deadlock to happen is the creation of a cyclic credit dependency. This does not mean that when a cyclic credit dependency is present, there will always be a deadlock, but it makes the deadlock occurrence possible. In this paper we demonstrate that when a deadlock occurs in an interconnection network, it prevents a part of the network from communicating at all. Therefore, the solution that provides full connectivity between all the nodes has to be deadlock free.

An example of a deadlock occurring in a fat-tree is presented on Figure 2(a). This is a simple 3-stage fat-tree topology that was routed without any consideration for deadlock freedom. We show that only four communication pairs are required to create a deadlock. The first pair, $0 \rightarrow 3$, is marked with red arrows, and the second pair, $3 \rightarrow 6$, is marked with black arrows. These two pairs are the switch-to-switch communication patterns. Two node-to-node pairs are also present and marked with blue arrows: $B \rightarrow D$ and $D \rightarrow A$. The deadlock that occurs with these four pairs is further illustrated using channel dependency subgraphs (see Definition 5.5 and Definition 5.6) that are shown on Figure 2(b) through Figure 2(e). A channel dependency graph is constructed by representing links in the network topology by vertices in the graph. Two vertices are connected by an edge if a packet in the network can hold one link while requesting the next one. On the figures, the number in each circle is the link between the two devices, for example, 04 is the link between the switch marked as 0 and the switch marked as 4 in Figure 2(a). Looking at these four channel dependency graphs, we are able to see that they in fact contain a cycle, which is shown on Figure 2(f). This example also shows that any 3-stage fat-tree can deadlock with node-to-node and switch-to-switch traffic present, if minhop is run on it. This is because minhop is unable to route a ring of 5 nodes or bigger in a deadlock-free manner [Guay et al. 2010], and almost any (apart from a single-root tree) 3-stage fat-tree will contain a 6-node ring. By having shown this deadlock example, we have demonstrated the need for an algorithm that will provide deadlock-free routing in fat-tree topologies when switch-to-switch communication between all switches is present.

Currently, the only deadlock-free routing algorithm for IB that complies with full connectivity requirement is LASH. It uses VLs to break the credit cycles in channel dependency graphs and provides full connectivity between all the nodes in the fabric.



(a) A 3-stage fat-tree with cycles

(b) Communication pattern $0 \rightarrow 3$

(c) Communication pattern $B \rightarrow D$

(d) Communication pattern $D \rightarrow A$

(e) Communication pattern $3 \rightarrow 6$

(f) Deadlock dependency graph

Fig. 2. An example of a deadlock occurring in a fat-tree.

However, for fat-tree topologies, LASH is a suboptimal choice because it does not exploit the properties of the fat-trees when assigning the paths, and thus gives worse network performance than ordinary fat-tree routing as shown in the performance evaluation section and also shown by Domke et al. [2011]. Additionally, route calculation with LASH is time consuming and due to the shortest-path requirement, it unnecessarily uses VL resources to provide deadlock freedom for a fat-tree topology. Another routing protocol that could theoretically support all-to-all switch-to-switch communication in a deadlock-free manner in fat-trees is DFSSSP. However, it has the same limitations as LASH when it comes to performance and route calculation time, and it does not break credit loops when they occur between non-HCA nodes, which means that switch-to-switch communication is not deadlock free.

Deadlocks can be avoided by using VLs that segment the available physical resources as LASH or DFSSSP do. We show, however, that for fat-trees, using VLs for deadlock avoidance is inefficient because the whole fabric can be routed in a deadlock-free manner by only using a single VL. The additional VLs can be used for other purposes like QoS.

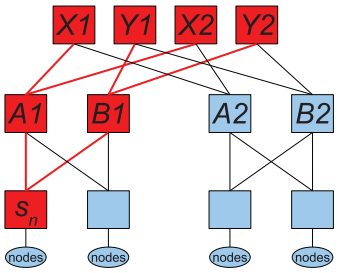


Fig. 3. A 3-stage fat-tree with a sample subtree converging to s_n.

4. THE SFTREE ALGORITHM

Our proposed design is based on the fat-tree routing algorithm presented by Zahavi [2009] and can be treated as a modular extension, which for every *switch source* takes all *switch destinations* and, if such a destination is marked as unreachable by the default fat-tree routing tables (meaning that it has no $up \rightarrow down$ path), it finds a deadlock-free path.

4.1. Design of the Algorithm

The current fat-tree routing algorithm proposed by Zahavi et al. [2009] is already capable of assigning switch-to-switch paths, but only those that follow the up/down turn model on which the fat-tree routing is based. For example, in Figure 3, the switches *A1* and *A2* would have connectivity by going up through switch *X1* or *X2*. However, switches *A1* and *B1* do not have a lowest common ancestor, so one possibility for them to achieve connectivity is to go through one of the common descendants (shortest-path method). The alternative method (suboptimal path or non-shortest-path) for them would be to go through any leaf switch in the topology, not necessarily located directly below them. This would require first going up, then down, until reaching a switch that has a path to both of the communicating switches, and then up again and down again until the destination. In this paper, we use the second approach, because it can be implemented without the need for additional VLs. The shortest-path method is only used for a specific set of switches (an example of such a set is marked in red on Figure 3). To guarantee deadlock freedom, the key to our approach is that all the switches in the topology collectively choose the same leaf switch for communication.

To achieve deadlock-free full connectivity between the switches, we propose using an upsidedown subtree whose root is one of the leaf switches (formally defined in Definition 5.7) and illustrated in Figure 3. The subtree concept allows us to localize all the down/up turns in a fabric to a single, deadlock-free tree. As we demonstrate in Section 5, accommodating all the prohibited turns in a subtree is deadlock free, and it permits full connectivity if a subtree root (s_n) is chosen properly. To illustrate the routing through a subtree concept, we can observe that the switches *X1* and *Y1* will communicate through a subtree root s_n whereas *X1* and *X2* will go through switch *A1*. Combining the description of the subtree with the discussion in the previous paragraphs, we can observe that switches external to the subtree must communicate through it to reach other switches previously marked as unreachable, which leads to usage of suboptimal (non-shortest) paths. In other words, any switch in the fabric that cannot reach any other switch using the traditional upward to downward scheme, must first send the packets to the subtree and the packets are passed down in the subtree until they reach a switch that

has a path to the destination switch. In this switch the packets make a U-turn, i.e., a down-to-up turn and follow a traditional up-to-down path towards the destination. The deadlock freedom of this approach relies on the fact that all the U-turns take place only within the subtree and that any up-to-down turn will leave the subtree. Traffic leaving the subtree will not be able to circulate back into it because, by design, there are no U-turns outside of the subtree. This will be further explained in the proof in Section 5. It is also important to note that this communication scheme does not change the existing upward to downward paths between the switches that can reach each other using the traditional fat-tree routing.

ALGORITHM 1: Select a subtree root function

Require: Routing table has been generated.
Ensure: A subtree root (sw_{root}) is selected.

```
1:  found = false
2:  for sw_leaf = 0 to max_leaf_sw do
3:    if found == true then
4:      break
5:    end if
6:    sw_root = sw_leaf
7:    found = true
8:    for dst = 1 to max_dst_addr do
9:      if sw_root.routing_table[dst] == no_path then
10:       found = false
11:       break
12:     end if
13:   end for
14: end for
15: if found = false then
16:   sw_root = get_leaf(0)
17: end if
```

In detail, the algorithm first finds a subtree by finding a subtree root. The pseudocode for this step is presented in Algorithm 1. The algorithm traverses all the leaf switches in the topology and, for each leaf switch, checks whether any of the switch destination addresses are marked as unreachable in its routing table. If all the switch destinations in the routing table are marked as reachable, the first encountered leaf switch is selected as a subtree root switch. There may be as many potential subtrees in a fat-tree as there are leaf switches having full connectivity, however, only one of those leaf switches will be selected as the subtree root and there will be only one subtree with a root in this particular leaf switch. Algorithm 2 is only called for those switch pairs that do not have a path established using the traditional fat-tree routing.

When the switch-to-switch routing function is called, the first step is to determine whether the routing table of the *subtree root* (sw_{root}) contains a path to the *destination switch* (sw_{dst}) as shown in Algorithm 2 line 1. If it does, then the path to the *destination switch* is inserted into the routing table of the *source switch* (sw_{src}), and the output port for the *destination switch* is the same as the output port for the path to the *subtree root* (lines 2-5). Because the switch-to-switch routing function is called for every switch, in the end, all the paths to each unreachable *destination switch* will converge to the subtree. In other words, the selected *subtree root* is the new target for all the unreachable *destination switches*. The down/up turn will take place at the first switch located in the subtree that has an upward path both to the source switch and the destination switch.

In a single-core fat-tree there will always be a path from the *source switch* to the *subtree root*, and the *subtree root* will have a path to every *destination switch*. However,

ALGORITHM 2: Switch-to-switch routing function

Require: Subtree root (sw_{root})
Ensure: Each sw_{src} reaches each sw_{dst} at worst by sw_{root}.

```
 1: if found or sw_root.routing_table[dst] ≠ no_path then
 2:    sw_src.routing_table[dst] = sw_src.routing_table[sw_root.addr]
 3:    get_path_length(sw_src, null, dst, sw_root, hops)
 4:    set_hops(sw_src, hops)
 5:    return true
 6: else if (sw_sib = get_sibling_sw(sw_root)) ≠ null then
 7:    if sw_src.routing_table[sw_sib.addr] ≠ no_path then
 8:       if sw_sib.routing_table[dst] ≠ no_path then
 9:          sw_root.routing_table[dst] = get_port_to_sibling(sw_sib)
10:          hops = get_hops(sw_sib, dst)
11:          set_hops(sw_root, hops + 1)
12:          hops = 0
13:          sw_src.routing_table[dst] = sw_src.routing_table[sw_sib.addr]
14:          get_path_length(sw_src, null, dst, sw_root, hops)
15:          set_hops(sw_src, hops)
16:          return true
17:       end if
18:    end if
19: end if
20: print 'SW2SW failed for sw_src and sw_dst.'
21: return false
```

for complex multi-core or irregular fat-trees this may not always be the case, and the second part of the pseudocode presented in Algorithm 2 deals with cases where the best-effort approach is needed and the *subtree root* does not have paths to all the destinations. For best effort, it is necessary to check whether the subtree root has a direct neighbor (to which they are connected through horizontal links as shown in Figure 1). Next, a check is done to verify whether that neighbor, also called a sibling, has a path to the *destination switch*. This is done in lines 7 and 8 of the presented pseudocode. If the sibling exists and it has a path to the *destination switch*, a path is set both on the *subtree root* and the originating *source switch* to the *destination switch* through the sibling switch (lines 9-16). In other words, the target for the unreachable *destination switch* will still be the *subtree root*, but in this case, it will forward the packets to the *destination switch* to its sibling which in turn will forward them to the *destination switch*.

If both previous steps do not return from the function with a true value, the algorithm has failed to find a path between two switches. This occurs only for such topologies on which it is not recommended to run the fat-tree routing at all due to multiple link failures or very irregular connections between the nodes. An example of such a topology would be two fat-trees of different sizes connected to each other in an asymmetrical manner, for example, from a few middle-stage switches on the larger tree to roots on the smaller one. Using various command-line parameters, fat-tree routing in OpenSM can be forced to run on such a topology, however, it will give suboptimal routing not only when it comes to performance, but also when we consider connectivity.

4.2. OpenSM Implementation

OpenSM requires that every path be marked with a hop count to the destination. The older versions of the fat-tree routing algorithm (pre-3.2.5, current ones are 3.3.x) calculated the number of hops using the switch rankings in the tree. In the newer

versions, the counting is done using a simple counter in the main routing function. Because the switch-to-switch routing we perform is totally independent of the main routing done by the fat-tree algorithm, we could not use the counters stored there, and because of the possible zig-zag paths (i.e., paths not following the shortest hop path), counting using the switch ranks is unreliable. Therefore, we devised a simple recurrence function for hop calculation, called *get_path_length* only to be used during the switch-to-switch routing (it is called in line 14 of the pseudocode shown in Algorithm 2). This function iterates over the series of the switches that constitute the path, and when it reaches the one having a proper hop count towards the destination, by backtracking it writes the correct hop count into the routing tables of the switches on the whole path.

Moreover, one of the enhancements that we made is to choose the subtree root in such a way that the subnet manager node (the end-node on which the subnet manager is running) is not connected to the switch marked as the subtree root. This will draw the switch-to-switch traffic away from the subnet manager, thus, not creating a bottleneck in a critical location.

5. DEADLOCK FREEDOM PROOF

In this section we prove that the sFtree algorithm is deadlock free. The first part of the proof contains the definitions of the terms used later in the text.

Definition 5.1. By switch s_n we mean a switch with a rank n. Root switches have rank $n = 0$.

Definition 5.2. By a *downward channel* we mean a link between s_{n-1} and s_n where the traffic is flowing from s_{n-1} to s_n. By an *upward channel* we mean a link between s_n and s_{n-1} where the traffic is flowing from s_n to s_{n-1}.

Definition 5.3. A *path* is defined as a series of switches connected by channels. It begins with a source switch and ends with a destination switch.

Definition 5.4. A *U-turn* is a turn where a downward channel is followed by an upward channel.

Definition 5.5. A *channel dependency* between channel c_i and channel c_j occurs when a packet holding channel c_i requests the use of channel c_j.

Definition 5.6. A *channel dependency graph* $G = (V, E)$ is a directed graph where the vertices V are the channels of the network N, and the edges E are the pairs of channels (c_i, c_j) such that there exists a (channel) dependency from c_i to c_j.

Definition 5.7. By a *subtree* we mean a logical upside down tree structure within a fat-tree that converges to a single leaf switch s_n, which is the single root of the subtree. A subtree expands from its root and its leaves are all the top-level switches in the fat-tree. Any upward to downward turn will leave the subtree structure.

The following theorem states the sufficient condition for deadlock freedom of a routing function [Duato et al. 2003].

THEOREM 5.1. *A deterministic routing function R for network N is deadlock free if and only if there are no cycles in the channel dependency graph G.*

Next, we prove the necessary lemmas followed by the deadlock freedom theorem of the sFtree algorithm.

LEMMA 5.8. *There exists at least one subtree within a fat-tree that allows full switch-to-switch connectivity using only U-turns within that subtree.*

PROOF. In a fat-tree, from any leaf we can reach any other node (including all the switches) in the fabric using an up/down path or a horizontal sibling path. This is because every end-node can communicate with every other end-node. Because end-nodes are directly connected to the leaf switches, it follows that leaf switches are able to reach any other node in the topology. Consequently, if it contains no link faults, any leaf switch can act as the root of the subtree. □

LEMMA 5.9. *There can be an unlimited number of U-turns within a single subtree without introducing a deadlock.*

PROOF. A subtree is a logical tree structure and two types of turns can take place within it: U-turns, i.e., downward-to-upward turns, and ordinary upward-to-downward turns. The U-turns cannot take place at the top switches in a subtree (and fat-tree) because these switches have no output upward ports. Any upward to downward turn will leave the subtree according to Definition 5.7.

Because a subtree is a connected graph without any cycles, it is deadlock free by itself. Any deadlock must involve U-turns external to the subtree since a U-turn is followed by an upward to downward turn leaving the subtree, which is also shown on Figure 2(a). All the traffic going through an up/down turn originating in the subtree will leave the subtree when the turn is made and follow only the downward channels to the destination. Such a dependency can never enter the subtree again and can never reach any other U-turn, and so, cannot form any cycle. □

THEOREM 5.2. *The sFtree algorithm using the subtree method is deadlock free.*

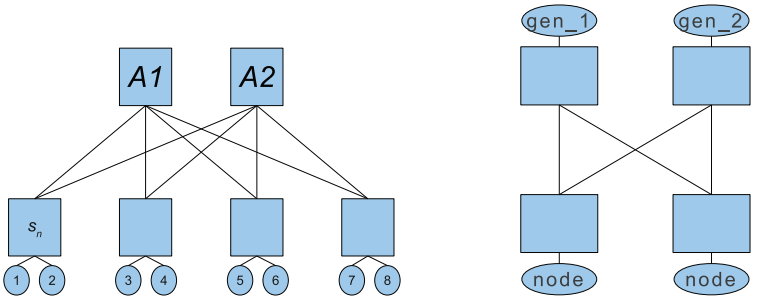
PROOF. Following Lemma 5.8 and Lemma 5.9, we observe that a deadlock in a fat-tree can only occur if there are U-turns outside the subtree. Such a situation cannot happen due to the design of the sFtree algorithm where all U-turns take place within a subtree. The traffic leaving a subtree will not circle back to it because once it leaves the subtree, it is forwarded to the destination through downward channels only, and it is consumed, thus, no cyclic credit dependencies will occur in the topology. □

6. EXPERIMENT SETUP

To evaluate our proposal, we have used a combination of simulations and measurements on an IB cluster. In the following sections, we present the hardware and software configurations used in our experiments.

6.1. Experimental Test Bed

Our test bed consisted of eight nodes and six switches. Each node is a Sun Fire X2200 M2 server [Oracle Corporation 2006] that has a dual port Mellanox ConnectX DDR HCA with an 8x PCIe 1.1 interface, one dual core AMD Opteron 2210 CPU, and 2GB of RAM. The switches were: two 36-port Sun Datacenter InfiniBand Switch 36 [Oracle Corporation 2011a] QDR switches which acted as the fat-tree roots; two 36-port Mellanox Infiniscale-IV QDR switches [Mellanox Technologies 2009], and two 24-port SilverStorm 9024 DDR switches [Qlogic 2007], all of which acted as leaves with the nodes connected to them. The port speed between the QDR switches was configured to be 4x DDR, so the requirement for the constant bisectional bandwidth in the fat-tree was assured. The cluster was running the Rocks Cluster Distribution 5.3 with kernel version 2.6.18-164.6.1.el5-x86_64, and the IB subnet was managed using a modified version of OpenSM 3.2.5 with our sFtree implementation. The topology on which we performed the measurements is shown in Figure 4(a). Switches *A1* and *A2* are the Sun devices, which are able to produce and consume traffic. These two switches were used as follows. We established a connection between the switches, and used the sFtree



(a) The experimental cluster. (b) Deadlock scenario topology.

Fig. 4. The clusters used during the experiments.

routing algorithm to forward traffic in a deadlock-free manner between the devices through the chosen subtree root s_n.

The Sun switches are the only managed switches in the topology, and they are able to generate and consume arbitrary traffic. The firmware installed on the switches is 1.1.3-2, and the BIOS revision is NOW1R112. Those switches have an enhanced port 0 which is connected to the IB switching fabric using a 1x SDR link (signaling rate is 2.5 Gb/s, and the effective speed is 2 Gb/s). That port 0 is also connected to the internal host using a 1x PCI Express 1.1. The IPoIB interfaces on both switches were configured with an MTU size of 2044 octets, which is the maximum supported size in the IPoIB Datagram Mode (IPoIB-UD) [Chu and Kashyap 2006] (in most implementations). The MTU size on the switches cannot be increased because there is no support for the optional IPoIB Connected Mode (IPoIB-CM) [Kashyap 2006]. The link MTU provides a limit to the size of the payload that may be used. The 2044 octet MTU is in fact a 2048 octet MTU minus the 4-octet encapsulation overhead, which is done to reduce problems with fragmentation and path-MTU discovery.

