Quasi Fat Trees for HPC Clouds and Their Fault-Resilient Closed-Form Routing

Eitan Zahavi
Mellanox
eitan@mellanox.com

Isaac Keslassy
Technion
isaac@ee.technion.ac.il

Avinoam Kolodny
Technion
kolodny@ee.technion.ac.il

***Abstract*—High-Performance Computing (HPC) Clusters and Data Center Networks often rely on fat-tree topologies. However, fat trees and their known variants are not designed for concurrent small jobs. As a result, in recent years, HPC designers have introduced ad-hoc topologies to offer better performance for these concurrent small jobs.**

In this paper, we present and formally define these topologies, which we call Quasi Fat Trees (QFTs). Specifically, we formulate the graph structure of these new topologies, and show that they perform better for concurrent small jobs. Furthermore, we derive a closed-form and fault-resilient contention-free routing algorithm for all global shift permutations. This routing optimizes the run-time of large computing jobs that utilize MPI collectives. Finally, we verify the algorithm by running its implementation as an OpenSM routing engine on various sizes of QFT topologies, and show that it exhibits good performance.

I. Introduction

A. Outline

High Performance Computing (HPC) clusters are built to conduct large-scale computations. While they used to be a scarce resource, they have evolved to become a standard utility for researchers and engineers. In the cloud era, HPC Clouds are built to concurrently run many distributed jobs. They often rely on a *fat-tree topology* (or some fat-tree variant), because of its good path-diversity and scalability properties.

Traditionally, fat trees were designed to provide the best performance *for the most critical large jobs running on the entire cluster*. Furthermore, routing algorithms were written to provide the optimal network performance to these jobs. As a result, many past works deal with the problem of maximizing the performance of a single job, and in particular attempt to optimize the topology, routing, traffic patterns and communication algorithms - known as collectives. New interconnection networks, transports and protocols were invented for that end [1]–[6].

However, in recent years, significant new problems have emerged from *the sharing of these clusters by multiple jobs*. Unfortunately, to our knowledge, *no work in the literature on fat trees has suggested a new topology to deal with many concurrent small jobs*. A few recent studies have considered the extra runtime that multiple concurrent jobs impose on each other due to network contention [7]. In particular, utilizing different buffering resources to reduce this impact has been proposed in [8]. However, fat-tree topology considerations have surprisingly been absent from the literature.

On the other hand, to offer better performance for these concurrent small jobs, state-of-the-art HPC clusters and Data Center Networks (DCNs) have increasingly adopted an emerging ad-hoc fat-tree variant topology. It has become a dominant network topology in the industry, yet has not yet been documented in the literature and does not even have a formal name and general definition. For instance, the National Center for Atmospheric Research (NCAR) Yellowstone [9] and NCAP Tide [10] are two clusters built in 2013 that rely on this topology. This topology has a reduced effective network diameter for smaller jobs and may also increase separation between jobs. Note that this benefit is achieved without any increase in the number of cables or switches.

Unfortunately, not only this new topology lacks a formal definition, but more significantly, *there are no known contention-free routing algorithms for this topology*. This is especially significant in the cases where it carries the traffic of full-cluster-size MPI (Message Passing Interface) collectives [11]. Hence, while small concurrent jobs achieve better performance running on this topology, the performance of single large jobs may be degraded.

In this paper, we formally define this emerging fat-tree-like topology, which we denote as Quasi Fat Tree (QFT). We also formulate a closed-form contention-free routing for all global shift permutations. Furthermore, we present a fault-tolerant version of the routing algorithm, which can also extend to fat trees. Finally, we show that QFT with contention-free routing for global collectives outperforms other fat-tree topologies, because it improves the performance for many small jobs, without degrading the performance of a single large job.

B. Fat-tree topologies

The fat-tree topology is a well-defined graph structure that was formulated in [12] and extended in [13]. The latter formulation, denoted as *fat tree* in this paper, is more flexible to describe topologies of more sizes, as it allows each level of the tree to have a different number of connections to upper and lower levels.

