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 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

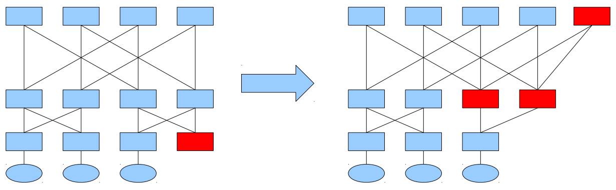
Figure 1: Flipping switch in a simple 3-stage fat-tree.

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.
1: if $switch\ has\ no\ CNs$ then
2:     if $smpquery(switch,\ role)\ ==\ leaf$ then
3:         $switch.rank = tree\_rank$
4:     end if
5: 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

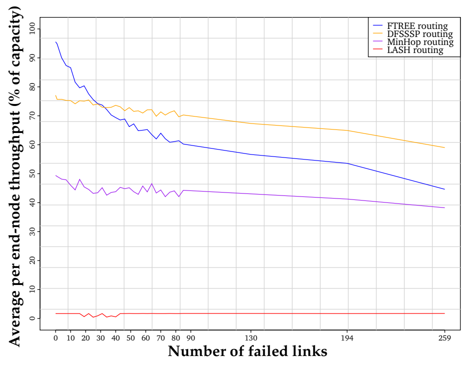

Figure 2: Comparing the algorithms for uniform traffic and failing links

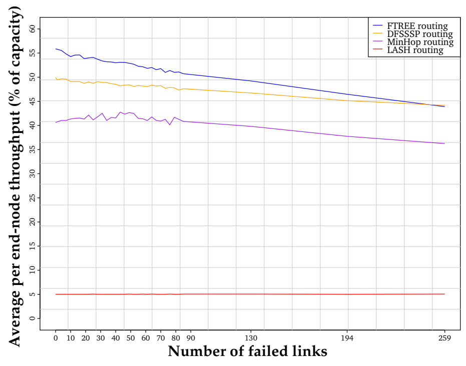

Figure 4: Random Ring - Average results for failing links scenario

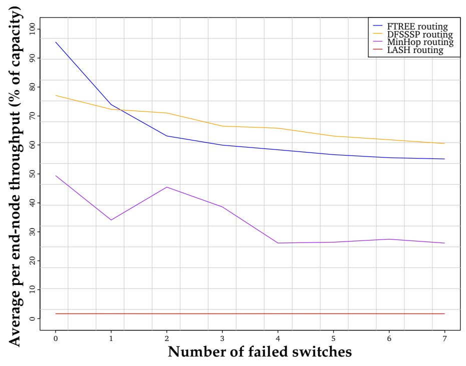

Figure 3: Comparing the algorithms for uniform traffic and failing switches

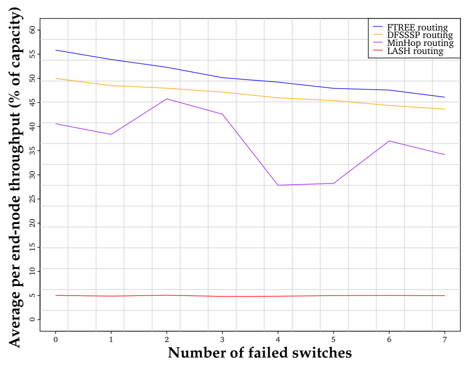

Figure 5: Random Ring - Average results for failing switches scenario

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.