Furthermore, we constructed a separate topology to show the effects of a deadlock in a small fat-tree when a suboptimal routing engine is chosen. The topology is shown in Figure 4(b). That fat-tree can be unfolded into a ring which, when routed improperly, will deadlock. Because the 1x SDR generators built into the Sun Datacenter InfiniBand Switches injected packets too slow to create a deadlock, we created an artificial scenario in which we connected ordinary end-nodes, namely, *gen_1* and *gen_2* to the root switches, and routed the topology with sFtree and minhop algorithms. We used the same settings for the hardware as in the previous scenario.

6.2. Simulation Test Bed

To perform large-scale evaluations and verify the scalability of our proposal, we use an InfiniBand model for the OMNeT++ simulator [Gran and Reinemo 2011]. The model contains an implementation of HCAs and switches with support for routing tables and virtual lanes. The network topology and the routing tables were generated using OpenSM and converted into OMNeT++ readable format in order to simulate real-world systems. The simulations were performed on a 648-port fat-tree topology, illustrated in Figure 5. This topology is the largest 2-stage fat-tree topology that can be constructed using 36-port switch elements. When fully populated, this topology consists of 18 root switches and 36 leaf switches. We chose the 648-port fabric because it is a common configuration used by switch vendors in their own 648-port systems [Oracle

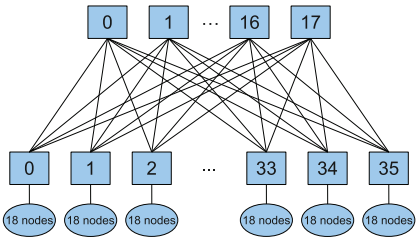


Fig. 5. A 648-port fat-tree topology.

Corporation 2011b; Voltaire 2011; Mellanox Technologies 2011]. Additionally, such switches are often connected together to form larger installations like the JuRoPa supercomputer [Jülich Supercomputing Centre 2011].

For the simulations we used a few different traffic patterns, but because of space limitations we are only presenting the results for the uniform traffic. All other simulations using nonuniform traffic exhibited the same trends. The uniform traffic pattern had a random destination distribution and the end-nodes were sending only to other end-nodes and switches only to other switches. Each simulation run was repeated eight times with different seeds and the average of all simulation runs was taken. The message size was 2 kB for every simulation, and the packet generators and sinks at the switches were configured to match those from the hardware test. These simulations were performed to verify whether the switch traffic influences the end-node traffic, and if so, to what degree for large-scale scenarios. We also performed simulations for the different network topologies listed in Table IV, and the results show the same trends as the results for the 648-port switch.

7. PERFORMANCE EVALUATION

Our performance evaluation consists of measurements on the experimental fat-tree cluster and simulations of large-scale topologies. Before running the performance evaluation, we confirmed that the sFtree algorithm works as expected by using the command-line tool *ibtracert* and, after setting up IPoIB, *ping* and *route*. For measurements on the cluster, we use the results from the HPCC benchmark [HPCC 2011] run under MVAPICH2-1.4.1 [MVAPICH2 2011] to show how the sFtree algorithm impacts application traffic. In the HPCC benchmark the number of ring patterns was increased from 30 to 50000 to provide a better average when comparing the results. For the deadlock scenario, we used *perftest* to initiate communication between the nodes. For the simulations, we use the achieved average throughput per end node or per switch as the metric for measuring the impact of switch traffic on the simulated 648-port network topology.

7.1. Experimental Results

In the HPCC experiments we were simultaneously running the HPCC benchmark on the compute nodes and generating the switch-to-switch traffic that consisted of sequential reads and writes of random blocks of data from the internal hard drive. The data was sent using the TCP connection between the IB interfaces on the switches. The average data throughput between the switches was 25 MB/s both ways. There are several reasons why the switches are not able to send with the full capacity of their SDR link. First, the only available transport service is the IPoIB-UD, which

Table I. Results from the HPC Challenge Benchmark with All Communication in VL0

Network latency and throughput	a) SW2SW off	b) SW2SW on	c) Difference
Min Ping Pong Lat. (ms)	0.001997	0.001997	0.0%
Avg Ping Pong Lat. (ms)	0.002267	0.002266	−0.044%
Max Ping Pong Lat. (ms)	0.002369	0.002369	0.0%
Naturally Ordered Ring Lat. (ms)	0.002193	0.002193	0.0%
Randomly Ordered Ring Lat. (ms)	0.002157	0.002165	0.371%
Min Ping Pong BW (MB/s)	1587.248	1586.048	−0.076%
Avg Ping Pong BW (MB/s)	1588.806	1588.379	−0.027%
Max Ping Pong BW (MB/s)	1590.559	1591.162	0.038%
Naturally Ordered Ring BW (MB/s)	1488.926	1495.895	0.468%
Randomly Ordered Ring BW (MB/s)	1226.432	1226.347	0.007%

Table II. Results from the HPC Challenge Benchmark with Node Communication in VL1 and Switch Communication in VL0

Network latency and throughput	a) SW2SW off	b) SW2SW on	c) Difference
Min Ping Pong Lat. (ms)	0.001997	0.001997	0.0%
Avg Ping Pong Lat. (ms)	0.002300	0.002291	−0.391%
Max Ping Pong Lat. (ms)	0.002429	0.002429	0.0%
Naturally Ordered Ring Lat. (ms)	0.002289	0.002313	1.048%
Randomly Ordered Ring Lat. (ms)	0.002197	0.002214	0.773%
Min Ping Pong BW (MB/s)	1586.048	1586.048	0.0%
Avg Ping Pong BW (MB/s)	1588.650	1588.486	−0.01%
Max Ping Pong BW (MB/s)	1591.766	1591.011	−0.047%
Naturally Ordered Ring BW (MB/s)	1505.4256	1487.8035	−1.171%
Randomly Ordered Ring BW (MB/s)	1228.2845	1228.3750	0.007%

has a limited MTU size. No direct application access to the IB interface is possible because there is no user space access for establishing a Queue Pair (QP), so all the user data has to be encapsulated within IP and then forwarded through IB. Second, the packet-processing seems to be limited by the CPU whose utilization was constantly above 90%. In addition, there is no support for RDMA Read and Write requests, which further increases the CPU usage.

During the tests, we used a few different network performance tools (qperf, netperf, NetPIPE). These tools also support UDP traffic benchmarking, but after saturating the internal link with UDP traffic, the responsiveness of the switches was severely limited due to the high CPU utilization. Furthermore, we were not able to see any difference in the experimental results, and for that reason, we decided to use TCP traffic.

The experiment was repeated twice: In the first scenario, we mapped both the switch and the application traffic (HPCC) to VL0 and, in the second scenario, the application traffic was running on VL1 and switch traffic on VL0. To establish a base reference, for each scenario, we also disabled the switch traffic and measured the application traffic only. The reference results are presented in the second column of Tables I and II.

7.1.1. First scenario. Table I shows the results for the first scenario. The most interesting observation is that the influence of the switch traffic on the application traffic is negligible. There are variations between the reference results and the scenario results, but they are very small as seen in the last column of Table I. The influence of the switch traffic on the end-node traffic is negligible because there are more nodes than switches in the fabric and the nodes are able to inject traffic 8 times faster using all the links in the fabric. In comparison, the switches are unable to send at their full capacity because of the technical limitations described in the previous paragraphs and they only use a single path to communicate.

7.1.2. Second scenario. The results for the second scenario are presented in Table II. The key observation here, as seen in the last column of Table II is that for small data

streams, which emulate the management traffic between the switches, there is almost no influence of switch traffic on the end-node traffic when compared with the reference results presented in the second column of Table II. However, the latencies for both the natural and random rings are higher when switch-to-switch traffic is present. The difference is only 1%, but this pattern also corresponds well with the simulation results where using an additional VL for traffic separation decreases the overall throughput (see Section 7.2.2).

To conclude, the results from both scenarios clearly illustrate that there is little or no performance overhead when using the sFtree algorithm. However, we can suspect that if the volume of the traffic generated by the switches was larger, we would observe a drop in performance when assigning different VLs to different types of traffic (as in the second scenario). Because our solution is deadlock free, there is no need to use additional VLs for deadlock-avoidance. Still, one of the limitations of the hardware tests is the fact that only two switches out of six were able to send and receive traffic. The other limitation was the topology size and the fact that a large part of the node-to-node communication was taking part in the crossbar switches. Therefore, in Section 7.2 we will show through simulations that the same trends hold for a larger number of switches.

7.1.3. Deadlock scenario. We also created an artificial deadlock scenario. For the topology shown on Figure 4(b), we run the perftest to measure the sent bandwidth over time. As mentioned in Section 6, the built in generators in the currently available switches are not able to create a deadlock due to their low bandwidth. Therefore, we substituted those generators with fully capable end-node generators and configured the routing to emulate the paths assigned by the sFtree and minhop routing algorithms. However, the implementation of minhop available in OpenSM will not deadlock on such a topology, so by modifying the routing table on the fly we managed to obtain the desired effect. The reason why minhop will not deadlock on a ring constructed out of four nodes is the fact that its implementation in OpenSM assigns symmetric paths for *source-destination* and *destination-source* pairs, thus, it breaks the credit loop necessary for a deadlock to occur. However, it is a well known fact that minhop deadlocks on at least a 5-node ring [Guay et al. 2010], and any 3-stage fat-tree that has more than one root contains such a ring. Therefore, almost any 3-stage or larger fat-tree routed with minhop will potentially deadlock, thus, it makes our experiment valid and practical.

In this experiment all the communication streams were launched during the first five seconds. By analyzing Figure 6, we observe that minhop routing deadlocks immediately after the last stream that closes the credit loop is added, and then that the average per node bandwidth drops to 0 MB/s. For the sFtree routing algorithm, we observe that the average per node network bandwidth stabilizes after the last stream is added because all the U-turns take place on one leaf switch, and no credit loop is created.

This experiment shows the effect that a deadlock has on a fat-tree topology which is routed improperly. The sFtree routing algorithm, however, is safe to use and will not deadlock on a fat-tree topology as proven in Section 5.

7.2. Simulation Results

An important question is how well the sFtree algorithm scales. Specifically, the purpose of the simulations was to show that the same trends that were observed in the experiments exist when the number of switches generating the traffic will be much larger and the network topology corresponds to real systems. Like the hardware measurements, we performed two different tests. In the first scenario, we mapped the switch-to-switch and node-to-node traffic both to VL0, and, for the second scenario, we separated the node-to-node traffic by mapping it to VL1. Furthermore, for every measurement the

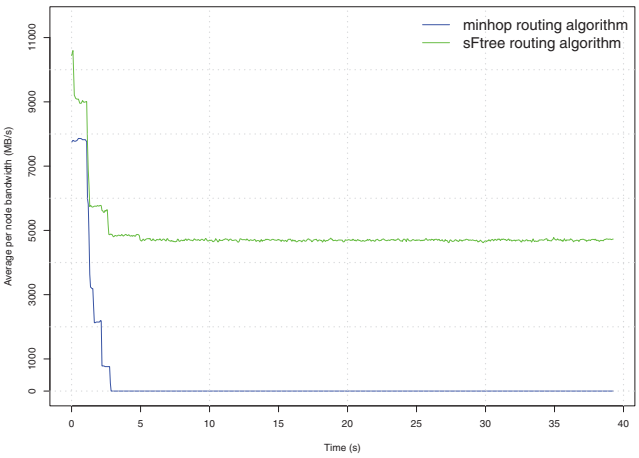
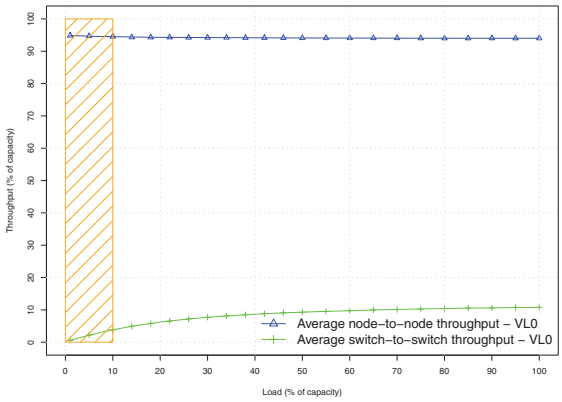


Fig. 6. Average per node network bandwidth comparison for minhop and sFtree.

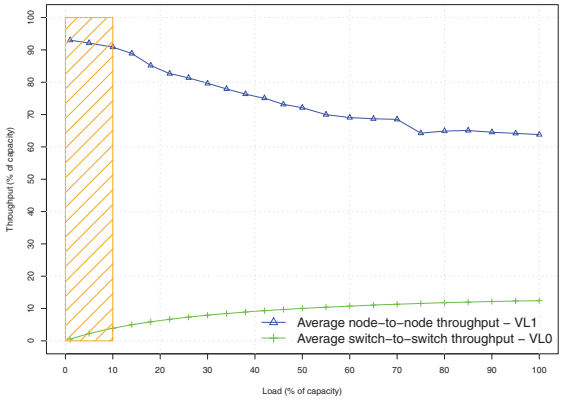
nodes were communicating at their full capacity, and the load of the switch traffic was gradually increased from 0% to 100%. It has to be noted that the scale on the Y-axis of the graphs presented in Figure 7 corresponds to the node-to-node sending and receiving capacity which is 4x DDR (20 Gb/s), and not to the 1x SDR (2.5 Gb/s) capacity of the switches. This means that all the results are normalized to 20 Gb/s and 12.5% of throughput achieved by the switches is in fact their full sending/receiving capability (1x SDR is 12.5% of 4x DDR). On all graphs, the throughput is presented as an average throughput per node. The second important detail is the shaded rectangles in Figure 7 that correspond to the hardware-obtainable results, i.e., the range of the results within those boxes can also be obtained with the available switches. The results beyond those boxes could not be obtained using the hardware due to limitations of the enhanced ports in real IB switches. With these two experiments we show the effect of using the same VL for both node-to-node traffic and switch-to-switch traffic and the effects of assigning these two types of traffic to different VLs.

7.2.1. First scenario. We observe that when both the node and switch traffic is mapped to the same VL, then the switch traffic does not influence the node traffic as shown in Figure 7(a), and the average achieved throughput per node decreases only by 1% when the switches send at their full capacity. There are two reasons why this happens. First, there are many more end-nodes in the network than there are switches (648 nodes and only 54 switches). The nodes are injecting more traffic into the fabric and the switch traffic suffers from contention (and possible congestion). Second, the switches have a limited number of paths they can choose from and the switch traffic is not balanced, always taking the first possible output port towards the destination, which leads to some congestion on switch-to-switch paths. Also, the subtree root is a potential bottleneck as a large part of the switch traffic is sent through it.

7.2.2. Second scenario. The situation changes when we separate the node traffic and map it to VL1 as shown in Figure 7(b). In this case, for higher loads of switch-to-switch traffic, the node traffic decreases to 64% of the capacity for the highest switch



(a) Simulation results for a 648-port fat-tree with end-node traffic in VL0.



(b) Simulation results for a 648-port fat-tree with end-node traffic in VL1.

Fig. 7. Simulation results for a 648-port switch built as fat-tree topology.

load. This happens because the traffic in the two VLs are given an equal share of the link bandwidth, artificially increasing the impact of the comparatively light switch-to-switch traffic. Clearly, care must be taken when separating management traffic in a separate VL to manage the VL arbitration weights for a minimal system impact.

7.2.3. Routing algorithm comparison. In Table III we compare various routing algorithms (sFtree, minhop, Up*/Down*, DFSSSP, LASH) at 100% load both for switch and node traffic on two different commercially available fabrics. The settings were the same as for other experiments and matched the hardware configuration. All the node-to-switch

Table III. Routing Algorithm Performance as Percentage of Throughput Per Node

Topology	sFtree	minhop	Up*/Down*	DFSSSP	LASH
648-port fat-tree switch (2-stage)	91.94%	66.98%	66.98%	85.49%	5.257%
648-port fat-tree with rack switches (3-stage)	92.93%	54.01%	54.01%	84.20%	0.8103%

and switch-to-switch links were 4x DDR and the switch generators were set to 1x SDR. We chose a 648-port fat-tree switch, which was also used for other simulations presented in this paper, and a 648-port fat-tree switch with an additional layer of rack switches attached at the bottom of it. In this case the rack switches are also interconnected horizontally to create two-switch units [Oracle Corporation 2010]. The results presented in Table III illustrate that only the sFtree algorithm is able to achieve high performance, and while DFSSSP may still be considered a viable choice, both minhop and Up*/Down* are not suitable for any of those two fat-tree topologies. Moreover, we observe that if the topology becomes more complex, the performance of all algorithms apart from sFtree deteriorates even more. It is worth noting that LASH is unsuitable for routing any type of fat-tree topology. The performance results are shown in Table III, being 5.257% and 0.8103% of average throughput per node for the 2-stage and 3-stage fat-trees, respectively. Such a low routing performance is explained by the fact that LASH does not balance the paths (like DFSSSP does) and selects the first possible shortest path towards the destination. Therefore, in the 648-port 2-stage fat-tree, out of 18 possible root switches, only one switch is used for forwarding the data packets from all 36 leaf switches and no traffic passes through the other 17 root switches. The same situations happens in the 3-stage fat-tree, but due to differences in cabling, the performance is even lower. In other words, when used for routing a fat-tree topology, LASH constructs a logical tree within the physical fat-tree fabric, which dramatically decreases the number of available links that could be used for forwarding the packets and leads to congestion. When it comes to VL usage, LASH only uses one VL for the 648-port fat-tree in Figure 5, but for a 3456-port fat-tree (3-stage fabric) LASH requires 6 VLs, and for the JuRoPa fat-tree it needs 8 VLs to ensure deadlock freedom. This means that using LASH not only leads to a decrease in routing performance, but also to a waste of valuable VL resources that could be used differently in fat-trees [Guay et al. 2011].

To summarize, the simulations confirmed what we observed in the hardware measurements. For the small loads (<10% of capacity) which are obtainable on the hardware, we see no influence of switch traffic on the end-node traffic. The simulations also confirmed that our solution is scalable and can be applied to larger topologies without negative impact on the end-node network performance. Apart from the 648-port topology presented in this paper, we simulated other topologies like a 3456-port switch (3456 nodes, 720 switches) and a JuRoPa-like supercomputer (5184 nodes, 864 switches) and the results obtained during these simulations exhibit the same trends as the results for the 648-port topology. Furthermore, for small loads, there is little difference when it comes to separating the node-to-node and switch-to-switch traffic. We also show that LASH despite having the desired properties of deadlock freedom and all-to-all communication, is not suitable for routing fat-tree topologies.