In order to reduce system cost, most fat-tree implementations are built from many instances of the switch device that provides the best cost/performance trade-off at the time of cluster implementation. To reduce the number of switches, the topology should also utilize all the available switch ports. Therefore, the fat-tree formulation, which requires each switch port to connect to a different switch, can only describe a single maximal topology if all switches have the same number

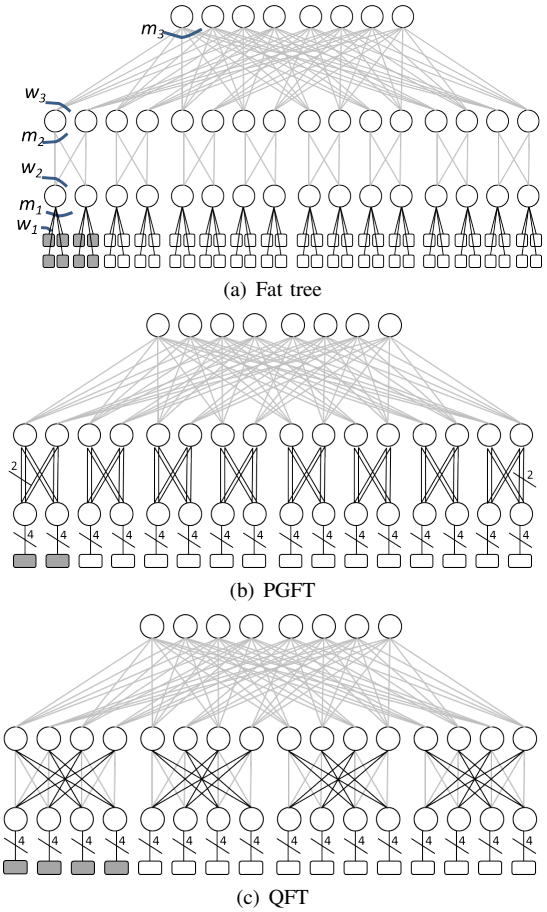

Fig. 1. Fat-tree, PGFT and QFT topologies with 64 hosts and 40 switches of 8 ports. (a) A non-maximal fat tree, thus leaving 2 unused ports in middle- and lower-level switches (b) PGFT utilizing the free ports by adding parallel links between the low and middle level switches; (c) QFT where the leaf switches connect the unused ports to different switches. As a result, the size of host groups that are within a 3-switch distance is doubled (as denoted by the shadowed hosts).

of ports. On the contrary, as illustrated in Figure 1(a), in a non-maximal fat-tree topology, some switches necessarily have unused ports.

To fully use all the switch ports in clusters that are smaller than the maximum, it is possible to rely on a generalized family of fat trees, called Parallel Ports Generalized Fat Trees (PGFTs) [11]. While at most one link can connect two switches in the original fat trees, multiple parallel links can connect a pair of switches in PGFTs. As a result, PGFTs enable many possible topologies of diverse sizes, and they are used in many HPC cluster installations. Figure 1(b) illustrates a PGFT topology with 64 hosts connected using 8-port switches. We can see how each of the 16 bottom switches uses four ports to connect to four hosts, and four ports to connect to two middle switches, i.e. two parallel ports per middle switch.

QFT is a different modification of fat tree which connects the extra links, available on non-maximal topologies, to as many switches as possible. Intuitively, by branching out, QFT attempts to maximize the number of hosts that can communicate at full link bandwidth *within a short distance* (a formal definition is given in Section III). Figure 1(c) shows

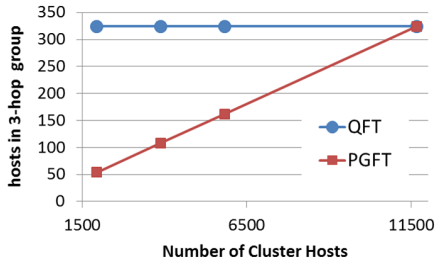

Fig. 2. Number of hosts in a 3-hop group versus cluster size. While the PGFT group size is proportional to the cluster size, the QFT group size depends only on the number of ports in a switch.

an example of QFT topology, given the same parameters as the above PGFT topology. We can see how each of the 16 bottom switches now connects to 4 middle-switches instead of 2. As a result, we observe that in this QFT topology, the number of hosts that are within 3 switches of a given host (denoted as its *3-hops group*) is twice the number in the PGFT topology, i.e., 16 versus 8 hosts. In fact, unlike the fat tree, the QFT is not a graph that can be represented as a collection of trees, because of the additional connections.