7.3. Execution Time

The execution time of the sFtree algorithm depends only on the number of switches in the network. For the largest single-core fat-tree shown in Table IV the execution overhead is around 0.5 seconds. For more complex topologies, like JuRoPa-like (864 switches) or Ranger-like (1440 switches) supercomputers, the overhead is 2.4 and 6.1 seconds respectively. For comparison, we also added the execution time results from LASH. Due to the fact that LASH does cycle search between all the possible pairs, routing takes much longer on all fabrics apart from the smallest ones. This is

Table IV. Execution Time of the OpenSM Routing in Seconds

Topology	no SW2SW	SW2SW	LASH
648-port fat-tree switch (s)	0.047	0.047	0.04
648-port fat-tree with rack switches (s)	0.104	0.112	0.27
3456-port fat-tree switch (s)	2.29	2.796	400.99
JuRoPA-like supercomputer (s)	6.43	8.858	–
Ranger-like supercomputer (s)	21.73	27.8	–

another reason why LASH should not be used for routing fat-trees because a possible rerouting takes a very long time. It is visible for a 3456-port fat-tree where LASH execution took almost 401 seconds. Unfortunately, we were not able to run LASH on topologies larger than the 3456-port fat-tree in a reasonable time.

8. CONCLUSIONS AND FUTURE WORK

In this paper we have proposed and demonstrated the sFtree routing algorithm which provides full connectivity and deadlock-free switch-to-switch routing for fat-trees. The algorithm enables full connectivity between any of the switches in a fat-tree when using IPoIB, which is crucial for fabric management features such as the Simple Network Management Protocol (SNMP) for management and monitoring, Secure SHell (SSH) for arbitrary switch access, or generic web interface access.

We have implemented the sFtree algorithm in OpenSM for evaluation on a small IB cluster and studied the scalability of our algorithm through simulations. Through the evaluation we verified the correctness of the algorithm and showed that the overhead of the switch-to-switch communication had a negligible impact on the end-node traffic. Moreover, we were able to demonstrate that the algorithm is scalable and works well even with the largest fat-trees. Furthermore, we compared the sFtree algorithm to several topology agnostic algorithms and showed that sFtree was by far the most optimal solution for switch-to-switch connectivity in fat-trees.

REFERENCES

Bogdanski, B., Sem-Jacobsen, F. O., Reinemo, S.-A., Skeie, T., Holen, L., and Huse, L. P. 2010. Achieving predictable high performance in imbalanced fat trees. In *Proceedings of the 16th IEEE International Conference on Parallel and Distributed Systems*. X. Huang, Ed. IEEE Computer Society, 381–388.

Chu, J. and Kashyap, V. 2006. RFC 1633 transmission of IP over InfiniBand (IPoIB). IETF.

Dally, W. J. 1992. Virtual-channel flow control. *IEEE Trans. Parall. Distrib. Syst. 3,* 2, 194–205.

Domke, J., Hoefler, T., and Nagel, W. 2011. Deadlock-free oblivious routing for arbitrary topologies. In *Proceedings of the 25th IEEE International Parallel & Distributed Processing Symposium (IPDPS)*. IEEE Computer Society, 613–624.

Duato, J., Yalamanchili, S., and Ni, L. 2003. *Interconnection Networks An Engineering Approach*. Morgan Kaufmann.

Gómez, C., Gilabert, F., Gómez, M. E., López, P., and Duato, J. 2007. Deterministic versus adaptive routing in fat-trees. In *Proceedings of the IEEE Symposium on Parallel and Distributed Processing*.

Gran, E. G. and Reinemo, S.-A. 2011. Infiniband congestion control, modelling and validation. In *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques (SIMUTools2011)*. (OMNeT++ 2011 Workshop).

Gran, E. G., Zahavi, E., Reinemo, S.-A., Skeie, T., Shainer, G., and Lysne, O. 2011. On the relation between congestion control, switch arbitration and fairness. In *Proceedings of the 11th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing (CCGrid 2011)*.

Guay, W. L., Bogdanski, B., Reinemo, S.-A., Lysne, O., and Skeie, T. 2011. vFtree - A Fat-tree routing algorithm using virtual lanes to alleviate congestion. In *Proceedings of the 25th IEEE International Parallel & Distributed Processing Symposium*.

Guay, W. L., Reinemo, S.-A., Lysne, O., Skeie, T., Johnsen, B. D., and Holen, L. 2010. Host side dynamic reconfiguration with InfiniBand. In *Proceedings of the IEEE International Conference on Cluster Computing*. X. Gu and X. Ma, Eds. IEEE Computer Society, 126–135.

HPCC. 2011. HPC challenge benchmark. http://icl.cs.utk.edu/hpcc/.

IBTA. 2007. Infiniband architecture specification 1.2.1 Ed.

JÜLICH SUPERCOMPUTING CENTRE. 2011. FZJ-JSC JuRoPA. http://www.fz-juelich.de/jsc/juropa/configuration/.

KASHYAP, V. 2006. IP over InfiniBand: Connected mode. IETF.

LEISERSON, C. E. 1985. Fat-Trees: Universal networks for hardware-efficient supercomputing. *IEEE Trans. Computers*.

LIN, X.-Y., CHUNG, Y.-C., AND HUANG, T.-Y. 2004. A multiple LID routing scheme for fat-tree-based Infiniband networks. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposiums*.

MELLANOX TECHNOLOGIES. 2009. MTS3600 36-port 20 Gb/s and 40Gb/s InfiniBand switch system. Product brief. http://www.mellanox.com/related-docs/prod_ib_switch_systems/PB_MTS3600.pdf.

MELLANOX TECHNOLOGIES. 2011. IS5600—648-port InfiniBand chassis switch. http://www.mellanox.com/related-docs/prod_ib_switch_systems/IS5600.pdf.

MVAPICH2. 2011. MVAPICH: MPI over InfiniBand, 10GigE/iWARP and RoCE. http://mvapich.cse.ohio-state.edu/overview/mvapich2/.

ÖHRING, S., IBEL, M., DAS, S., AND KUMAR, M. 1995. On generalized fat trees. In *Proceedings of the 9th International Parallel Processing Symposium*. 37–44.

OPENFABRICS ALLIANCE. 2011. The OpenFabrics Alliance. http://openfabrics.org/.

ORACLE CORPORATION. 2006. Sun Fire X2200 M2 server. http://www.sun.com/servers/x64/x2200/.

ORACLE CORPORATION. 2010. Sun Blade 6048 InfiniBand QDR switched NEM. http://www.oracle.com/us/products/servers-storage/servers/blades/031095.htm.

ORACLE CORPORATION. 2011a. Sun datacenter InfiniBand switch 36. http://www.oracle.com/us/products/servers-storage/networking/infiniband/036206.htm.

ORACLE CORPORATION. 2011b. Sun datacenter InfiniBand switch 648. http://www.oracle.com/us/products/servers-storage/networking/infiniband/034537.htm.

PETRINI, F. AND VANNESCHI, M. 1995. K-ary n-trees: High performance networks for massively parallel architectures. Tech. rep., Dipartimento di Informatica, Universita di Pisa.

QLOGIC. 2007. SilverStorm 9024 switch. http://www.qlogic.com/Resources/Documents/DataSheets/Switches/Edge_Fabric_Switches_datasheet.pdf.

REINEMO, S.-A., SKEIE, T., SØDRING, T., LYSNE, O., AND TØRUDBAKKEN, O. 2006. An overview of qos capabilities in infiniband, advanced switching interconnect, and ethernet. *IEEE Comm. Mag. 44,* 7, 32–38.

RODRIGUEZ, G., MINKENBERG, C., BEIVIDE, R., AND LUIJTEN, R. P. 2009. Oblivious routing schemes in extended generalized fat tree networks. In *Proceedings of the IEEE International Conference on Cluster Computing and Workshops*.

SCHROEDER, M. D., BIRRELL, A. D., BURROWS, M., MURRAY, H., NEEDHAM, R. M., AND RODEHEFFER, T. L. 1991. Autonet: a high-speed, self-configuring local area network using point-to-point links. *IEEE J. Select. Areas Comm. 9,* 8.

SKEIE, T., LYSNE, O., AND THEISS, I. 2002. Layered shortest path (lash) routing in irregular system area networks. In *Proceedings of the Communication Architecture for Clusters Conference*.

TACC. 2011. Texas Advanced Computing Center. http://www.tacc.utexas.edu/.

TOP 500. 2011. Top 500 supercomputer sites. http://top500.org/.

VOLTAIRE. 2011. Voltaire QDR InfiniBand grid director 4700. http://www.voltaire.com/Products/InfiniBand/Grid_Director_Switches/Voltaire_Grid_Director_4700.

ZAHAVI, E., JOHNSON, G., KERBYSON, D. J., AND LANG, M. 2009. Optimized Infiniband fat-tree routing for shift all-to-all communication patterns. *Concurrency Computat. Pract. Exper*.

Received July 2011; revised October 2011, December 2011; accepted December 2011

Paper IV

Discovery and Routing of Degraded Fat-Trees

Bartosz Bogdański, Bjørn Dag Johnsen, Sven-Arne Reinemo,
Frank Olaf Sem-Jacobsen

2012 13th International Conference on Parallel and Distributed Computing, Applications and Technologies

Discovery and Routing of Degraded Fat-Trees

Bartosz Bogdański, Bjørn Dag Johnsen
Oracle Corporation
Oslo, Norway
bartosz.bogdanski@oracle.com
bjorn-dag.johnsen@oracle.com

Sven-Arne Reinemo, Frank Olaf Sem-Jacobsen
Simula Research Laboratory
Lysaker, Norway
svenar@simula.no
frankose@simula.no

***Abstract*—The fat-tree topology has become a popular choice for InfiniBand enterprise systems due to its deadlock freedom, fault-tolerance and full bisection bandwidth. In the HPC domain, InfiniBand fabric is used in almost 42% of the systems on the latest Top 500 list, and many of those systems are based on the fat-tree topology. Despite the popularity of the fat-tree topology, little research has been done to compare the behavior of InfiniBand routing algorithms on degraded fat-tree topologies.**

In this paper, we identify the weaknesses of the current fat-tree routing and propose enhancements that liberalize the restrictions imposed on the routed fabric. Furthermore, we present a thorough analysis of non-proprietary routing algorithms that are implemented in the InfiniBand Open Subnet Manager. Our results show that even though the performance of a fat-tree routed network deteriorates predictably with the number of failed links, fat-tree routing algorithm is still the best choice for severely degraded fat-tree fabrics.

I. Introduction

The fat-tree topology is one of the most common topologies for high performance computing clusters today, and for clusters based on InfiniBand (IB) technology the fat-tree is the dominating topology. This includes large installations such as Nebulae/Dawning, TGCC Curie and SuperMUC [1]. There are three properties that make fat-trees the topology of choice for high performance interconnects: deadlock freedom, the use of a tree structure makes it possible to route fat-trees without special considerations for deadlock avoidance; inherent fault-tolerance, the existence of multiple paths between individual source destination pairs makes it easier to handle network faults; full bisection bandwidth, the network can sustain full speed communication between the two halves of the network.

For fat-trees, as with most other topologies, the routing algorithm is crucial for efficient use of the underlying topology. The popularity of fat-trees in the last decade led to many efforts to improve their routing performance. These proposals, however, have several limitations when it comes to flexibility and scalability. This also includes the current approach that the OpenFabrics Enterprise Distribution [2], the de facto standard for InfiniBand system software, is based on [3], [4]. One problem is the static routing used by IB technology that limits the exploitation of the path diversity in fat-trees as pointed out by Hoefler et al. in [5]. Another problem with the current routing is its shortcomings when routing oversubscribed fat-trees as addressed by Rodriguez et al. in [6]. A third problem, and the one that we are analyzing in this paper, is that any irregularity in a fat-tree fabric makes the subnet manager select a suboptimal fallback routing algorithm.

In this paper, we analyze the performance of four major routing algorithms implemented in InfiniBand Open Subnet Manager (OpenSM) running on fat-trees with a number of random faults. These algorithms are: optimized fat-tree routing devised by Zahavi et al. [4], Layered-Shortest Path Routing (LASH) [7], Deadlock-Free Single-Source-Shortest-Path routing [8] and the default fallback algorithm for OpenSM - MinHop [9]. Through simulations, we show how susceptible is each of these algorithms to random link and switch failures. Moreover, we demonstrate that even though the fat-tree routing is the most susceptible algorithm, it still delivers the highest performance even in extreme cases for non-trivial traffic patterns. Furthermore, we extend the fat-tree algorithm to remove the most common factors leading to lower performance on degraded fat-trees which are the over-restrictive topology discovery and flipping switch anomaly. The major contributions of our work are:

- We present a thorough analysis of non-proprietary routing algorithms implemented in the InfiniBand Open Subnet Manager (OpenSM) running on degraded fat-trees.
- We present enhancements that liberalize the restrictions imposed on the fat-tree discovery and routing of degraded fabrics.

The rest of this paper is organized as follows: we discuss related work in Section II and continue with introducing the InfiniBand Architecture in Section III. We follow with a description of OpenSM routing algorithms in Section IV and discuss our enhancements in Section V. Next, we describe the experimental setup in Section VI followed by the experimental analysis in Section VII. Finally, we conclude in Section VIII.

II. Related Work

There was much research done in the general topic of routing algorithms and fat-tree routing for interconnection networks. First, a thorough survey of interconnection routing algorithms was published by Flich et al. [10], but the authors did not discuss the fat-tree algorithm and focused only on algorithms for routing meshes and tori.

Second, Sem-Jacobsen et al. proposed various methods to improve fault-tolerance in fat-trees [11], [12], [13], [14], however, that work did not compare fat-tree routing to other algorithms available in OpenSM.

There was further work by Bermudez [15], [16], [17] and Vishnu [18] on performance of subnet management for fat-tree topologies (including the discovery process), but the authors dealt strictly with fabric management and did not discuss performance of the routing algorithms themselves.

The popularity of fat-trees in the last decade also led to many efforts trying to improve their routing performance: exploitation of path diversity [5], routing oversubscribed

978-0-7695-4879-1/12 $26.00 © 2012 IEEE
DOI 10.1109/PDCAT.2012.67

fat-trees [6] or utilizing virtual lanes to increase network performance [19].

Unlike previous research on IB routing algorithms, we focus on multiple routing algorithms running on the same fabric. Our work is partially based on [20] where we also analyzed degraded fat-trees, however, only with failing nodes. In this work we widen the scope and, first, consider also failing links and, second, evaluate multiple routing algorithms.

III. The InfiniBand Architecture

InfiniBand networks are referred to as *subnets*, where a subnet consists of a set of hosts interconnected using switches and point-to-point links. An IB fabric constitutes one or more subnets, which can be interconnected using routers. Hosts and switches within a subnet are addressed using local identifiers (LIDs) and a single subnet is limited to 49151 LIDs.

An IB subnet requires at least one subnet manager (SM), which is responsible for initializing and bringing up the network, including the configuration of all the IB ports residing on switches, routers, and host channel adapters (HCAs) in the subnet. At the time of initialization the SM starts in the *discovering state* where it does a sweep of the network in order to discover all switches and hosts. During this phase it will also discover any other SMs present and negotiate who should be the master SM. When this phase is complete the elected SM enters the *master state*. In this state, it proceeds with LID assignment, switch configuration, routing table calculations and deployment, and port configuration. When this is done, the subnet is up and ready for use. After the subnet has been configured, the SM is responsible for monitoring the network for changes.

A major part of the SM's responsibility is to calculate routing tables that maintain full connectivity, deadlock freedom, and proper load balancing between all source and destination pairs. Routing tables must be calculated at network initialization time and this process must be repeated whenever the topology changes in order to update the routing tables and ensure optimal performance.

During normal operation the SM performs periodic *light sweeps* of the network to check for topology changes e.g. a link goes down, a device is added, or a link is removed). If a change is discovered during a light sweep or if a message (trap) signaling a network change is received by the SM, it will reconfigure the network according to the changes discovered. This reconfiguration also includes the steps used during initialization. Moreover, for each device a subnet management agent (SMA) residing on it generates responses to control packets (subnet management packets (SMPs)), and configures local components for subnet management.

IB is a lossless networking technology, where flow-control is performed per *virtual lane* (VL) [21]. VLs are logical channels on the same physical link, but with separate buffering, flow-control, and congestion management resources. The concept of VLs makes it possible to build virtual networks on top of a physical topology. These virtual networks, or layers, can be used for various purposes such as efficient routing, deadlock avoidance, fault-tolerance, and service differentiation. Some routing algorithms, like LASH or DFSSSP, use VLs to break credit dependency cycles and avoid deadlock.

IV. Routing in InfiniBand

In this paper, we focus on fat-tree topologies where the default routing algorithm is the fat-tree routing because it is optimal for fault-free fat-trees. However, if any failure in the fabric occurs or if the fabric does not comply with the strict rules that define a proper fat-tree, the subnet manager fails over to another routing algorithm. In the following subsections, we will describe the algorithms analyzed in this paper. In Section VII, we will analyze and compare the performance of those routing algorithms under different conditions and for different traffic patterns.

A. Fat-Tree Routing Algorithm

The fat-tree topology was introduced by Leiserson in [22], and has since become a common topology in HPC. The fat-tree is a layered network topology with equal link capacity at every tier (applies for balanced fat-trees), and is commonly implemented by building a tree with multiple roots, often following the m-port n-tree definition [23] or the k-ary n-tree definition [24]. An XGFT notation is also used to describe fat-trees and was presented by Öhring [25]. More recently, Zahavi also proposed PGFT and RLFT notations to describe real-life fat-trees [26].

To construct larger topologies, the industry has found it more convenient to connect several fat-trees together rather than building a single large fat-tree. Such a fat-tree built from several single fat-trees is called a multi-core fat-tree. Multi-core fat-trees may be interconnected through the leaf switches using horizontal links [27] or by using an additional layer of switches at the bottom of the fat-tree where every such switch is connected to all the fat-trees composing the multi-core fat-tree [28].

Regardless of how a fat-tree is constructed, the routing function always works in a similar manner. The fat-tree routing is divided into two distinct phases: the upward phase in which a packet is forwarded from the source in the direction of one of the root (top) switches, and the downward phase when a packet is forwarded downwards to the destination. The transition between these two phases occurs at the lowest common ancestor, which is a switch that can reach both the source and the destination through its downward ports. Such an implementation ensures deadlock freedom, and the implementation presented in [4] also ensures that every path towards the same destination converges at the same root switch such that all packets toward that destination follow a single dedicated path in the downward direction. By having a dedicated downward path for every destination, contention in the downward phase is effectively removed (moved to the upward stage), so that packets for different destinations have to contend for output ports in only half of the switches on their path. In oversubscribed fat-trees, the downward path is not dedicated and is shared by several destinations. The fabric discovery complexity for optimized fat-tree routing algorithm is given by $\mathcal{O}(m+n)$ where m is the number of edges (links) and n is the number of vertices (nodes). The routing complexity is $\mathcal{O}(k \cdot n)$, where k is the number of end-nodes and n is the number of switches.

B. Layered-Shortest Path routing

Layered-Shortest Path (LASH) is a deterministic shortest path routing algorithm for irregular networks. All packets are routed using the minimal path, and the algorithm achieves

deadlock freedom by finding and breaking cycles through virtual lanes.

However, LASH does not balance the traffic in any manner, which is especially evident in fat-tree fabrics. The algorithm aims at using the lowest number of VLs and, therefore, routes all possible deadlock-free pairs on the same layer, i.e. using the same links. The computing complexity for LASH is $\mathcal{O}(n^3)$ where n is the number of nodes.

C. Deadlock-Free Single-Source-Shortest-Path routing

Deadlock-Free Single-Source-Shortest-Path routing (DFSSSP) [8] is an efficient oblivious routing for arbitrary topologies developed by Domke et al. [8]. It uses virtual lanes to guarantee deadlock freedom and, in comparison to LASH, aims at not limiting the number of possible paths during the routing process. It also uses improved heuristics to reduce the number of used virtual lanes in comparison to LASH.

The problem with DFSSSP is that for switch-to-switch traffic it assumes deadlock freedom, and does not break any cycles that may occur for switch-to-node and switch-to-switch pairs. The computing complexity for the offline DFSSSP is $\mathcal{O}(n^2 \cdot \log(n))$ where n is the number of nodes [8].

D. MinHop routing

MinHop is the default fallback routing algorithm for the OpenSM. It finds minimal paths among all endpoints and tries to balance the number of routes per link at the local switch. However, using MinHop routing usually leads to credit loops, which may deadlock the fabric. The complexity of MinHop is given by $\mathcal{O}(n^2)$ where n is the number of nodes.

V. Degraded Fat-Tree Discovery

The main weakness of the current implementation in OpenSM of the fat-tree routing is its inability to route by default any non-pure fat-tree fabrics that do not pass the rigorous topology validation. Currently, topology validation fails if the number of up and down links on any two switches on the same level is not equal (e.g. if one link in the whole fabric fails, fat-tree routing falls back to MinHop routing). There are also other scenarios where fat-tree routing fails, but we will not consider those in this paper due to limited space.

Our proposal is to provide three simple enhancements to the routing algorithm to deal properly with any fat-tree fabric. The first enhancement is to liberalize the restrictions on the topology validation and disable link count inconsistency check. This way, fat-tree routing will not fail by default on any incomplete fat-tree.

The second enhancement is to fix the flipped switches issue. This problem occurs when there exists a leaf switch that has no nodes connected. When this happens, such a switch is not classified as a leaf switch by the fat-tree algorithm but as a switch located at level $leaf\_level + 2$. In general, fat-tree routing is runnable on such a fabric, but this counter-intuitive behavior makes troubleshooting more difficult due to incorrect ranks assigned to the switches. The situation is illustrated on Figure 1 and a fix is proposed in Algorithm 1. The problem with providing a fix is that when a ranking conflict occurs in the fabric, the SM can only act

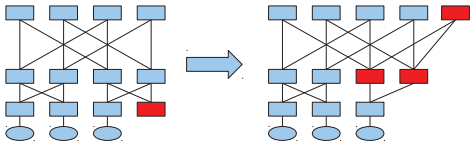


Figure 1: Flipping switch in a simple 3-stage fat-tree.

reactively, i.e. it has to first detect the conflict and then re-rank the fabric. This is cumbersome and it may happen that the conflict will not be detected due to high complexity of the fabric. Therefore, the fix for this problem is built using the last enhancement we propose.

The third and last enhancement is an implementation of *switch roles* mechanism for explicitly defining switch roles that can be later detected by the SM. For this, we use vendor SMP attributes that can be queried via vendor specific SMPs. Basically, each switch in the IB fabric can be assigned a hostname, an IP address and a node description. By using vendor attributes, we are able to make any specific information available to the SM without having a dependency on SM config input or any other out-of-band interfaces for providing config information in a dynamic manner.

We are aware that providing RootGUIDs to the routing algorithm yields the same effect, but currently it requires non-trivial effort to maintain a correct list following (multiple) component replacement operations. On the other hand, switch roles can be saved and restored as part of normal switch configuration maintenance following component replacements since it is not tied to the actual hardware instance like hardware GUIDs.

The switch roles mechanism that we implemented will provide each switch with a simple role that it should adhere to. In our first simple implementation, we will physically tag each switch in the fabric with its respective role, i.e. root switches (placed at the top of the fabric with no uplinks) will have the role "root" and the leaf switches will have the role "leaf".

This not only shortens the fabric discovery time (consistency checks are not required), but almost completely removes the need to discover the fabric from the routing algorithm, which means that the probability of making a mistake during routing table generation will be much lower. In other words, we are decoupling the complex problem of fabric discovery from the routing problem.

Using the switch roles mechanism, we can redefine the $\_\_osm\_ftree\_rank\_fabric(p\_tree)$ OpenSM function in a very simple manner to always construct a proper fat-tree as shown in Algorithm 1.

Algorithm 1 $\_\_osm\_ftree\_rank\_fabric(p\_tree) function$

Require: Firmware vendor specific switch roles.
Ensure: Each sw in the fabric is placed at correct rank.

```
if switch has no CNs then
    if smpquery(switch, role) == leaf then
        switch.rank = tree_rank
    end if
end if
```

VI. Experiment Setup

To evaluate the differences between various routing algorithms, we performed a number of simulations. The subsequent sections will describe the simulation model that we used and examine the topology that we selected.

A. Simulation model

To perform large-scale evaluations of the routing algorithms, we use an InfiniBand model for the OMNEST simulator [29] (OMNEST is a commercial version of the OMNeT++ simulator). The IB model consists of a set of simple and compound modules to simulate an IB network with support for the IB flow control scheme, arbitration over multiple virtual lanes, congestion control, and routing using linear routing tables. The model supports instances of HCAs, switches and routers with routing tables.

The network topology and the routing tables were generated using OpenSM and converted into OMNEST readable format in order to simulate real-world systems. As for our simulations, we measured average throughput per node as a function of the number of failed links and switches under different traffic patterns.

For the uniform traffic pattern, we used a link speed of 20 Gbit/s (4x DDR), a default MTU size of 2kB, a variable packet size (from 84B up to 2kB) and a constant message size of 2kB. The destination was chosen randomly and the network load was set constant at 100%. Each simulation run was repeated 16 times with a different seed and an average was taken.

For HPCC simulations, we implemented a ping-pong traffic pattern that was used to run the HPC Challenge Benchmark tests in the simulator. For the bandwidth tests we used a message size of 1954KB. The MTU size was 2KB and the network load was set constant at 100%. The bandwidth tests were performed on the default 31 ring patterns: one natural-ordered ring and 30 random-ordered rings from which the minimum, maximum and average results were taken. For this measurement, each node sends a message to its left neighbor in the ring and receives a message from its right neighbor. Next, it sends a message back to its right neighbor and receives a return message from its left neighbor.

B. Topology

As a base for our topology we selected a 3-stage 648-port fat-tree where each two leaf switches are interconnected with each other with 12 links and form a single switching unit. For each subsequent simulation run, we randomly failed one link in the fabric (both in the up and down direction), until 85 of all non-horizontal links in the fabric were disconnected. In a 648-node 3-stage fabric, there are 1296 non-horizontal links (as there are 36 36-port middle-stage switches), so the number 85 represents approximately 6.5% of all links. Furthermore, we added three measurements where the number of failed links is 10% (130 failed links), 15% (194 failed links) and 20% (259 failed links) to accommodate for extreme situations. The links were not failed incrementally, but randomly, that is, the set of failed links from one scenario can and will usually differ from the set of failed links from a subsequent scenario. This was done to show the location of a failed link in the fabric also influences the network performance. Furthermore, for comparison, we added simulations where we did not fail links but whole switches. The number of failed links when a single switch fails is 36, so the measurements are less granular.

We chose a 3-stage 648-port fat-tree as the base fabric because it is a common configuration used by switch vendors in their own 648-port systems [30], [31], [32]. Additionally, such switches are often connected together to form larger installations like the JuRoPa supercomputer [27].

VII. Performance Evaluation

A. Uniform Traffic

Figure 2 shows the results for uniform traffic scenario. The first observation is the fact that the fat-tree routing algorithm is very susceptible to link failure (reducing the performance by 35% for 80 failed links). Other algorithms are also negatively influenced by link failure, but not to such an extent, and LASH delivers constant, while very low, throughput. Secondly, we observe that if the number of failed links reaches 34 (approximately 2.6%), the DFSSSP routing algorithm outperforms the fat-tree routing algorithm. This cut-off point may be used to define whether a particular fabric is still a fat-tree topology or has become a hybrid topology.

Lastly, we observe that the plot for each routing algorithm apart from LASH is not a strictly monotonically decreasing function as one would expect (i.e. less links is lower throughput). This means - due to random link failure - that the location of a failed link also influences the network performance. We checked in detail which links have failed in each case, and we learned that if a link connecting a leaf switch with a middle-stage switch fails, the performance drop is much lower than if a link fails between a middle-stage switch and a root switch. This is because, in this particular topology, each leaf switch is connected with 3 links with each of its four upward neighbors. A failure of a single link does not limit connectivity, but only changes the number of paths per port in the port group that connects to the upward switch (failure is localized). On the other hand, the root switches are connected with only a single link with each downward neighbor, which means that a link failure at this stage leads to a complete lack of connectivity between the two switches and the need to distribute the paths across the whole network (failure with global influence).

On Figure 3 we see the same scenario, but instead of single links, we failed whole 36-port switches (up to 7 switches which gives 252 failed links). We see that fat-tree routing is slightly more susceptible (in terms of delivered network performance) to link failure than to switch failure. It is because fat-tree routing unknowingly chooses switches with a large number of failed links (but still aims to divide the destinations equally among all switches), which leads to much higher traffic congestion on those switches that have limited bandwidth (more failed links). If a whole switch fails, then the rest of the traffic is evenly distributed among all other devices, so bandwidth is higher than in case of single-link failures.

MinHop, however, deals better with link failure than with switch failure, which is surprisingly explained by inefficient upward balancing that MinHop does. Due to multiple links connecting middle-stage switches with leaf switches, MinHop balancing is broken even for a fully populated fabric. In our case, for middle-stage switches, the odd ones (counting from the left) have 16 upward paths per each port 1, 2, 3, zero paths per ports 4, 5, 6, and again 16 paths per each

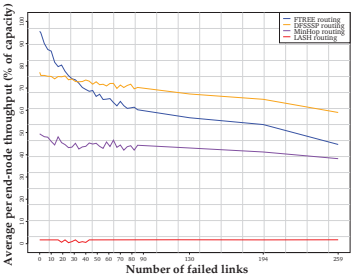


Figure 2: Comparing the algorithms for uniform traffic and failing links

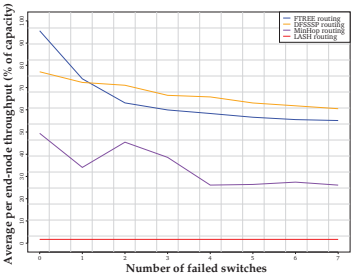


Figure 3: Comparing the algorithms for uniform traffic and failing switches

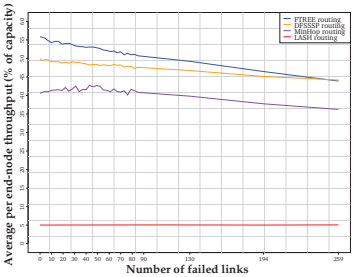


Figure 4: Random Ring - Average results for failing links scenario

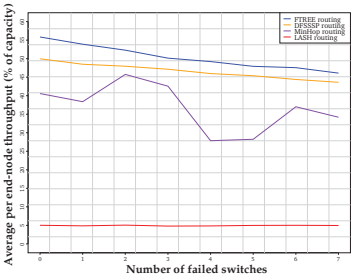


Figure 5: Random Ring - Average results for failing switches scenario

port 7, 8, 9 and so on. The even switches have a reverse path assignment, i.e. ports 1, 2, 3 have 0 paths and ports 4, 5, 6 have 16 paths per port and so on. For root switches, for odd ones (again counting from the left), even ports have 0 paths and for even root switches odd ports have 0 paths. This means that there are 324 links that have 0 paths and if such a link failure happens at an unused port (25% probability), it will not influence the network performance. The further explanation for the positive spike seen for a failing switches scenario is given in the next subsection where this behavior is more evident as balancing plays a more important role there.

To summarize, the results indicate that fat-tree routing delivers high performance only until a certain threshold of failures is reached. However, as shown in the next subsection, we will demonstrate that this threshold shifts if a different traffic pattern is used.

B. HPC Challenge Benchmark

Figure 4 and Figure 5 show the results for the HPC Challenge Benchmark simulations where the average from the 30 random rings was taken. We do not present the results for the maximum, minimum and natural rings because they do not provide any additional insight. For this kind of traffic, if we fail single links, all the algorithms deliver lower performance with each subsequent failed link. However, DFSSSP outperforms fat-tree routing only at 20% of failed links (not at 2.6% like for uniform traffic). Furthermore, as seen on Figure 5, which also confirms the previous observation for uniform traffic, fat-tree routing is less susceptible to switch failures than link failures and is not outperformed by DFSSSP even if close to 20% of links fail.

An interesting anomaly occurs with MinHop for failing switches scenario, where failing a switch in a fabric does not necessarily mean that there will be a performance drop. This also confirms the observations for uniform traffic where the plot is similar but these spikes are less pronounced. The explanation is that for a fully populated fabric, MinHop does not balance the paths correctly as described and explained in the previous subsection. However, the visible spikes on Figure 5 are a result of MinHop not being able to properly balance a regular fabric, but being able to balance the paths for an irregular fabric. In detail, for zero switch failures and for a single switch failure, MinHop does not properly balance the paths by leaving 25% of all ports unused. The performance spike for the scenario with 2 and 3 failed switches is explained by proper path balancing. Similarly, the drop for the scenario with 4 and 5 switches is explained by the lack of proper balancing. The last two scenarios again have proper balancing, but only for middle-stage switches while the downward paths from the root switches remain imbalanced (there are unused ports). This counter-intuitive behavior is a bug in the sorting function for MinHop and is caused by two factors: the topology with port groups (i.e. multiple links connecting the same devices) and a corresponding bug in the sorting function during the balancing phase of the MinHop routing, which, if middle-stage switches have an even number of upward ports but an

odd number of ports in a port group, does not deliver the expected results. If the number of upward ports is odd, then the balancing is proper. The question remains unanswered whether the MinHop anomaly regarding balancing is a simple bug or a more complex implication of the algorithm.

To summarize, the results for HPC Challenge Benchmark clearly demonstrate that even though fat-tree routing is susceptible to link failures, it still delivers the highest performance of all analyzed routing algorithms for non-trivial traffic pattern on degraded fat-trees.

VIII. Conclusions

In this paper, we identified flaws in the existing fat-tree routing algorithm for InfiniBand networks, and we proposed three extensions that alleviate problems encountered when discovering and routing degraded fabrics. First, we liberalized topology validation to make fat-tree routing more versatile. Second, we proposed a switch tagging through vendor SMP attributes that can be queried via vendor specific SMPs and are used to configure the switches with specific fabric roles, which decouples topology discovery from actual routing. Lastly, we proposed solving the flipping switches problem through using the SMP attributes. With this insight, we compared four non-proprietary routing algorithms running on degraded fat-trees. The results indicate that the fat-tree routing is still the preferred algorithm even if the number of failures is very large.

In the future, we plan to work on native IB-IB routing. The current work will be expanded to cover hybrid fabrics with multiple IB subnets and concurrently running multiple routing protocols.

References

[1] "Top 500 supercomputer sites," http://top500.org/, November 2010.

[2] "The OpenFabrics Alliance," http://openfabrics.org/.

[3] C. Gomez, F. Gilabert, M. E. Gomez, P. Lopez, and J. Duato, "Deterministic versus Adaptive Routing in Fat-Trees," in *Workshop on Communication Architecture on Clusters, IPDPS*, 2007.

[4] E. Zahavi, G. Johnson, D. J. Kerbyson, and M. Lang, "Optimized Infiniband fat-tree routing for shift all-to-all communication patterns," in *Concurrency and Computation: Practice and Experience*, 2009.

[5] T. Hoefler, T. Schneider, and A. Lumsdaine, "Multistage switches are not crossbars: Effects of static routing in high-performance networks," in *Cluster Computing, 2008 IEEE International Conference on*, 29 2008-Oct. 1 2008, pp. 116–125.

[6] G. Rodriguez, C. Minkenberg, R. Beivide, and R. P. Luijten, "Oblivious Routing Schemes in Extended Generalized Fat Tree Networks," *IEEE International Conference on Cluster Computing and Workshops*, 2009.

[7] T. Skeie, O. Lysne, and I. Theiss, "Layered shortest path (lash) routing in irregular system area networks," in *Proceedings of Communication Architecture for Clusters*, 2002.

[8] J. Domke, T. Hoefler, and W. Nagel, "Deadlock-Free Oblivious Routing for Arbitrary Topologies," in *Proceedings of the 25th IEEE International Parallel and Distributed Processing Symposium*. IEEE Computer Society, May 2011, pp. 613–624.

[9] J. Duato, S. Yalamanchili, and L. Ni, *Interconnection Networks An Engineering Approach*. Morgan Kaufmann, 2003.

[10] J. Flich, T. Skeie, A. Mejia, O. Lysne, P. López, A. Robles, J. Duato, M. Koibuchi, T. Rokicki, and J. Sancho, "A survey and evaluation of topology agnostic routing algorithms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 3, pp. 405–425, March 2012.

[11] F. O. Sem-Jacobsen, T. Skeie, O. Lysne, and J. Duato, "Dynamic fault tolerance in fat-trees," *IEEE Transactions on Computers*, vol. 60, no. 4, pp. 508–525, April 2011.

[12] F. O. Sem-Jacobsen and O. Lysne, "Fault tolerance with shortest paths in regular and irregular networks," in *22nd IEEE International Parallel & Distributed Processing Symposium*, Y. Robert, Ed., IEEE. Unknown, April 2008.

[13] F. O. Sem-Jacobsen, T. Skeie, O. Lysne, and J. Duato, "Dynamic fault tolerance with misrouting in fat trees," in *Proceedings of the International Conference on Parallel Processing (ICPP)*, W. chi Feng, Ed. IEEE Computer Society, August 2006, pp. 33–45.

[14] F. O. Sem-Jacobsen, T. Skeie, and O. Lysne, "A dynamic fault-torlerant routing algorithm for fat-trees," in *International Conference on Parallel and Distributed Processing Techniques and Applications, Las Vegas, Nevada, USA, June 27-30*, H. R. Arabnia, Ed. CSREA Press, 2005, pp. 318–324.

[15] A. Bermudez, R. Casado, F. Quiles, T. Pinkston, and J. Duato, "On the infiniband subnet discovery process," *Proceedings of International Conference on Cluster Computing*, pp. 512–517, 1-4 Dec. 2003.

[16] A. Bermudez, R. Casado, F. Quiles, and J. Duato, "Fast routing computation on infiniband networks," *Transactions on Parallel and Distributed Systems*, vol. 17, no. 3, pp. 215–226, March 2006.