Thus, QFT reduces the effective network diameter for smaller jobs that fit within the 3-hops groups (which is a well-known figure of merit for a network topology [14], [15]). For real-world practical cluster sizes, the ratio of hosts in 3-hops groups in QFT over PGFT is commonly 2x, but may be as large as 6x when using 36-port switches, or even reach 12x for 48-port switches.

C. QFT Advantages

Before delving into the QFT formulation, let's first build some intuition for why cluster designers implement QFT when dealing with many concurrent jobs.

The improved performance of QFT for many concurrent jobs stems from the reduced effective job diameter and the reduced probability of contention with traffic from other jobs. Figure 2 illustrates the number of hosts in 3-hops groups as a function of the total cluster size for 3-level fat trees made of 36-port switches. We can see how the QFT group size is independent of the cluster size, and is at least double the PGFT 3-hops group size for topologies that are smaller than the maximal possible cluster using those switches. The PGFT group size is proportional to the cluster size. Ideally, jobs that fit within a single 3-hops group could be placed within such a network partition, and their traffic only needs to traverse the links within that sub-tree. However, job schedulers have to trade off between *contiguous job placement* and *cluster utilization*. High cluster utilization is known to fragment jobs into multiple contiguous placement ranges [16].

Figure 3 compares the performance of PGFT and QFT. In order to compare the job performance under a realistic job placement, jobs are randomly placed on simulated QFT and PGFT clusters of 4536 hosts. To simulate the effect of job-placement fragmentation, the jobs are placed on multiple contiguous-host ranges (fragments) of sizes that follow a normal distribution of 36 hosts on average, and a standard deviation of 10 (restricted of course to positive sizes).

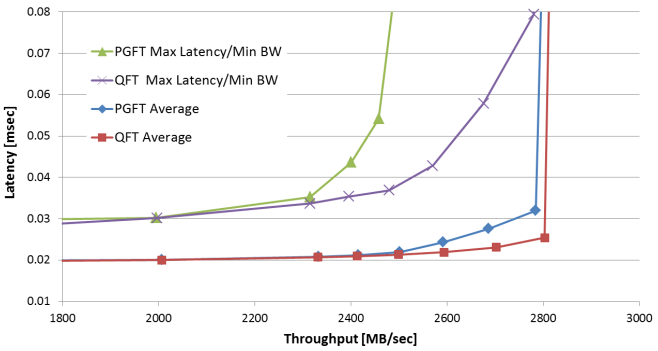

Fig. 3. Simulation results showing latency vs. throughput curves, parameterized by offered load, for PGFT and QFT. The simulated cluster has 4536 hosts and runs 12 concurrent jobs. Jobs are placed in contiguous-placement fragments of sizes distributed as a truncated Normal(36,10).

The average job size is 300 hosts. The jobs perform MPI collectives, thus a random set of shift-permutation traffic is simulated (see the evaluation section for more details about the simulator). Figure 3 plots the latency-versus-throughput curves of PGFT and QFT. Specifically, it plots both the average-throughput/average-latency as well as the max-latency/min-bandwidth curves; the former is important to asynchronous and the latter to synchronous MPI communication schemes. As can be observed in both cases, the QFT throughput saturation value is higher.

D. Contention-free routing

While the above example shows why QFT can be efficient for *many concurrent jobs*, QFT also needs to efficiently handle a *single large full-network job*, like other fat-tree architectures.

Network contention is defined as the case where two or more data flows are forwarded through the same network link. Such network contention results in excessive network delays; therefore, a network contention-less routing scheme significantly reduces run-time for MPI applications [2]. Hence it is desirable to avoid contention caused by MPI jobs traffic. Many researchers have worked on optimizing the collective algorithms and adapting them to the topology [3]–[6]. In particular, we are interested in contention freedom for *shift permutation* patterns. These shift traffic patterns are created by most MPI communication library APIs (known as collectives) and thus govern the traffic of many HPC applications [11]. Other routing schemes, which improve the network performance of random-permutation traffic, were proposed by [19]. However, in these algorithms the output port depends on both the source and the destination, thus require routing-table resources that are much higher than those required by destination-based routing.