[17] A. Bermudez, R. Casado, F. Quiles, T. Pinkston, and J. Duato, "Evaluation of a subnet management mechanism for infiniband networks," *Proceedings of International Conference on Parallel Processing*, pp. 117–124, 6-9 Oct. 2003.

[18] A. Vishnu, A. Mamidala, H.-W. Jin, and D. Panda, "Performance modeling of subnet management on fat tree infiniband networks using opensm," *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, pp. 8 pp.–, 4-8 April 2005.

[19] W. L. Guay, B. Bogdanski, S.-A. Reinemo, O. Lysne, and T. Skeie, "vFtree - A Fat-tree Routing Algorithm using Virtual Lanes to Alleviate Congestion," in *Proceedings of the 25th IEEE International Parallel & Distributed Processing Symposium*, 2011.

[20] B. Bogdanski, F. O. Sem-Jacobsen, S.-A. Reinemo, T. Skeie, L. Holen, and L. P. Huse, "Achieving predictable high performance in imbalanced fat trees," in *Proceedings of the 16th IEEE International Conference on Parallel and Distributed Systems*, X. Huang, Ed. IEEE Computer Society, 2010, pp. 381–388.

[21] W. J. Dally, "Virtual-channel flow control," *IEEE Transactions on Parallel and Distributed Systems*, vol. 3, no. 2, pp. 194–205, March 1992.

[22] C. E. Leiserson, "Fat-Trees: Universal Networks for Hardware-Efficient Supercomputing," *IEEE Transactions on Computers*, 1985.

[23] X.-Y. Lin, Y.-C. Chung, and T.-Y. Huang, "A Multiple LID Routing Scheme for Fat-Tree-Based Infiniband Networks," *Proceedings of IEEE International Parallel and Distributed Processing Symposiums*, 2004.

[24] F. Petrini and M. Vanneschi, "K-ary n-trees: High performance networks for massively parallel architectures," Dipartimento di Informatica, Universita di of Pisa, Tech. Rep., 1995.

[25] S. Öhring, M. Ibel, S. Das, and M. Kumar, "On Generalized Fat Trees," in *Proceedings of 9th International Parallel Processing Symposium*, 1995, pp. 37–44.

[26] E. Zahavi, "D-Mod-K Routing Providing Non-Blocking Traffic for Shift Permutations on Real Life Fat Trees," http://www.technion.ac.il/~ezahavi/, September 2010.

[27] "Julich Supercomputing Centre," http://www.fz-juelich.de/jsc/juropa/configuration/.

[28] "Texas Advanced Computing Center," http://www.tacc.utexas.edu/.

[29] E. G. Gran and S.-A. Reinemo, "Infiniband congestion control, modelling and validation," in *4th International ICST Conference on Simulation Tools and Techniques (SIMUTools2011, OMNeT++ 2011 Workshop)*, 2011.

[30] "Sun Datacenter InfiniBand Switch 648," Oracle Corporation, http://www.oracle.com/us/products/servers-storage/networking/infiniband/034537.htm.

[31] "Voltaire QDR InfiniBand Grid Director 4700," http://www.voltaire.com/Products/InfiniBand/Grid_Director_Switches/Voltaire_Grid_Director_4700.

[32] "IS5600 - 648-port InfiniBand Chassis Switch," Mellanox Technologies, http://www.mellanox.com/related-docs/prod_ib_switch_systems/IS5600.pdf.

Paper V

Making the Network Scalable: Inter-subnet Routing in InfiniBand

Bartosz Bogdański, Bjørn Dag Johnsen, Sven-Arne Reinemo, José Flich

Making the Network Scalable: Inter-subnet Routing in InfiniBand

Bartosz Bogdański[1], Bjørn Dag Johnsen[1], Sven-Arne Reinemo[2], and José Flich[3]

[1] Oracle Corporation, Oslo, Norway
{bartosz.bogdanski,bjorn-dag.johnsen}@oracle.com
[2] Simula Research Laboratory, Lysaker, Norway
svenar@simula.no
[3] Universidad Politécnica de Valencia, Valencia, Spain
jflich@disca.upv.es

Abstract. As InfiniBand clusters grow in size and complexity, the need arises to segment the network into manageable sections. Up until now, InfiniBand routers have not been used extensively and little research has been done to accommodate them. However, the limits imposed on local addressing space, inability to logically segment fabrics, long reconfiguration times for large fabrics in case of faults, and, finally, performance issues when interconnecting large clusters, have rekindled the industry's interest into IB-IB routers. In this paper, we examine the routing problems that exist in the current implementation of OpenSM and we introduce two new routing algorithms for inter-subnet IB routing. We evaluate the performance of our routing algorithms against the current solution and we show an improvement of up to 100 times that of OpenSM.

1 Introduction

Until recently, the need for routers in InfiniBand (IB) networks was not evident and all the essential routing and forwarding functions were performed by layer-2 switches. However, with the increased complexity of the clusters, the need for routers becomes more obvious, and leads to more discussion about native IB routing [1,2,3]. Obsidian Research was the first company to see the need for routing between multiple subnets, and provided the first hardware to do that in 2006 [4].

There are several reasons for using routing between IB subnets with the two main being address space scalability and fabric management containment. Address space scalability is an issue for large installations whose size is limited by the number of available *local identifiers* (LIDs). Hosts and switches within a subnet are addressed using LIDs and a single subnet is limited to 49151 unicast LIDs. If more end-ports are required, then the only option is to combine multiple subnets by using one or more IB routers. Because LID addresses have local visibility, they can be reused in the subnets connected by routers, which theoretically yields an unlimited addressing space. It is worth observing that there are multiple suggestions to expand the address space of IB without introducing

F. Wolf, B. Mohr, and D. an Mey (Eds.): Euro-Par 2013, LNCS 8097, pp. 685–698, 2013.

routers. One of the more mature proposals aims at extending the LID addressing space to 32 bits [5], however, it would not be backward compatible with older hardware, which limits its usability.

Fabric management containment has three major benefits: 1) fault isolation, 2) increased security, and 3) intra-subnet routing flexibility. First, by dividing a large subnet into several smaller ones, faults or topology changes are contained to a single subnet and the subnet reconfiguration will not pass through a router to other subnets. This shortens the reconfiguration time and limits the impact of a fault. Second, from a security point of view, segmenting a large fabric into subnets using routers means that the scope of most attacks is limited to the attacked subnet [6]. Third, from a routing point of view, fabric management containment leads to more flexible routing schemes. This is particularly advantageous in case of a hybrid fabric that consists of two or more regular topologies. For example, a network may consist of a fat-tree part interconnected with a mesh or a torus part (or any other regular topology). The problem with managing this in a single subnet is that it is not straightforward to route each part of the subnet separately because intra-subnet routing algorithms have a subnet scope. Moreover, there are no general purpose agnostic routing algorithms for IB that will provide optimal performance for a hybrid topology. However, if a hybrid topology is divided into smaller regular subnets then each subnet can be routed using a different routing algorithm that is optimized for a particular subnet. For example, a fat-tree routing algorithm could route the fat-tree part and the dimension-order routing could route the mesh part of the topology. This is because each subnet can run its own subnet manager (SM) that configures only the ports on the local subnet and routers are non-transparent to the subnet manager.

In this paper, we present two inter-subnet routing algorithms for IB. The first one, *inter-subnet source routing* (ISSR), is an agnostic algorithm for interconnecting any type of topology. The second one is fat-tree specific and only interconnects two or more fat-trees. With these algorithms we solve two problems: how to optimally choose a local router port for a remote destination and how to best route from the router to the destination. We compare the algorithms against the solution that is implemented in OpenSM. Inter-subnet routing in OpenSM is at the time of writing very limited, the configuration is tedious and the performance is only usable for achieving connectivity - not for high performance communication between multiple sources and destinations [2]. It is, however, the only available inter-subnet routing method for IB.

The rest of this paper is organized as follows: we discuss related work in Sect. 2, and we introduce the IB Architecture in Sect. 3. We follow with a description of our proposed layer-3 routing for IB in Sect. 4. Next, we continue with a presentation and discussion of our results in Sect. 5. Finally, we conclude in Sect. 6.

2 Related Work

Obsidian Strategics was the first company to demonstrate a device marketed as an IB-IB router (the Longbow XR) in 2006 [4]. That system highlighted the need

for subnet isolation through native IB-IB routing. The Longbow XR featured a content-addressable memory for fast address resolution and supported up to 64k routes. The drawbacks of the router included a single 4x SDR link, and its primary application was disaster recovery - it was aimed at interconnecting IB subnets spanning large distances as a range extender. Furthermore, the Longbow XR appears to the subnet manager as a transparent switch, so the interconnected subnets are merged together into one large subnet. When releasing the router, Obsidian argued that while the IB specification 1.0.a defines the router hardware well, the details of subnet management interaction (like routing) are not fully addressed. This argument is still valid for the current release of the specification [7]. In 2007, Prescott and Taylor verified how range extension in IB works for campus area and wide area networks [8]. They demonstrated that it is possible to achieve high performance when using routers to build IB wide area networks. However, they did not mention the deadlock issues that can occur when merging subnets, and they only focused on remote traffic even though local traffic can be negatively affected by suboptimal routing in such a hybrid fabric. In 2008, Southwell presented how native IB-IB routers could be used in System Area Networks [1]. He argued that IB could evolve from being an HPC-oriented technology into a strong candidate for future distributed data center applications or campus area grids. While the need for native IB-IB routing was well-demonstrated, Southwell did not address the routing, addressing and deadlock issues. In 2011, Richling et al. [9] addressed the operational and management issues when interconnecting two clusters over a distance of 28 kilometers. They described the setup of hardware and networking components, and the encountered integration problems. However, they focus on IB-IB routing in the context of range extension and not on inter-subnet routing between local subnets.

When reviewing the literature, we noticed that the studies of native IB-IB routing is focused on disaster recovery and interconnection of wide area IB networks. Our work explores the foundations of native IB-IB routing in the context of performance and features in inter-subnet routing between local subnets. Furthermore, we assume full compliance with the IB specification and we deal with issues previously not mentioned including the deadlock problem and path distribution.

3 The InfiniBand Architecture

InfiniBand is a serial point-to-point full-duplex interconnection network technology, and was first standardized in October 2000 [7]. The current trend is that IB is replacing proprietary or low-performance solutions in the high performance computing domain [10], where high bandwidth and low latency are the key requirements. The de facto system software for IB is OFED developed by dedicated professionals and maintained by the OpenFabrics Alliance [5].

Every IB subnet requires at least one subnet manager (SM), which is responsible for initializing and bringing up the network, including the configuration of all the IB ports residing on switches, routers, and host channel adapters (HCAs) in the subnet. At the time of initialization the SM starts in the *discovering state*

where it does a sweep of the network in order to discover all switches and hosts. During this phase it will also discover any other SMs present and negotiate who should be the master SM. When this phase is completed the SM enters the *master state*. In this state, it proceeds with LID assignment, switch configuration, routing table calculations and deployment, and port configuration. At this point the subnet is up and ready for use. After the subnet has been configured, the SM is responsible for monitoring the network for changes.

A major part of the SM's responsibility is routing table calculations. Routing of the network aims at obtaining full connectivity, deadlock freedom, and proper load balancing between all source and destination pairs in the local subnet. Routing tables must be calculated at network initialization time and this process must be repeated whenever the topology changes in order to update the routing tables and ensure optimal performance. Despite being specific about intra-subnet routing, the IB specification does not say much about inter-subnet routing and leaves the details of the implementation to the vendors.

IB is a lossless networking technology, and under certain conditions it may be prone to *deadlocks* [11,12]. Deadlocks occur because network resources such as buffers or channels are shared and because packet drops are usually not allowed in lossless networks. The IB specification explicitly forbids IB-IB routers to cause a deadlock in the fabric irrespective of the congestion policy associated with the inter-subnet routing function. Designing a generalized deadlock-free inter-subnet routing algorithm where the local subnets are arbitrary topologies is challenging. In this paper we limit our scope to fat-tree topologies and by making sure our routing functions use only the standard up/down routing mechanism, we eliminate the deadlock problem.

3.1 Native InfiniBand Routers

The InfiniBand Architecture (IBA) supports a two-layer topological division. At the lower layer, IB networks are referred to as subnets, where a subnet consists of a set of hosts interconnected using switches and point-to-point links. At the higher level, an IB fabric constitutes one or more subnets, which are interconnected using routers. Hosts and switches within a subnet are addressed using LIDs and a single subnet is limited to 49151 LIDs. LIDs are local addresses valid only within a subnet, but each IB device also has a 64-bit *global unique identifier* (GUID) burned into its non-volatile memory. A GUID is used to form a GID - an IB layer-3 address. A GID is created by concatenating a 64-bit subnet ID with the 64-bit GUID to form an IPv6-like 128-bit address. In this paper, we when using the term GUID we mean a port GUIDs, i.e. the GUIDs assigned to every port in the IB fabric.

IB-IB routers operate at the layer-3 of IB addressing hierarchy and their function is to interconnect layer-2 subnets as shown in Fig. 1(a). A thorough description of the inter-subnet routing scheme is currently out of scope of the IBA specification and much freedom is given to the router vendors when implementing inter-subnet routing. The inter-subnet routing process defined in the IBA specification is similar to the routing in TCP/IP networks. First, if an end-node

want to send a packet to another subnet, the address resolution makes the local router visible to that end-node. The end-node puts the local router's LID address in the *local routing header* (LRH) and the final destination address (GID) in the *global routing header* fields. When the packet reaches a router, the packet fields are replaced (the source LID is replaced with the LID of the router's egress port, the destination LID is replaced with the LID of the next-hop port, and CRCs are recomputed) and the packet is forwarded to the next hop. The pseudo code for the rest of the packet relay model is described in [7] on page 1082. In this paper, we will only consider topologies similar to that presented in Fig. 1(a), i.e. cases where one or more subnets are directly connected using routers. Furthermore, each subnet must be a fat-tree topology and it must be directly attached to the other subnets without any transit subnets in between.

4 Layer-3 Routing in InfiniBand

Up until now, IB-IB routers were considered to be superfluous. Even the concept of *routing*, which in IP networks strictly refers to layer-3 routers, in IB was informally applied to forwarding done by layer-2 switches that process packets based only on their LID addresses. With the increasing size and complexity of subnets the need for routers has become more evident. There are two major problems with inter-subnet routing: which router should be chosen for a particular destination (first routing phase) and which path should be chosen by the router to reach the destination (second routing phase). Solving these problems in an optimal manner is not possible if adhering to the current IB specification: the routers are non-transparent subnet boundaries (local SM cannot see beyond), so full topology visibility condition is not met. However, in this paper, by using regularity features provided by the fat-tree topology, we propose a solution for these problems. Nevertheless, for more irregular networks where the final destination is located behind another subnet (at least two router hops required) there may be a need for a super subnet manager that coordinates between the local subnet managers and establishes the path through the transit subnet. We consider such scenarios to be future work. In this section we present two new routing algorithms: Inter-Subnet Source Routing (ISSR) and Inter-Subnet Fat-Tree Routing (ISFR). ISFR is an algorithm designed to work best on fat-trees while ISSR is a more generic algorithm that works well on other topologies also. However, in this paper we only focus on fat-trees and fat-tree subnets as the deadlock problem becomes more complex when dealing with irregular networks. Nevertheless, we plan to address deadlock free inter-subnet IB routing in a more general manner in subsequent publications.

4.1 Inter-subnet Source Routing

We designed ISSR to be a general purpose routing algorithm for routing hybrid subnets. It needs to be implemented both in the SM and the router firmware. It is a deterministic oblivious routing algorithm that always uses the same path

for the same pair of nodes. In general, it offers routing performance comparable to ISFR algorithm provided a few conditions explained in Sect. 5 are met.

The routing itself consists of two phases. First, for the local phase (choosing an ingress router port for a particular destination) this algorithm uses a mapping file. Whereas the *find_router*() function which chooses the local router looks almost exactly the same (it just matches the whole GID) for ISSR algorithm as for the OpenSM routing algorithm, the main difference lies in the setup of the mapping file. In our case, we provide full granularity meaning that instead of only a subnet prefix as for the OpenSM inter-subnet routing, the file now contains a fully qualified port GID. This means that we can map every destination end-port to a different router port while OpenSM routing can only match a whole subnet to a single router port. In the case of ISSR, an equal number of destinations is mapped to a number of ports in a round robin manner. In our example, *dst_gid1* and *dst_gid3* are routed through port 1 and port 2 on router A, and *dst_gid2* and *dst_gid4* are routed through the same ports on router B. Backup and default routes can also be specified.

Code 1. A high-level example of a mapping file for ISSR and ISFR algorithms

```
1: dst_gid1    router_A_port_1_guid
2: dst_gid2    router_B_port_1_guid
3: dst_gid3    router_A_port_2_guid
4: dst_gid4    router_B_port_2_guid
5: #default route
6:     *       router_A_port_1_guid
7:     *       router_B_port_1_guid
```

Second difference is the implementation of the additional code in the router firmware. A router receiving a packet destined to another subnet will source route that packet. The routing decision is based both on the source LID (of the original source or the egress port of the previous-hop router in a transit subnet scenario) and the destination LID (final destination LID or the LID of the next-hop ingress router port). The router knows both these values because it sees the subnets attached to it. To obtain the destination LID, a function mapping the destination GID to a destination LID or returning the next-hop LID based on the subnet prefix located in the GID is required. In our case, this function is named *get_next_LID* (line 2 of the pseudo code in Algorithm 1).

The algorithm calculates a random number based on the source and destination LIDs. This is done in a deterministic manner so that a given src-dst pair always generates the same number, which prevents out of order delivery when routing between subnets and, unlike round-robin, makes sure that each src-dst always uses the same path through the network. This number is used to select a single egress port from a set of possible ports. There is a set of possible ports because a router may be attached to more than two subnets and therefore a two-step port verification is necessary: first choose the ports attached to the subnet

Algorithm 1. choose_egress_port() function in ISSR

Require: Receive an inter-subnet packet
Ensure: Forward the packet in a deterministic oblivious manner
1: **if** *received_intersubnet_packet*() **then**
2: $dstLID = get\_next\_LID(dGID)$
3: $srand(srcLID + dstLID)$
4: $port\_set = choose\_possible\_out\_ports()$
5: $e\_port = port\_set[(rand()\%port\_set.size)]$
6: **end if**

(or in the direction of the subnet) in which the destination is located, and then, by using a simple hash based on a modulo function, choose the egress port.

4.2 Inter-subnet Fat-Tree Routing

As mentioned previously, a problem that needs to be solved is the communication between SMs that are in different subnets and are connected through non-transparent routers. Our solution is based on the fact that the IBA specification does not give the exact implementation for inter-subnet routing, so our proposal provides an interface in the routers through which the SMs will communicate. In other words, we implement handshaking between two SMs located in neighboring subnets. The algorithm uses the previously defined file format containing the GID-to-router port mappings in Code 1. The ISFR algorithm is presented in Algorithm 2. Like the ISSR algorithm, it is also implemented in the router device. ISFR works only on fat-trees and with fat-tree routing running locally in every subnet. It will fall back to ISSR if those conditions are not met.