A closed-form destination-based routing algorithm (where the output port is defined by an algebraic function of the destination) is already known in basic fat trees for switches with a number of ports that is a power of two [17], [18], and for an arbitrary number of ports [11]. This routing, known as d-mod-k, carries shift permutations[1] without any network contention. With d-mod-k routing, a single large job can be run on the entire cluster with absolutely zero network contention as long as its traffic utilizes MPI collectives. However, since QFT breaks some of the basic properties of fat trees, a different routing algorithm is required for contention-free traffic.

Note that a *closed-form* routing scheme has lower computational complexity compared to other routing algorithms that require graph traversal (e.g. BFS [2] or DFS [1]). Moreover, with a closed-form routing, the forwarding tables can be directly computed in parallel. In this paper, we will look for a closed-form contention-free routing for QFT.

The rest of the paper is organized as follows: The model for the fat-tree topology is defined in Section II and for QFT in Section III. QFT contention-free routing is presented in Section IV and the fault resilient extension in Section V. Finally QFT routing is evaluated in Section VI.

II. Fat-Tree Formulation and Properties

To define a closed-form routing in a topology, we first need to formally describe it and characterize its properties. This section describes the well-known formulation and properties of fat-tree topologies. They are then extended in the next section to support QFT topologies as well.

There are several ways to define fat trees [20], [21]. It is useful to start with an extended generalized fat-tree definition, which is simply denoted as *fat tree* in this paper. As illustrated in Figure 1(a), we follow the notation $FT(h; m_1, m_2.., m_h; w_1, w_2.., w_h)$ of [13]. In this notation, h represents the number of switch levels of the tree. Tree levels are numbered from 0 at the bottom, where hosts are connected, to h at the top. The m_l coefficients define the number of children a switch at level l has, and w_l denote the number of parent-switches a switch at level $l-1$ has. For instance, w_1 defines the number of parents of each host.

As an example, in Figure 1(a), the topology is $FT(3; 4, 2, 8; 1, 2, 8)$. The sequence $m_1, m_2, m_3 = 4, 2, 8$ represents the number of children each switch at levels 1, 2 and 3 has. Similarly the sequence $w_1, w_2, w_3 = 1, 2, 8$ represents the respective number of parents of the hosts, the switches at level 1, and the switches at level 2.

Based on the above notation, the following properties of fat trees were defined by [13]. The number of switches or hosts at level l is:

$$\left(\prod_{i=1}^{l} w_i\right)\left(\prod_{i=l+1}^{h} m_i\right) \tag{1}$$

We further denote each node S in the tree using the tuple $(l_s, s_h, ..., s_2, s_1)$. The tuple includes the tree level l_s of the node (i.e., switch or host), and a set of h digits $s_h, ..., s_2, s_1$ that represent a multi-radix number [22] describing the relative order of switches or hosts within that level. This number is also termed the *index* of the node within its level.

Tuple meaning: For a switch at level l the digits $s_h, \ldots, s_{l+1}$ represent the shortest path from the top of the tree to that node:

$$\forall i \in \{l+1, \ldots, h\} : s_i \in \{0, \ldots, m_i - 1\} \tag{2}$$

The rest of the digits $s_l, ..., s_1$ represent the shortest path from the bottom of the tree to that node.

$$\forall i \in \{1, \ldots, l\} : s_i \in \{0, \ldots, w_i - 1\} \tag{3}$$

[1] Of a specific set of job sizes

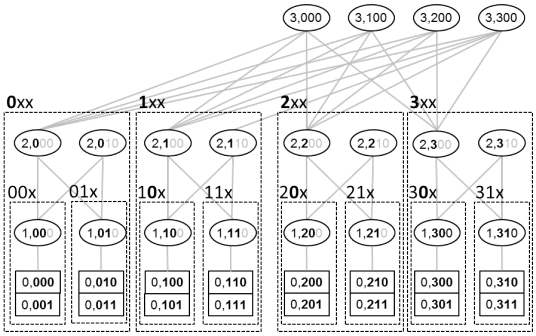

Fig. 4. Tuple assignment of fat tree (for clarity, only half of the tree is shown). The most significant digits are assigned by the top-to-bottom sub-tree they belong to. These digits are bolded in the drawing. The least significant digits (represented in gray) are assigned according to the bottom-to-top "flipped" trees.