Algorithm 2. query_down_for_egress_port() function in ISFR

Require: Local fat-tree routing is finished
Require: Received the mapping file
Ensure: Fat-tree like routing tables throughout the fabric
1: **if** *received_mapping_files* **then**
2: **for all** *port in down_ports* **do**
3: $down\_switch = get\_node(port)$
4: $lid = get\_LID\_by\_GID(GID)$
5: **if** $down\_switch.routing\_table[lid] == primary\_path$ **then**
6: $e\_port = port$
7: **end if**
8: **end for**
9: **end if**

Every single router in a subnet receives the port mappings from its local SM and is thereby able to learn which of its ports are used for which GIDs. Next, for each attached subnet, the router queries the switches in the destination subnets

to learn which of the switches has the primary path to that subnet's HCAs. If we assume a proper fat-tree (full bisection bandwidth) with routers on the top of the tree, then after such a query is performed, each router will have one path per port in the downward direction for each destination located in a particular subnet. In other words, if we substituted the top routers with switches, the routing tables for the pure fat-tree and the fat-tree with routers on top would be the same.

5 Simulations

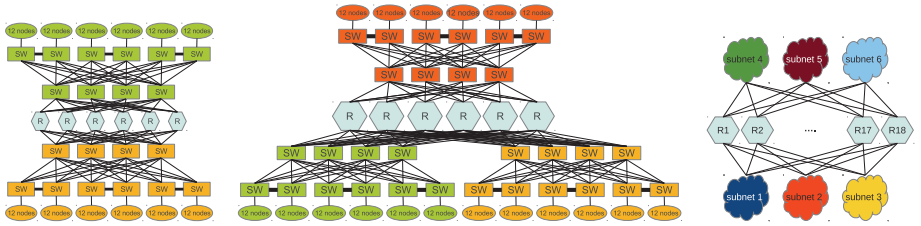
To perform large-scale evaluations and verify the scalability of our proposal, we use an InfiniBand model for the OMNEST/OMNeT++ simulator [13]. The IB model consists of a set of simple and compound modules to simulate an IB network with support for the IB flow control scheme, arbitration over multiple virtual lanes, congestion control, and routing using linear routing tables. In each of the simulations, we used a link speed of 20 Gbit/s (4x DDR) and Maximum Transfer Unit (MTU) equal to 2048 bytes. Furthermore, we use uniform, non-uniform and HPCC traffic patterns. We used synthetic traffic patterns to show baseline performance as these patterns have a predictable and easily understandable behavior, and are general rather than specific to a given application. We do not provide the baseline results for the same topologies implemented as a single subnet because ISFR routing provides exactly the same performance.

The simulations were performed on three different topologies shown in Fig. 1. Each of the topologies can be classified as a 3-stage (i.e. having three routing/switching stages and one node stage) fat-tree with routers placed on top of the tree (instead of normally placing root switches there). Even though there is a dedicated fat-tree routing algorithm delivering high performance on almost any fat-tree, we still decided to subnet a fat-tree fabric. The reason for that is that we consider the fat-tree topology to be a very good proof-of-concept topology for inter-subnet routing testing.

The fat-tree topology is scalable and by changing the number of ports we are able to vary the size of the topologies and show how our algorithms scale with regards to the number of nodes and subnets that are interconnected. All our subnets are 2-stage fat-trees that are branches in a larger 3-stage fat-tree so we can use routers and our routing algorithms to demonstrate how to seamlessly interconnect smaller fat-tree installations without using oversubscription. We chose a 3-stage 648-port fat-tree as the base fabric because it is a common configuration used by switch vendors in their own 648-port systems [14,15,16]. Additionally, such switches are often connected together to form larger installations like the JuRoPA supercomputer [17].

5.1 Routing Algorithm Comparison

We perform three sets of simulations: with uniform traffic, with non-uniform traffic and we run the HPCC benchmark. For non-uniform traffic we vary packet

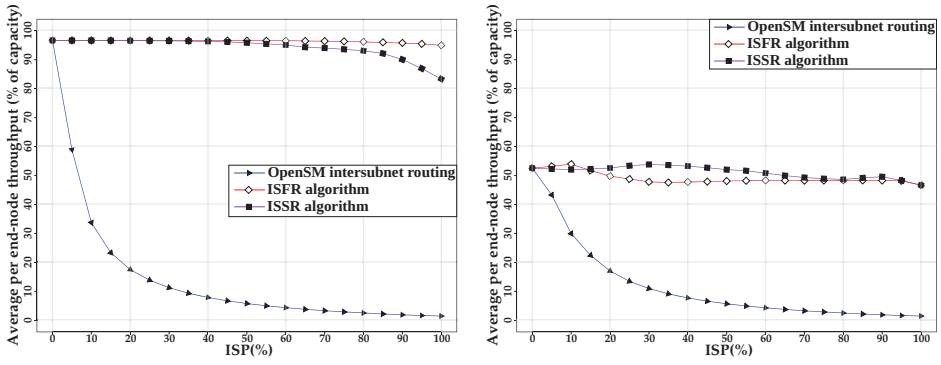


(a) 2-subnet fabric (b) 3-subnet fabric (c) 6-subnet fabric

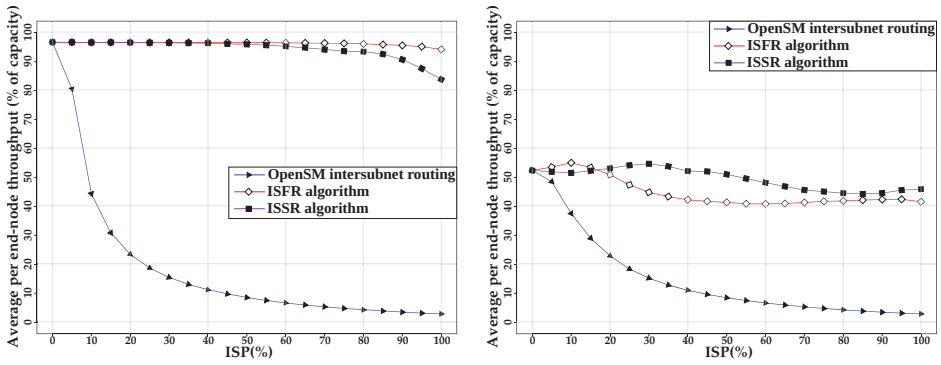
Fig. 1. Topologies used for the experiments

length from 84 bytes to 2 kB, keep the message length constant at 2 kB. We also introduce some randomly preselected hot spots (different for every random seed, not varying in time): one hot spot per subnet, with a probability of $ISP * 0.05$ for remote traffic and the same hot spot with a probability of $(1 - ISP) * 0.05$ for local traffic. The ISP (Inter-Subnet Percentage) value is the probability that a message will be sent to the local or the remote subnet. It varies from 0% (where all messages remain local) to 100% (where all messages are sent to remote subnets). The non-hot spot destined part of the traffic selects their destination randomly from all other available nodes. This means that there could be some other random hot spots that vary in time, and some nodes could also contribute unknowingly to the preselected hot spots. We express the measured throughput as the percentage of the available bandwidth for all the scenarios. The parameters for that traffic pattern were chosen to best illustrate the impact of congestion on the routing performance, which is a good baseline for algorithm comparison.

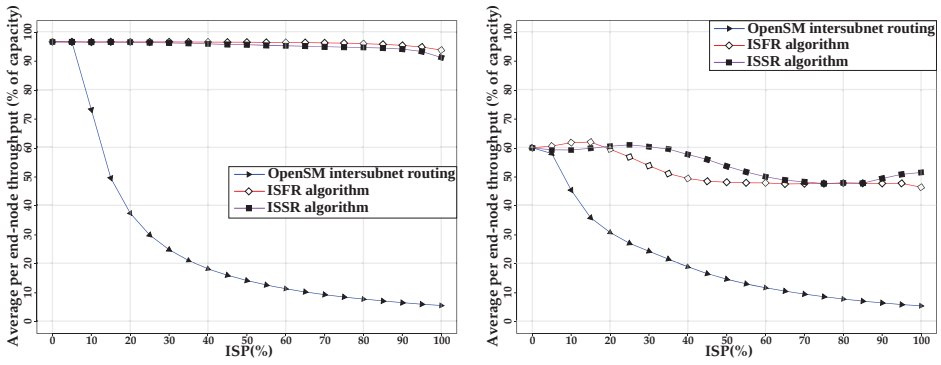
Uniform Traffic. The results for this scenario are shown in Fig. 2. When it comes to uniform traffic, we can establish that the performance of the OpenSM inter-subnet routing deteriorates in the presence of even a very small amount of inter-subnet traffic. At ISP equal to 20%, throughput is reduced to 17.5% for the 2-subnet scenario in Fig. 2(a), 23.46% for the 3-subnet scenario in Fig. 2(c), and 37.5% for the 6-subnet scenario in Fig. 2(e). The increase in performance for a larger number of subnets is explained by the fact that traffic is spread across more routers, i.e. each subnet in the topology uses a different ingress port locally. ISFR algorithm provides almost constant high performance under uniform traffic conditions whereas the performance of ISSR algorithm deteriorates slightly for very high ISP values as shown in Fig. 2(a) and Fig. 2(c). This is caused by the fact that egress ports from the routers may not be unique as they are chosen randomly. However, the deterioration is smaller when the number of subnets increases (12% decrease for the 3-subnet scenario compared to 5% decrease for the 6-subnet scenario at ISP=100%) as shown in Fig. 2(e). This occurs because the more subnets we have in the fabric and the higher is the ISP value, the more inter-subnet traffic pairs are created, so the hash function has a higher probability to utilize more links from the defined subnet-port-set as there are more random numbers generated.



(a) 2-subnet scenario uniform traffic (b) 2-subnet scenario non-uniform traffic



(c) 3-subnet scenario uniform traffic (d) 3-subnet scenario non-uniform traffic



(e) 6-subnet scenario uniform traffic (f) 6-subnet scenario non-uniform traffic

Fig. 2. Throughput as a function of ISP with uniform and non-uniform traffic

Non-uniform Traffic. Whereas under uniform traffic our algorithms gave almost optimal performance, the situation worsens if some non-uniformity is added as seen in Fig. 2.

What is first noticeable is the fact that ISFR algorithm is clearly outperformed by the ISSR algorithm for the middle range of the ISP values (20% to 70%), as

best seen in Fig. 2(d). Second, we observe that ISSR algorithm deteriorates for ISP values greater than 40%, which is best seen in Fig. 2(f). ISFR, on the other hand, becomes stable at ISP values close to 40%. This behavior is explained by the addition of the hot spots to our traffic pattern, the occurrence of head-of-line (HOL) blocking, and the migration of the root and the branches of the congestion trees. For lower ISP values ($\leq$50%) local traffic is dominant and in every subnet 5% of such traffic is destined to the local hot spot. In such a case the branches of the congestion tree will mostly influence the local traffic, but they will also grow through the single dedicated downward link (a thick branch) to influence the victim nodes in other subnets if the ISFR algorithm is used. For ISSR, the same will happen, but there is no dedicated downward link and the branches growing through multiple downward links will be much thinner, therefore, influencing the local traffic in other subnets to a lesser extent. This happens because ISSR spreads the traffic destined towards the hot spot across multiple downward links. This is the reason why ISFR algorithm is outperformed by ISSR in such a hot spot scenario for almost all ISP values.

For ISFR algorithm, for higher ISP values ($\geq$50%), the root of the congestion tree will move from the last link towards the destination to the first downward dedicated link towards the destination (i.e. a router port), and the congestion tree will influence mostly the inter-subnet traffic as there will be little or no local traffic, which is why ISFR algorithm reaches stability at around 50% ISP. For ISSR, for higher ISP values, the root of the congestion tree will not move and the branches will grow much thicker (as there is more incoming remote traffic). This will not only slightly influence the local traffic (that is low for high ISP values) in the contributor's subnets, but also it will cause HOL blocking for the downward traffic that uses the same links that the hot spot traffic uses to reach other destinations. It happens because ISSR does not use dedicated paths for downward destinations. Such a deterioration can be best observed in Fig. 2(d) or Fig. 2(f).

Another vital observation is the fact that by increasing the number of subnets, we increase the performance of all the routing algorithms. This is best visible when comparing Fig. 2(d) and Fig. 2(f). The explanation for that is the segmentation of the hot spot contributors. In other words, the more hot spots there are, the weaker is the influence of the head-of-line blocking (the congestion tree branches are thinner).

We also see that OpenSM routing still yields undesirable performance for every scenario. However, an important observation here is that the congestion does not originate from the hot spots, but from the utilization of a single ingress link to transmit the traffic to the other subnet.

HPC Challenge Benchmark. We implemented a ping-pong traffic pattern that was used to run the HPC Challenge Benchmark [18] tests in the simulator. We used a message size of 1954KB and kept the load constant at 100%. The tests were performed on 500 ring patterns: one natural-ordered ring (NOR) and 499 random-ordered rings (ROR) from which the minimum, maximum and average results were taken. In this test each node sends a message to its left neighbor in the ring and receives a message from its right neighbor. Next, it sends a message

Table 1. The HPC Challenge Benchmark results (in MB/s)

Measurement	OpenSM	ISSR	ISFR
NOR BW	1572.49	1572.49	1572.49
	ROR BW min/max/avg		
2 subnets	528/878/703	847/1166/1001	1064/1314/1187
3 subnets	345/611/482	753/993/867	946/1165/1069
6 subnets	202/343/270	709/875/775	841/1018/933

back to its right neighbor and receives a return message from its left neighbor. We treated the whole fabric as a continuous ring and we disregarded the subnet boundaries.

Table 1 presents the HPCC Benchmark results. For any fat-tree the NOR bandwidth results give the maximum throughput as there is no contention in the upward or the downward direction. However, when we compare the results for the ROR, we observe differences between the routing algorithms. For the 2-subnet scenario, we observe an increase in throughput of 536 MB/s (102%) when comparing the minimum throughput for the OpenSM and the ISFR algorithms. Furthermore, we observe that the average throughput for the ISFR algorithm is higher than the maximum throughput for the ISSR algorithm in all cases. For the average throughput we observe an increase of 484 MB/s (69%) compared to OpenSM routing. For the 3-subnet scenario the trend is the same as for the previous scenario, but we observe that the throughput is lower than for the 2-subnet scenario. This happens because the larger the topology, the higher the probability that the destination is chosen from a set of non-directly connected nodes. For a 144-node fabric, each source can address 143 end-nodes and 23 out of those end-nodes (15.7%) are reachable through a non-blocking path (11 at the local switch, 12 at the neighbor switch). For a 216-node fabric, the same number of nodes is reachable through a non-blocking path, but the overall number of nodes is larger, which gives only 10.6% of nodes reachable through a non-blocking path. This means that a ROR pattern in a larger fabric has a lower probability for reaching a randomly chosen node in a non-blocking manner. Furthermore, in larger topologies more nodes are non-local, which means that the routing algorithm uses the longest hop path to reach them (traversing all stages in a fat-tree), which further decreases the performance. For the 6-subnet scenario, we observe a similar situation as for the 3-subnet scenario: that there is an overall decrease in performance. The explanation is the same as for the 3-subnet scenario: more nodes are used to construct a ROR, but the number of nodes accessible in a non-blocking manner stays the same, so the generated ROR pairs have an even lower probability to use a non-blocking path.

The general observation is that for the HPCC benchmark, the ISFR algorithm delivers the best performance. It is because this traffic pattern does not create any destination hot spots and the congestion occurs only on the upward links towards the routers, while the dedicated downward paths are congestion-free. Despite using the same upward path as the ISFR algorithm, ISSR algorithm may

not provide a dedicated downward path, which is why there is a performance difference between these two algorithms.

6 Conclusions and Future Work

Native IB-IB routers will make the network scalable, and designing efficient routing algorithms is the first step towards that goal. In this paper, we laid the groundwork for layer-3 routing in IB and we presented two new routing algorithms for inter-subnet routing: the inter-subnet source routing and the inter-subnet fat-tree routing. We showed that they dramatically improve the network performance compared to the current OpenSM inter-subnet routing.

In future, we plan to generalize our solution to be able to support many different regular fabrics in a deadlock-free manner. Another candidate for research will be evaluating the hardware design alternatives. Looking further ahead, we will also propose a deadlock-free all-to-all switch-to-switch routing algorithm.

References

1. Obsidian Strategics: Native InfiniBand Routing (2008), http://www.nsc.liu.se/nsc08/pres/southwell.pdf
2. Southwell, D.: Next Generation Subnet Manager - BGFC. In: Proceedings of HPC Advisory Council Switzerland Conference 2012 (2012)
3. InfiniBand Trade Association: Introduction to InfiniBand for End Users (2010)
4. Obsidian Strategics: Native InfiniBand Routing (2006), http://www.obsidianresearch.com/archives/2006/Mellanox_Obsidian_SC06_handout_0.2.pdf
5. The OpenFabrics Alliance: Issues for Exascale, Scalability, and Resilience (2010)
6. Yousif, M.: Security Enhancement in InfiniBand Architecture. In: 19th IEEE International Parallel and Distributed Processing Symposium, pp. 105a. IEEE (April 2005)
7. InfiniBand Trade Association: Infiniband Architecture Specification, 1.2.1 edn. (November 2007)
8. Prescott, C., Taylor, C.: Comparative Performance Analysis of Obsidian Longbow InfiniBand Range-Extension Technology (2007)
9. Richling, S., Kredel, H., Hau, S., Kruse, H.G.: A long-distance InfiniBand interconnection between two clusters in production use. In: State of the Practice Reports on - SC 2011. ACM Press, New York (2011)
10. Top 500 Supercomputer Sites (November 2012), http://top500.org/
11. Dally, W.J., Towles, B.: Principles and practices of interconnection networks. Morgan Kaufmann (2004)
12. Duato, J., Yalamanchili, S., Ni, L.: Interconnection Networks an Engineering Approach. Morgan Kaufmann (2003)
13. Gran, E.G., Reinemo, S.A.: Infiniband congestion control, modelling and validation. In: 4th International ICST Conference on Simulation Tools and Techniques (SIMUTools 2011, OMNeT++ 2011 Workshop) (2011)
14. Oracle Corporation: Sun Datacenter InfiniBand Switch 648, http://www.oracle.com/us/products/servers-storage/networking/infiniband/046267.pdf

15. Mellanox Technologies: Voltaire Grid Director 4700, http://www.voltaire.com/assets/files/Datasheets3/Grid-Director-4700-DS-WEB-020711.pdf
16. Mellanox Technologies: IS5600 - 648-port InfiniBand Chassis Switch, http://www.mellanox.com/related-docs/prod_ib_switch_systems/IS5600.pdf
17. Forschungszentrum Jülich: JuRoPA - Jülich Research on Petaflop Architectures
18. Luszczek, P., Dongarra, J.J., Koester, D., Rabenseifner, R., Lucas, B., Kepner, J., McCalpin, J., Bailey, D., Takahashi, D.: Introduction to the HPC Challenge Benchmark Suite (April 2005)

Paper VI

Multi-homed Fat-Tree Routing with InfiniBand

Bartosz Bogdański, Sven-Arne Reinemo, Bjørn Dag Johnsen

Multi-homed fat-tree routing with InfiniBand

Bartosz Bogdański, Bjørn Dag Johnsen
Oracle Corporation
Oslo, Norway
bartosz.bogdanski@oracle.com
bjorn-dag.johnsen@oracle.com

Sven-Arne Reinemo
Simula Research Laboratory
Lysaker, Norway
svenar@simula.no

***Abstract*—For clusters where the topology consists of a fat-tree or more than one fat-tree combined into one subnet, there are several properties that the routing algorithms should support, beyond what exists today. One of the missing properties is that current fat-tree routing algorithm does not guarantee that each port on a multi-homed node is routed through redundant spines, even if these ports are connected to redundant leaves. As a consequence, in case of a spine failure, there is a small window where the node is unreachable until the subnet manager has rerouted to another spine.**

In this paper, we discuss the need for independent routes for multi-homed nodes in fat-trees by providing real-life examples when a single point of failure leads to complete outage of a multi-port node. We present and implement methods that may be used to alleviate this problem and perform simulations that demonstrate improvements in performance, scalability, availability and predictability of InfiniBand fat-tree topologies. We show that our methods not only increase the performance by up to 52.6%, but also, and more importantly, that there is no downtime associated with spine switch failure.