The index d of a host with tuple $(0, s_h, ..., s_2, s_1)$ is calculated as a multi-radix number; each digit represents a value that equals the product of the radix of all previous digits:

$$d = \sum_{l=1}^{h} \left(s_l \prod_{i=1}^{l-1} m_i \right) \tag{4}$$

For example, consider the host tuples on the fat tree of Figure 4. The radix (i.e., the number of different digits) is 2 for s_1 ($m_1 = 2$) and 2 for s_2 ($m_2 = 2$). Thus the index of the host $(0; 2, 0, 1)$ can be calculated as $1 + 0 \cdot 2 + 2 \cdot (2 \cdot 2) = 9$.

Tree construction: Based on the above notation a fat tree may be constructed by first instantiating all the tree nodes, assigning them unique tuples, and then connecting them according to the following fat-tree connection rule: *node* $S(l_s, s_h, ..., s_2, s_1)$ *connects via a bi-directional link to node* $Q(l_q, q_h, ..., q_2, q_1)$ *iff:*

$$l_s = l_q + 1 \wedge \forall i \in \{1...h\} : i \neq l_s \implies s_i = q_i \tag{5}$$

This connection rule leads to some of the key properties of fat trees:

Sub-Tree Rule: If two nodes are within the same sub-tree of height h' they must share the same $h - h'$ most significant digits. This is true since the l^{th} digit may change only on links connecting level l to level $l - 1$. This also means that considering just the paths from top to bottom, the topology is a collection of such trees (since no merging of branches is possible).

Flipped-Sub-Tree Rule: By "flipped", we mean that we are considering the fat tree from bottom to top. If two nodes are within the same sub-tree, of height k they must share the same k least significant digits. This also means that no merging of the flipped-sub-trees is possible.

Level Orthogonality in Path Selection [23]: Consider a path from one host to another. Then, for each level of the tree, selecting the up-going port on a switch on the way up, defines the up-port through which the path will go back down through a switch on the same level.

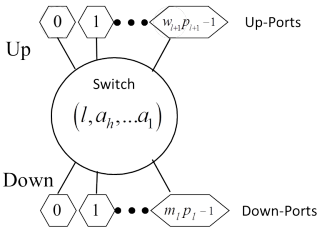

Fig. 5. The extension of switch object with its connected ports. $w_{l+1}p_{l+1}$ up-ports and $m_l p_l$ down-ports connecting to upper and lower tree levels, respectively.

III. Quasi Fat Trees

The QFT topology is defined by the notation $QFT(h; m_1, m_2..., m_h; w_1, w_2..., w_h; p_1, p_2..., p_h)$. The h, m_i and w_i have the same meaning as in a fat tree. The p_i represents the number of cross-connections, including both the original cross-connections and those that were added to the fat tree. They connect switches of levels i and $i - 1$ that are adjacent but belong to different sub-trees. The total number of switches and hosts is exactly the same as in a fat tree, and thus a similar construction procedure can be performed to create and assign tuples to all nodes. The new connection rule defines these extra p_i connections.

While the above formulation supports multiple levels of cross connections, all the known QFT clusters in the industry are only cross-connected at the lowest level (i.e., $p_2 > 1$ and $p_{i>2} = 1$). This configuration is the one that maximizes the 3-hop host groups—the main motivation behind this topology. As a result, in this paper, we assume for simplicity that the QFT topologies have a single level l where $p_l > 1$, and we also require that w_{l+2} be divisible by p_l. This condition is met by all QFT topologies maintaining cross-bisectional bandwidth and built from switches of the same number of ports.

In order to formulate a closed-form routing, i.e. a routing where we can define a function F that provides an output port as a function of the destination, we had to extend the tree formulation with a new type of object: the port object. This object enables us to define the exact port number that connects a switch to its adjacent switch.

Figure 5 illustrates the switch ports. As can be observed, there are $m_l p_l$ down-ports, connecting the switch of level l to switches at level $l - 1$. Similarly there are $w_{l+1}p_{l+1}$ up-ports, connecting the switch of level l to switches at level $l+1$. We can define the following new *QFT connection rule: a switch* $S = (l, s_h, ..., s_2, s_1)$ *down-port* f *connects to switch* $Q = (l - 1, q_h, ..., q_2, q_1)$ *up-port* g *iff:*

$$\begin{cases} (\forall i \in \{1, \ldots, l-1\} \cup \{l+2, \ldots, h\} : s_i = q_i) \ \wedge \\ \left\lfloor \frac{s_{l+1}}{p_l} \right\rfloor = \left\lfloor \frac{q_{l+1}}{p_l} \right\rfloor & l < h \\ (\forall i \in \{1, \ldots, h-2\} : s_i = q_i) \ \wedge \\ \left\lfloor \frac{s_{h-1}}{p_h} \right\rfloor = \left\lfloor \frac{q_{h-1}}{p_h} \right\rfloor & l = h \end{cases} \tag{6}$$

The connected port numbers are set accordingly as:

$$f = \begin{cases} q_l + m_l \left(q_{l+1} \mod p_l\right) & l < h \\ q_l + m_l \left(q_{l-1} \mod p_l\right) & l = h \end{cases} \tag{7}$$

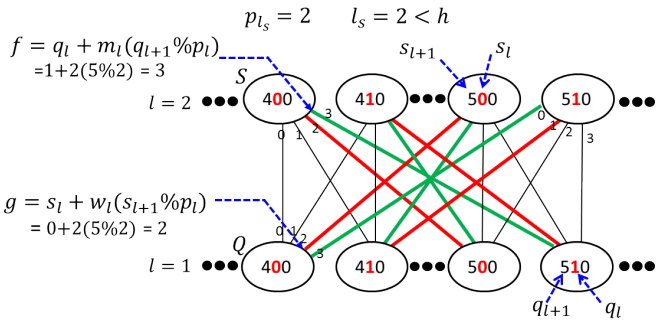

Fig. 6. Example of QFT connection rule, given a subset of a QFT in levels 1 and 2($< h$). The extra cross-connections that do not appear in the fat tree are drawn using wider lines. For instance, there are $p_2 = 2$ cross-connections from level-2 switch to level-1 switches, in addition to the 2 regular connections.

$$g = \begin{cases} s_l + w_l\,(s_{l+1} \mod p_l) & l < h \\ s_l + w_l\,(s_{l-1} \mod p_l) & l = h \end{cases} \tag{8}$$

Figure 6 illustrates the above connection rules. The emphasized connections are the new cross-connections. For this example, $p_2 = 2$, so there are 2 cross-connections.

IV. QFT Routing

We recognize the following algorithmic steps as the building blocks for QFT closed-form routing:
(i) Descendant criteria: a predicate that allows to test if a destination host is located within the sub-tree rooted by a specific switch.
(ii) Up Routing: the formula by which an up-port is assigned to a given non-descendant destination.
(iii) Down Routing: the formula by which a down-port is assigned to a given descendant destination.

As QFT extends the connections of the fat tree, known routing algorithms could also be used for it, but they would fail to utilize the added connections in QFT, and thus would suffer from contention among flows under full-permutation traffic. Therefore, we need to present new closed-form routing algorithms. In this section, we present these algorithms without proofs, as the proofs are quite long. The detailed proofs and the tuple assignment algorithm are presented in an online technical report [24].

A. Descendant Criteria

We start by defining the following *fat-tree* descendant criteria, which return *true* if for a given pair of nodes $Q(l_q, q_h, \ldots, q_2, q_1)$ and $S(l_s, s_h, \ldots, s{+}2, s_1)$, Q is a descendant of S: (a) the level of S must be higher, (b) Q must be in the sub-tree of S, and (c) S must be in the flipped-sub-tree of Q. Formally, Q is a descendant of S iff:

$$(l_s > l_q) \ \wedge \tag{9a}$$
$$(\forall i \in \{l_s + 1, \ldots, h\} : s_i = q_i) \ \wedge \tag{9b}$$
$$(\forall j \in \{1, \ldots, l_q\} : s_j = q_j) \tag{9c}$$

Note that if Q is a host and S is a switch, the descendant criteria in Equation (9) reduce to the second-line term.