I. Introduction

The fat-tree topology is one of the most common topologies for high performance computing clusters today where, for example, it is used in the currently fastest supercomputer in world - MilkyWay-2 [1]. Moreover, for clusters based on InfiniBand (IB) technology the fat-tree is the dominating topology. Fat-tree IB systems include large installations such as Stampede, TGCC Curie and SuperMUC [2]. There are three properties that make fat-trees the topology of choice for high performance interconnects: deadlock freedom, the use of a tree structure makes it possible to route fat-trees without special considerations for deadlock avoidance; inherent fault-tolerance, the existence of multiple paths between individual source destination pairs makes it easier to handle network faults; full bisection bandwidth, the network can sustain full speed communication between the two halves of the network.

For fat-trees, as with most other topologies, the routing algorithm is crucial for efficient use of the underlying topology. The popularity of fat-trees in the last decade led to many efforts to improve their routing performance. These proposals, however, have several limitations when it comes to flexibility and scalability. This also includes the current approach that the OpenFabrics Enterprise Distribution (OFED) [3], the de facto standard for InfiniBand system software, is based on [4], [5]. One problem is the static routing used by IB technology that limits the exploitation of the path diversity in fat-trees as pointed out by Hoefler et al. in [6]. Another problem with the current routing is its shortcomings when routing oversubscribed fat-trees as addressed by Rodriguez et al. in [7]. A third problem, and the one that we are analysing in this paper, is that for hosts with two or more host-channel adapter ports connected to the same fat-tree-based subnet, the routing algorithm does not guarantee that independent (relative to single points of failure) root switches are chosen for the corresponding down paths.

In this paper, we discuss the need for independent routes for multi-homed nodes in fat-trees by providing real-life examples when a single point of failure led to fatal consequences. We present and implement a modified fat-tree routing algorithm that may be used to alleviate this problem and perform experiments and simulations that demonstrate the usefulness of our approach. We decided to name our routing algorithm mFtree for *multi-homed fat-tree* routing. There are two key aspects that the routing algorithm addresses:

- It identifies the paths that need to be routed in a mutually redundant way.
- It ensures that the paths are in fact redundant.

The rest of this paper is organized as follows: we discuss related work in Section II and follow with introducing the InfiniBand Architecture and fat-tree routing in Section III. We continue with a discussion of our enhancements in Section IV. Next, we describe the experimental setup in Section V followed by the experimental analysis in Section VI. Finally, we conclude in Section VII.

II. Related Work

There was much research done in the topics of fat-tree routing, multipathing, dynamic reconfiguration and fault-tolerance. First of all, there are proposals [4] and proprietary implementations of adaptive routing algorithms available [8] that extend IB's destination routing capabilities such that traffic directed to a given endpoint can traverse different paths through the network. However, the presented adaptive routing is reactive (loss of throughput during the adaptation phase), not deterministic, is only available for new switch units (if some switches do not support adaptive routing, it leads to overall slowdown of the adaptive routing manager)

and is not yet widely available. Additionally, it does not give any guarantee that the chosen paths will be mutually independent and some transport layers like IB Reliable Connected (RC) cannot be routed in an adaptive way due to the possibility of out of-order packet delivery [9].

Next, there were proposals to use other routing algorithms [10]–[13] to achieve multipathing. However, when these algorithms are applied to regular topologies like fat-trees, they may not take all the properties of a regular topology into account and cannot deliver optimal performance, and, again, there is no guarantee that a multi-homed host can be reached using independent paths.

Furthermore, there were proposals to add LID Mask Control (LMC) support to fat-tree routing [14], however, these attempts failed because the Open Subnet Manager (OpenSM) still does not support LMC for fat-tree routing. Moreover, LMC is not a solution in a multi-homed environment and it does not guarantee that the chosen path will use independent switches.

Later, there was research done on multipath routing for Extended Generalized Fat-Trees (XGFTs) [15]. However, it was shown that XGFTs cannot be used to represent many real-life topologies [9] and the aim of that work is purely theoretical as the authors failed to consider how real enterprise systems are built.

Lastly, there were several proposals to use oblivious routing in fat-tree systems [5], [7], [16], [17]. The most successful one is [5], which is the default fat-tree routing for OpenSM. These routing algorithms assume that there are enough links in a fat-tree to reconfigure the routing without much performance loss in case of failures. However, all of these algorithms work on the port level, that is, they treat each port as an independent node and do not consider the extra fault-tolerance possibilities that are provided by the additional ports at the same node. We will show that using such an oblivious routing algorithm may lead to a total lack of connectivity even in cases where a node is multi-homed.

Unlike previous research on IB routing algorithms, we discuss and analyse various enterprise fat-trees where nodes are multi-homed, i.e. a single node is connected to two or more parts of the fat-tree through multiple ports. We focus on real-life enterprise systems where fault-tolerance, reachability and performance are of the utmost importance. Our work is partially based on [18], [19] where we also analysed fat-trees, however, we did not focus on the multi-homing aspect. In this work we widen the scope and, first, consider the enterprise fat-trees with multi-homed nodes and, second, propose and evaluate a fault-tolerant routing algorithm.

III. Technical Background

The InfiniBand Architecture (IBA) supports a two-layer topological division. At the lower layer, IB networks are referred to as subnets, where a subnet consists of a set of hosts interconnected using switches and point-to-point links. At the higher level, an IB fabric constitutes one or more subnets, which are interconnected using routers. Hosts and switches within a subnet are addressed using LIDs and a single subnet is limited to 49151 LIDs. Whereas LIDs are the local addresses valid only within a subnet, each IB device also has a 64-bit *Global Unique IDentifier* (GUID) burned into its non-volatile memory. A GUID is used to form a GID - an IB layer-3 address. A GID is created by concatenating a 64-bit subnet ID with the 64-bit GUID to form an IPv6-like 128-bit address. In this paper, we will focus on port GUIDs, i.e. the GUIDs assigned to every port connected to the IB fabric.

A. Subnet Management

Every IB subnet requires at least one subnet manager (SM), which is responsible for initializing and bringing up the network, including the configuration of all the IB ports residing on switches, routers, and host channel adapters (HCAs) in the subnet. At the time of initialization the SM starts in the *discovering state* where it does a sweep of the network in order to discover all switches and hosts. During this phase it will also find other SMs and negotiate who should be the master SM. When this phase is completed the SM enters the *master state*. In this state, it proceeds with LID assignment, switch configuration, routing table calculations and deployment, and port configuration. At this point the subnet is up and ready for use. After the subnet has been configured, the SM is responsible for monitoring the network for changes.

B. Fat-Tree Routing

A major part of the SM's responsibility is routing table calculations. Routing of the network aims at obtaining full connectivity, deadlock freedom, and proper load balancing between all source and destination pairs in the local subnet. Routing tables must be calculated at network initialization time and this process must be repeated whenever the topology changes in order to update the routing tables and ensure optimal performance.

In case of the fat-tree routing algorithm, the routing function iterates over an array of all leaf switches. When a leaf switch is selected, for each end-node port connected to that switch (in port numbering sequence), the routing function routes towards that node. All of that is performed by $route\_to\_cns$ function, for which a pseudocode is presented in Algo. 1. When routing the particular LIDs, the function goes up one level to route the downgoing paths, and next, on each switch port, it goes down to route the upgoing paths. This process is repeated until the root switch level is reached. After that the paths towards all nodes are routed and inserted into the linear forwarding tables (LFTs) of all switches in the fabric. The function route_downgoing_by_going_up() is a recurrence function whose main task is to balance the paths and call the route_upgoing_by_going_down function, which routes the upward paths in the fat-tree towards destination through the switch from which it was invoked.

Algorithm 1 $route\_to\_cns()$

Require: Addressing is completed
Ensure: All $hca\ ports$ are routed
1: **for** $sw_{leaf} = 0$ **to** max_leaf_sw **do**
2: **for** $sw_{leaf}.port = 0$ **to** max_ports **do**
3: $hca\_lid = sw_{leaf}.port- > remote\_lid$
4: $sw_{leaf}.routing\_table[hca\_lid] = sw_{leaf}.port$
5: $route\_downgoing\_by\_going\_up()$
6: **end for**
7: **end for**

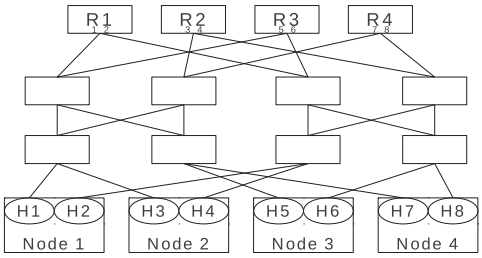


Figure 1: $H1$ and $H2$ are routed through a non-redundant path

There are several problems with the design of $route\_to\_cns()$ function. First, it is oblivious and routes the end-ports without any consideration as to which node they belong. Second, it depends on the physical port number for routing. This is a major problem, and a possible consequence is presented on Fig. 1 which depicts a scenario with four two-port nodes connected to a small fat-tree. In this case, ports $H1$, $H2$, $H5$ and $H6$ are connected to port 1 on each switch while the rest of the ports ($H3$, $H4$, $H7$ and $H8$) are connected to port 2 on each switch. The downward routes are presented for each root switch. Because the current fat-tree routing routes on leaf switch basis, after routing 4 end-ports (traversing through the second leaf switch in this case), it will wrap around and start assigning the paths again from the leftmost root switch. Therefore, each pair of end-ports ($H1$ and $H2$, $H3$ and $H4$, $H5$ and $H6$, and $H7$ and $H8$) will be routed through the same root switch. This is a very popular scenario that provides the user with counter-intuitive behaviour: a node that has built-in physical fault-tolerance (2 end-ports connected to different switches) has a single point of failure that is one of the root switches. There are many variations of this problem and, depending on the physical cabling, the single point of failure may be located on any switch in a fat-tree.

C. Subnet Reconfiguration

During normal operation the SM performs periodic *light sweeps* of the network to check for topology changes. If a change is discovered during a light sweep or if a message (trap) signalling a network change is received by the SM, it will reconfigure the network according to the changes discovered. The reconfiguration includes the steps used during initialization. Whenever the network changes (e.g. a link goes down, a device is added, or a link is removed) the SM must reconfigure the network accordingly. Reconfigurations have a local scope, i.e. they are limited to the subnets in which the changes occurred, which means that segmenting a large fabric with routers limits the reconfiguration scope.

IB is a lossless networking technology, and under certain conditions it may be prone to *deadlocks* [20], [21]. Deadlocks occur because network resources such as buffers or channels are shared and because IB is a lossless network technology, i.e. packet drops are usually not allowed. The necessary condition for a deadlock to happen is the creation of a cyclic credit dependency. This does not mean that when a cyclic credit dependency is present, there will always be a deadlock, but it makes the deadlock occurrence possible.

IV. Motivation and Design

A. Motivation

There are three main reasons that motivated us to create a multi-homed fat-tree routing and all come from real-world experience that we gained when designing and working with enterprise IB fabrics.

First, as it was mentioned before, the current fat-tree routing is oblivious whether ports belong to the same node or not. This makes the routing depend on the cabling and may be very misleading to the fabric administrator, especially when recabling is not possible due to a fixed cable length as often happens in enterprise IB systems. Furthermore, this requires the fabric designers to connect the end-nodes in such a way that will make the fat-tree routing algorithm route them through independent paths. Whereas this is a simple task for very small fabrics, when a fabric grows, it quickly becomes infeasible. Additionally, the scheme will break in case of any failure as usually the first thing that the maintenance does when a cable or a port does not work, is to reconnect the cable to another port, which changes the routing.

Second, due to the bandwidth limitations of the 8x PCI Express 2.0, it does not make sense to utilize both HCA ports on the same node with QDR speeds. This means that one of the ports is the active port and the second one is a passive port. The passive port will start sending and receiving traffic only if the original active port fails. Only with the advent of 8x PCI Express 3.0, where the bandwidth is doubled compared to 8x PCI Express 2.0, two QDR ports may be used simultaneously on the same card, which allows all ports connected to the fabric to be in an active state. These hardware limitations (active-passive case) mean that for today's routing only the active ports are important when routing and balancing the traffic because the passive ports do not generate anything (apart from management packets). The classic fat-tree routing algorithm deals with such a situation in an oblivious manner. The assumption here is that every port connected to the fabric is independent and has the same

priority when being routed and what matters is the switch number and switch port number to which the node port is connected. Multi-homed fat-tree routing algorithm is not oblivious in such a case and first routes the active port on each node, therefore making sure that the path for that port will be optimal.

Third, because of doubled bandwidth of PCI Express 3.0, all ports at all nodes need to be considered with equal weights when doing routing. Classic fat-tree routing is able to do that, since it is the original assumption, however, it does not provide for any fault-tolerance that comes from the fact that each node has two or more ports. It means that even though the node ports are connected to different leaves, their paths may meet higher in the fabric and a single point of failure may exist. Therefore, to obtain optimal fault-tolerance, it is necessary to remove the single point of failure by making sure that ports belonging to the same node take mutually independent paths, which is what mFtree routing does. What needs to be taken into account is to distinguish between a single point of failure that is always fatal for an end-port (local link, local leaf switch) and a single point of failure that can eventually be recovered from by rerouting. In other words, we wanted to ensure that no single point of failure will impact both redundant paths even for a very short time before the SM has been able to reroute and reconfigure.

B. Design

As mentioned earlier, to obtain multi-homed routing in fat-tree topologies, one must abandon the fat-tree routing algorithm that is optimized for shift all-to-all communication patterns. In other words, a new routing logic is needed to make the routing algorithm less oblivious than the classic fat-tree routing and keep the same level of determinism.

The sample pseudocode for the auxiliary function $route\_multihomed\_cns()$ is presented in Algo. 2 and Algo. 3 presents the code for the main function $route\_hcas$. There were also changes done in $route\_downgoing\_by\_going\_up()$ that are presented in Algo.4. The code for auxiliary functions is presented, so that the exact flow of the algorithm can be analysed.

Algorithm 2 $route\_multihomed\_cns()$

Require: Addressing is completed
Ensure: All hca_ports are routed through independent spines

```
for sw_leaf = 0 to leaf_sw_num do
    for sw_leaf.port = 0 to max_ports do
        hca_node = sw_leaf.port-> remote_node
        if hca_node.routed == true then
            continue
        end if
        route_hcas(hca_node)
    end for
end for
```

As seen on Algo. 2, there are many similarities when compared to the classic fat-tree routing. An iteration is done over all leaf switches and then over all leaf switch ports, so the routing can be deterministic. However, the first major difference is that mFtree routing does not simply take the LID of the remote port connected to the leaf switch, but uses the whole node as a parameter to the main routing, which is presented on Algo. 3.

Algorithm 3 $route\_hcas(hca)$

Require: Node that is to be routed
Ensure: All hca_ports belonging to the node with hca_lid are routed

```
for hca_node.port = 0 to port_num do
    hca_lid = hca_node.port-> lid
    sw_leaf = hca_node.port-> remote_node
    sw_leaf.port = hca_node.port-> remote_port_number
    sw_leaf.routing_table[hca_lid] = sw_leaf.port
    route_downgoing_by_going_up()
end for
hca_node.routed = true
clear_redundant_flag()
```

Having the end-node, we iterate over all its ports, and route each port in a usual way using the $route\_downgoing\_by\_going\_up()$ function. When all ports on the node are routed, we mark the node as $routed$, so that when that node is encountered on another leaf switch, it is not routed. For 2-port nodes, this saves half of the loop iterations.

A major change was done to the algorithm that selects the next-hop upward switch in $route\_downgoing\_by\_going\_up()$ function. Normally, the port group with lowest downward counters is selected, but in the case of mFtree algorithm, redundancy is considered to be the decisive factor. Upward node can only be chosen as the next-hop if it does not route any other ports belonging to that end-node (in other words, the $redundant$ flag is true). The redundant flag is cleared before the next end-node is routed. If there are no nodes that are redundant as may happen in heavily oversubscribed fabrics or in case of link failures, mFtree falls back to normal fat-tree routing.

We may observe the similarity of this routing function to the routing function presented in Algo. 1.

V. Experiment setup

To evaluate our proposal we have used a combination of simulations and measurements on a small IB cluster. In the following subsections, we present the hardware and software configuration used in our experiments.

A. Hardware Setup

Our test bed consisted of four two-port nodes and six switches. Each node is a Sun Fire X2200 M2 server [22]

Algorithm 4 $route\_downgoing\_by\_going\_up())$

Require: Current hop switch
Ensure: Best effort is done to find an upward redundant switch
Ensure: Switches on the path are marked with a redundant flag

```
1: group_min = 0
2: redundant_group = 0
3: for port_group = 0 to port_group_num do
4:   if group_min == 0 then
5:     if group_min -> remote_node.redundant then
6:       group_min = port_group
7:     end if
8:   else if port_group.cnt_down < group_min.cnt_down then
9:     group_min = port_group
10:    if group_min -> remote_node.redundant then
11:      min_redundant_group = group_min
12:    end if
13:  end if
14: end for
15: if group_min == 0 then
16:   fallback_normal_routing(hca_lid)
17: else if group_min -> remote_node.redundant then
18:   group_min = min_redundant_group
19:   group_min -> remote_node.redundant = false
20: end if
```

that has a dual port Mellanox ConnectX DDR HCA with an 8x PCIe 1.1 interface, one dual core AMD Opteron 2210 CPU, and 2GB of RAM. The switches were: two 36-port Sun Datacenter InfiniBand Switch 36 [23] QDR switches which acted as the fat-tree roots; two 36-port Mellanox Infiniscale-IV QDR switches [24], and two 24-port SilverStorm 9024 DDR switches [25], all of which acted as leaves with the nodes connected to them. The port speed between the QDR switches was configured to be 4x DDR, so the requirement for the constant bisectional bandwidth in the fat-tree was assured. The cluster was running the Rocks Cluster Distribution 5.3 with kernel version 2.6.18-164.6.1.el5-x86_64, and the IB subnet was managed using a modified version of OpenSM 3.2.6 (with and without our mFtree implementation). The topology on which we performed the measurements is shown in Fig. 2. We used *perftest*, a tool provided with OFED, to measure the downtime that occurred when one of the spine switches went down. *Perftest* was modified to support regular bandwidth reporting and continuous sending of traffic at full link capacity.