In QFT, since we route from host to host, we now want to define a QFT descendant criterion when the descendant is a given host. In particular, consider a switch $S(l_s, s_h, ..., s_2, s_1)$ and a host $Q(0, q_h, ..., q_2, q_1)$. We find that the criterion is very similar to that of fat tree, as provided in Equation (9), except that we now need to take into account the number p_{l_s} of cross-connections of switch S at the level l_s. Specifically, *the descendant criteria in QFT for hosts is that host Q is descendant of switch S iff:*

$$\forall i \in \{l_s + 2..h\} : s_i = q_i \ \wedge \left\lfloor \frac{s_{l_s+1}}{p_{l_s}} \right\rfloor = \left\lfloor \frac{q_{l_s+1}}{p_{l_s}} \right\rfloor \tag{10}$$

For example, in Figure 6, host 500 in level 1 is a descendant of switch 410 of level 2, since $l_s = 2$, $p_2 = 2$ and $\lfloor \frac{4}{2} \rfloor = \lfloor \frac{5}{2} \rfloor$.

B. Routing Up

We now need to define the up-routing that is a switch-specific function mapping each non-descendant destination to one of the switch's up-going ports. The up-routing function for fat trees (and their parallel-ports flavor PGFT) has been defined in [11]. It was proved to result in contention-free routing for global shift permutations. That proof shows first that for fat trees the set of destinations passing through a switch at level l is an arithmetic sequence (wrapping up modulo the total number of hosts) with a step of:

$$R_l = \prod_{i=1}^{l} w_i \tag{11}$$

It then shows that during a global-shift permutation, the set of destinations passing through a switch is a contiguous sub-range of that sequence and thus can be mapped using a modulo operator to a disjoint set of up-ports.

In order to route without contention global-shift permutation for QFT, we use a similar approach of spreading up-going traffic through all the up-going ports.

We denote as l_c (standing for cross connections) the first level l where $p_l > 1$. Below level $l_c - 1$, QFT and fat tree are identical and thus can use same routing. However, for the level $l = l_c - 1$ there are more up-going ports to choose from ($w_l p_l$ instead of w_l). So the resulting set of destinations passing through each switch in levels $l \leq l_c$ makes an arithmetic sequence of fixed step size of

$$R'_l = \prod_{i=1}^{l} \frac{w_i p_i}{p_{i-2}} \tag{12}$$

Furthermore up-port g, routing for QFT levels $l \leq l_c$, is defined as:

$$g = \left\lfloor \frac{d}{R'_l} \right\rfloor \mod (w_{l+1} p_{l+1}) \tag{13}$$

At level $l = l_c + 1$, one level above l_c, flows that were spread over all up-going ports two levels below (producing arithmetic sequences) are now merged back to produce a set of destinations that is not an arithmetic sequence, and therefore cannot be routed by Equation (13) without contention. Based

on the QFT connection rule, there are exactly p_{l-1} such cross-connections. Each such connection delivers destinations that form an arithmetic sequence of distance R'_l. The key for providing shift permutation traffic with no contention is to spread the merged traffic, originating from the same switch 2 levels below, to different up-ports. So the algorithm first calculates for each destination the index u of the cross-connected parent that the destination was routed through 2 levels below the merge level:

$$u = \frac{\left\lfloor \frac{d}{R'_{l-2}} \right\rfloor \mod (w_{l-1}p_{l-1})}{w_{l-1}} \tag{14}$$

Intuitively, the division by w_{l-1} yields the index of the output port within the p_{l-1} cross-connection groups.

At this stage, the up-port can be calculated by placing the merged destinations on continuous ports (multiplying the $\left\lfloor \frac{d}{R'_{l-2}} \right\rfloor$ by p_{l-1}), and sequencing them by the calculated u. So the *up-port routing* for levels where $p_{l-1} > 1$ is:

$$g = \left(\left\lfloor \frac{d}{R'_l} \right\rfloor p_{l-1} + u \right) \mod (w_{l+1}p_{l+1}) \tag{15}$$

We finally can establish our main result: Equation (15), which is a generalization of Equation (13), provides a contention-free routing.