B. Simulation Setup

To perform large-scale evaluations and verify the scalability of our proposal, we use an InfiniBand model for the OMNEST simulator [26] (OMNEST is a commercial version

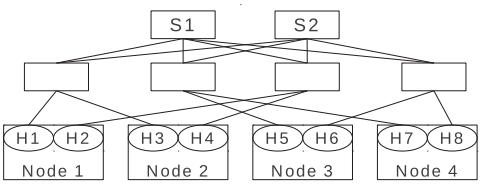


Figure 2: Experimental cluster

of the OMNeT++ simulator). The IB model consists of a set of simple and compound modules to simulate an IB network with support for the IB flow control scheme, arbitration over multiple virtual lanes, congestion control, and routing using linear routing tables. The model supports instances of HCAs, switches and routers with linear routing tables. The network topology and the local routing tables were generated using OpenSM coupled with two InfiniBand simulators: ibsim and IBMgtSim, and converted into OMNEST readable format in order to simulate real-world systems.

In each of the simulations, we used a link speed of 20 Gbit/s (4x DDR) and Maximum Transfer Unit (MTU) equal to 2048 bytes. Furthermore, we use uniform and non-uniform (shift and recursive-doubling) traffic patterns. For the uniform traffic pattern, each source randomly chooses a destination from the list of available destinations. This may lead to a situation, where more than one source chooses the same destination at the same time, which may cause slight congestion. The probability of this happening is inversely proportional to the number of end-ports, and in large fabrics is quite low. For all simulations, the end-nodes are injecting variable length packets to the network using the full capacity of the link.

The non-uniform traffic patterns can represent collective communications and are named shift and recursive-doubling. The patterns were simulated by translating their algorithm into sequences of destinations specific for each end-port and their implementation was described in a paper by Zahavi [9]. A random node-ordering of the MPI node-number to cluster end-ports was used in the translation and during the simulation, the end-ports progress through their destinations sequence independently when the previous message has been sent to the wire, which is one of the features of the IB model implemented in OMNEST simulator.

C. Topologies

We simulated five topologies of varying size. There were two 648-port fat-trees: a 2-stage one built with 18 spines and 36 leaves, and a 3-stage one built with 18 spines, 36 middle-stage switches and 27 2-switch modules (54 separate leaf switches). Each switch in a 2-switch module has 36 ports: 12 ports connected to end-nodes, 12 horizontal links connected to the neighbouring switch in the module and 12 uplinks to the middle-stage switches. Next, the 3-stage 432-port and 216-port topologies are variations of the 3-stage

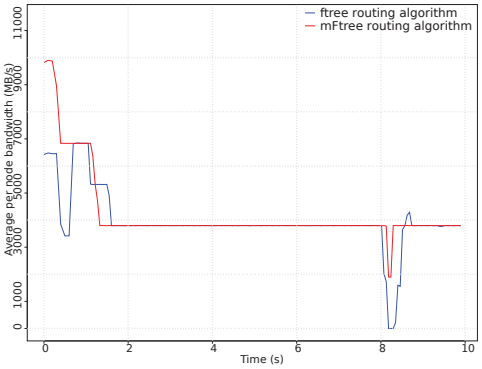


Figure 3: Results from the hardware cluster

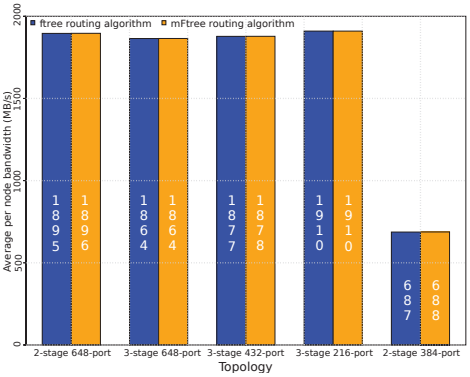


Figure 4: Test results for uniform traffic

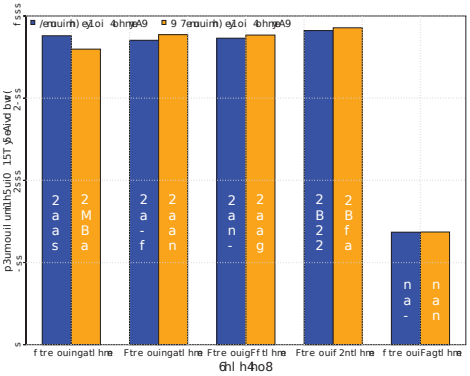


Figure 5: Test results shift traffic pattern

648-port fabric. The 432-port and 216-port fabrics are 2/3 and 1/3 of size the original 3-stage 648-port fabric, respectively. Last, the 384-port fabric is 2-stage oversubscribed fat-tree that consists of 8 spines and 16 leaf switches. The oversubscription ratio is 1.33:1.

VI. Performance Evaluation

Our performance evaluation consists of measurements on an experimental cluster and simulations of large-scale topologies. For the cluster measurements we use the *average per node throughput* and *time* as our main metrics. However, we are not comparing the network performance here, but the network downtime is of main interest. For the simulations we use the *achieved average throughput per end node* as the metric for measuring the performance of the mFtree algorithm on the simulated topologies. In both the experimental cluster and in the simulations, all traffic flows are started at the same time and they are based on transport layer of the IB stack.

A. Experimental results

The main aim of the experiment was to show that applications experience less downtime with mFtree than with the classic fat-tree routing algorithm [5]. In this experiment we rebooted the switch *S2* and measured how much time it took before the application started sending the traffic through the backup path. The whole experiment was conducted with two flows present in the network: port *H4* was sending traffic to port *H2* and *H3* was sending traffic to port *H1*. For classic fat-tree routing, both flows: $H4 \rightarrow H1$ and $H3 \rightarrow H2$ were routed through *S1* whereas for mFtree the flow $H4 \rightarrow H1$ was routed through *S1* and flow $H3 \rightarrow H2$ was routed through *S2*. In each case, we failed (rebooted) switch *S1* and measured how much time passed before normal operation was resumed.

The results are presented on Fig. 3. We observe that for mFtree routing, only half of the throughput (from 3795 MB/s down to 1897 MB/s) is lost as only one flow needs to be rerouted. For classic fat-tree routing, we observe that both flows must be rerouted, which means that even though a node receiving the traffic has two independent ports, a failure of a single switch makes the whole node unreachable.

The observation that needs to be highlighted is the fact that there is no downtime for connectivity between any pair of nodes with multiple ports operating in active-active mode when using the mFtree routing algorithm. The classic fat-tree routing engine that routes both traffic pairs through the same spine switch experiences a complete loss of service despite of the fact that the destination node has multiple ports. This means that the classic fat-tree routing algorithm is unable to use multiple ports on a node to provide for additional fault-tolerance.

B. Simulation results

The simulations were to show that using our routing algorithm does not deliver worse performance than the classic

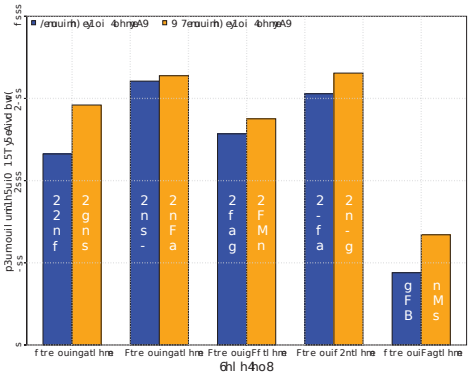


Figure 6: Test results for recursive traffic pattern

fat-tree routing algorithm when the tree size increases. We performed the simulations in an active-active mode, which means that no port was unused (passive).

The uniform traffic, for which the results are presented on Fig. 4, is considered to be very synthetic and not demanding on the routing algorithm. However, we used it to show the baseline performance as this traffic pattern has a predictable and easily understandable behaviour, and is general rather than specific to a given application. Under uniform traffic conditions, where each source chooses the destination from all other possible destinations (apart from the ports located on the same end-node), the performance of both the routing algorithms is equal. This means that mFtree routing algorithm does not create any additional overhead when routing and we do not sacrifice performance for redundancy.

The results for the shift traffic pattern shown on Fig. 5 start to exhibit slight differences between the two algorithms. The classic fat-tree routing [5], which was specifically designed for the shift traffic pattern delivers slightly better performance (on average 82 MB/s or 4.5% higher per node) only on a 2-stage 648-port fat-tree. However, for any 3-stage fabric, it is the mFtree routing algorithm that delivers slightly better performance (ranging from 0.8% to 1.8% on average per node). For the oversubscribed 384-port fabric the performance delivered by mFtree and classic fat-tree is equal. The slight performance increase that is observed for mFtree is caused by better traffic distribution while traversing middle-stage switches in the upward direction. The 3-stage fabrics are built in such a way that each pair of interconnected leaf switches share four middle-stage switches. The end-nodes are connected to the fabric in such a way that end-port:A is connected to the first leaf switch and the end-port:B is connected to the second leaf switch. What happens is that for the classic fat-tree routing, when routing in the upward direction, the traffic to end-port:A

Table I: Execution time comparison for different algorithms in seconds.

Topology	minhop	DFSSSP	ftree	mFtree
2-stage 384-port	5.12	5.04	0.02	0.02
2-stage 648-port	10.96	10.86	0.14	0.12
3-stage 216-port	5.38	5.14	0.08	0.02
3-stage 432-port	11.04	11.72	0.18	0.16
3-stage 648-port	16.5	19.38	0.38	0.36
3-stage 3456-port	105.86	286.38	11.66	10.58
4-stage 4608-port	401.1	1671.54	59.96	81.04

and end-port:B may traverse the same switch and even use the same link, which leads to slight congestion. For mFtree routing this does not happen as the traffic for end-port:A and end-port:B is separated from each other from the very beginning, which means that the traffic distribution is better. However, the influence of this phenomenon is minor because such a traffic overlap does not occur often for the shift traffic pattern we generated.

The doubling recursive pattern, for which the results are shown on Fig. 6, is the most demanding one from the patterns we tested, and the differences in performance here are significant. What is first noticeable is that the overall performance is lower for both algorithms. However, when we directly compare classic fat-tree routing and mFtree, we will observe that mFtree delivers better performance in each case. The differences are especially visible for 2-stage fabrics: for the 648-port fabric, the performance delivered by mFtree routing is 25.6% higher than for the classic fat-tree routing and for the 384-port fabric, the performance is 52.6% higher. For the 3-stage fabrics, the performance differences are 2%, 7.1%, 8.2% for the 648-port, 432-port and 216-port fabrics, respectively. Such performance differences are explained by the fact that the doubling recursive pattern is constructed in such a way that end-port:A at every end-node only sends to end-ports:A on other nodes. Furthermore, both $Node1 : portA$ when sending to $Node2 : portA$ (due to regularity of the classic fat-tree routing) uses second left-most spine as does $Node1 : portB$ when sending to $Node2 : portB$. For mFtree routing, the spines chosen will be always different. This is especially important in the oversubscribed fabric where the congestion not only occurs on the upward links, but also on some of the downward links.

C. Execution time

Another important metric of a routing algorithm is its execution time, which is directly related to the reconfiguration time of the fabric in case of topology changes. For comparison, we added two more routing algorithms: minimum-hop (minhop) and deadlock-free single-source shortest-path routing (DFSSSP) [13], and added two large topologies to observe how does the execution time scale with regard to the node number: a 3-stage 3456-port fat-tree and a 4-stage dual-

core 4608-port fat-tree. The results are presented in Table I. As we may observe, the fat-tree routing [5] (abbreviated ftree in in Table I) and mFtree routing are comparable when it comes to the execution time. For a smaller number of ports (up to 4608), mFtree is in fact slightly faster than classic fat-tree. This happens because the ratio of ports to switches is low (i.e. there are many more ports than switches). However, the problem encountered when routing a 4608-port topology is that it contains 1440 24-port switches. What is not optimized is clearing the switch redundancy flag in function $clear\_redundant\_flag()$ from Algorithm 3. The loop in that function iterates over all the switches regardless whether a particular switch was on the path or not. This could be optimized by creating a list of switches that are on the path and making sure only those switches are iterated upon.

When it comes to other routing algorithms, we observe that they have extremely long execution times for fat-tree topologies. This is especially true to DFSSSP whose execution time explodes 1671.5 seconds, which means almost 28 minutes of downtime in case of a failure.

VII. Conclusions and Future Work

In this paper, we presented the new mFtree routing algorithm. We showed that it improves the network performance compared to the current OpenSM fat-tree routing by up to 52.6%. Most importantly, however, mFtree routing algorithm gives much better redundancy than classic fat-tree routing, which means that multi-homed nodes will suffer no downtime in case of switch failures.

In future we plan to optimize the routing algorithm, so its execution time is shorter on larger fabrics. Furthermore, we will be working with other enhancements to the fat-tree routing.

References

[1] J. Dongarra, "Visit to the National University for Defense Technology Changsha, China," Report, June 2013, http://www.netlib.org/utk/people/JackDongarra/PAPERS/tianhe-2-dongarra-report.pdf.

[2] "Top 500 supercomputer sites," http://top500.org/, June 2013.

[3] "The OpenFabrics Alliance," http://openfabrics.org/.

[4] C. Gómez, F. Gilabert, M. E. Gómez, P. López, and J. Duato, "Deterministic versus Adaptive Routing in Fat-Trees." [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.96.5710

[5] E. Zahavi, G. Johnson, D. J. Kerbyson, and M. Lang, "Optimized Infiniband fat-tree routing for shift all-to-all communication patterns," in *Concurrency and Computation: Practice and Experience*, 2009.

[6] T. Hoefler, T. Schneider, and A. Lumsdaine, "Multistage switches are not crossbars: Effects of static routing in high-performance networks," in *Cluster Computing, 2008 IEEE International Conference on*, 29 2008-Oct. 1 2008, pp. 116–125.

[7] G. Rodriguez, C. Minkenberg, R. Beivide, and R. P. Luijten, "Oblivious Routing Schemes in Extended Generalized Fat Tree Networks," *IEEE International Conference on Cluster Computing and Workshops*, 2009.

[8] "InfiniBand in the Enterprise Data Center," White paper, Mellanox Technologies, April 2006, http://www.mellanox.com/pdf/whitepapers/InfiniBand__EDS.pdf.

[9] E. Zahavi, "Fat-trees routing and node ordering providing contention free traffic for mpi global collectives," in *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, may 2011, pp. 761 –770.

[10] P. Lopez, J. Flich, and J. Duato, "Deadlock-free routing in infiniband/sup tm/ through destination renaming," *Parallel Processing, International Conference on, 2001.*, pp. 427–434, 3-7 Sept. 2001.

[11] J. Sancho, A.Robles, J. Flich, P.Lopez, and J. Duato, "Effective methodology for deadlock-free minimal routing in infiniband networks," in *Proceedings of the 2002 International Conference on Parallel Processing*. IEEE Computer Society, August 2002, pp. 409–418.

[12] J. C. Sancho, A. Robles, and J. Duato, "Effective strategy to compute forwarding tables for infiniband networks," in *ICPP*, L. M. Ni and M. Valero, Eds. IEEE Computer Society, 2001, pp. 48–60.

[13] J. Domke, T. Hoefler, and W. Nagel, "Deadlock-Free Oblivious Routing for Arbitrary Topologies," in *Proceedings of the 25th IEEE International Parallel and Distributed Processing Symposium*. IEEE Computer Society, May 2011, pp. 613–624.

[14] X.-Y. Lin, Y.-C. Chung, and T.-Y. Huang, "A Multiple LID Routing Scheme for Fat-Tree-Based Infiniband Networks," *Proceedings of IEEE International Parallel and Distributed Processing Symposiums*, 2004.

[15] S. Mahapatra, X. Yuan, and W. Nienaber, "Limited multi-path routing on extended generalized fat-trees," in *Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*, ser. IPDPSW '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 938–945. [Online]. Available: http://dx.doi.org/10.1109/IPDPSW.2012.115

[16] X. Yuan, W. Nienaber, Z. Duan, and R. Melhem, "Oblivious routing for fat-tree based system area networks with uncertain traffic demands," in *Proceedings of the 2007 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, ser. SIGMETRICS '07. New York, NY, USA: ACM, 2007, pp. 337–348. [Online]. Available: http://doi.acm.org/10.1145/1254882.1254922

[17] S. Öhring, M. Ibel, S. Das, and M. Kumar, "On Generalized Fat Trees," in *Proceedings 9th International Parallel Processing Symposium*, 1995, pp. 37–44.

[18] B. Bogdanski, F. O. Sem-Jacobsen, S.-A. Reinemo, T. Skeie, L. Holen, and L. P. Huse, "Achieving predictable high performance in imbalanced fat trees," in *Proceedings of the 16th IEEE International Conference on Parallel and Distributed Systems*, X. Huang, Ed. IEEE Computer Society, 2010, pp. 381–388.

[19] B. Bogdanski, B. D. Johnsen, S.-A. Reinemo, and F. O. Sem-Jacobsen, "Discovery and routing of degraded fat-trees," in *2012 13th International Conference on Parallel and Distributed Computing, Applications and Technologies*, H. Shen, Y. Sang, Y. Li, D. Qian, and A. Y. Zomaya, Eds. Los Alamitos: IEEE Computer Society, December 2012, pp. 689–694.

[20] W. J. Dally and B. Towles, *Principles and practices of interconnection networks*. Morgan Kaufmann, 2004.

[21] J. Duato, S. Yalamanchili, and L. Ni, *Interconnection Networks An Engineering Approach*. Morgan Kaufmann, 2003.

[22] "Sun Fire X2200 M2 server," Oracle Corporation, November 2006, http://www.sun.com/servers/x64/x2200/.

[23] "Sun Datacenter InfiniBand Switch 36," Oracle Corporation, http://www.sun.com/products/networking/datacenter/ds36/.

[24] "MTS3600 36-port 20 Gb/s and 40Gb/s InfiniBand Switch System," Product brief, Mellanox Technologies, 2009, http://www.mellanox.com/related-docs/prod_ib_switch_systems/PB_MTS3600.pdf.

[25] "SilverStorm 9024 Switch," Qlogic, http://www.qlogic.com/Resources/Documents/DataSheets/Switches/Edge_Fabric_Switches_datasheet.pdf.

[26] E. G. Gran and S.-A. Reinemo, "Infiniband congestion control, modelling and validation," in *4th International ICST Conference on Simulation Tools and Techniques (SIMUTools2011, OMNeT++ 2011 Workshop)*, 2011.