Theorem 1. *The up-going routing provided by Equation (15) yields contention freedom for global shift permutations on constant bi-sectional bandwidth QFT.*

C. Routing Down

To our knowledge, the PGFT *down-routing* has not been previously formally defined in the literature. Before we present QFT down-routing, we first introduce a simple PGFT *down-routing* formula, which we later extend for QFT.

Assume we want to send a packet from level l to level $l-1$ in a PGFT. As consecutive destinations passing through a switch are separated by R_l, and since they are located below ports $d_l + jm_l$, the down-port f for switch $Q(l, q_h, ..., q_2, q_1)$ to route to a destination $(0, d_h, ..., d_2, d_1)$ can be given by:

$$f = d_l + m_l \left(\left\lfloor \frac{d}{R_l} \right\rfloor \mod p_l \right) \tag{16}$$

In QFT, a special care should be given to the level where different cross-connections are merged, i.e., where $p_{l-1} > 1$. Only at this level a switch may have more than one path, through different switches, towards its descendants. So the question is how to spread these destinations over the multiple paths? We use the following idea: Since shift permutation flows are spread by the up-routing with no contention on the way up, sending the down-going traffic through the same switches they traverse on the way up guarantees no contention on the way down. Note that we calculated in Equation (14) the index u that the flows to destination d use within the cross-connected parents. Since the down-ports connecting to cross-connected lower-level switches are contiguous, the down-port should be the first port of that group plus u:

$$f = \left\lfloor \frac{d_l}{p_{l-1}} \right\rfloor p_{l-1} + u \tag{17}$$

Furthermore, the down-routing for levels that are not *merge* levels simply follows the QFT connection rule of Equations (9) and (10), which defines the port number connecting towards a destination:

$$f = \begin{cases} d_l & 1 = p_l \\ d_l + m_l \left(d_{l-1} \mod p_l\right) & 1 < p_l \cap l = h \\ d_l + m_l \left(d_{l+1} \mod p_l\right) & 1 < p_l \cap l < h \end{cases} \tag{18}$$

Therefore, we obtain the following result, which establishes that our down-routing is contention-free as well. Together with Theorem 1, we obtain that we have a contention-free up- and down-routing.

Theorem 2. *Equation (18) yields a contention-free down-routing for all full shift permutations on constant bi-sectional bandwidth.*

Corollary 3. *Equations (10), (15) and (18) form a closed-form, contention-free routing for constant bi-sectional bandwidth QFT for all full shift permutations.*

V. Fault-Tolerant Routing

We have provided formula-based routing algorithms for QFT. While these formulas are fast and efficient when the topology is complete, they can quickly break down when some links or switches are missing from the network. Indeed, there is no way to provide contention freedom for full permutations if some links are lost and the network does not meet the rearrangeable-non-blocking condition anymore. However, a good routing engine should still preserve connectivity and provide rudimentary load-balancing. Hence, the goal of this section is to offer a fast algorithm to deal with link failures in QFT. *Note that to our knowledge, even in fat trees, there are no such algorithms today. So we also offer the first such algorithm for fat trees.*

Our algorithm is based on the new idea that faults impact only the relevant switches at the specific level, thus making a novel use of the property of level-orthogonality in path selection, which was presented at the end of Section II. Note that [18] describes a framework for dealing with fault-tolerance in simple k-ary n-trees. However, it does not deal with non-maximal trees such as QFT. Furthermore, that paper describes a specific hardware implementation instead of providing a general algorithm.

A. Fat-Tree Fault-Tolerant Routing

We start by providing a fat-tree algorithm, and later extend it to QFTs. In a fat tree, consider a missing link between switch S at level l and switch Q at level $l-1$. We denote this link as $(l, s_h, ..., s_2, s_1, f) \Leftrightarrow (l-1, q_h, ..., q_2, q_1, g)$. Based on the level-orthogonality property, it is required to redirect all flows going up on port g and program all other switches on level $l-1$ that may be forwarding traffic through the missing link. Although the above idea could yield an algorithm that treats each failure separately, it is actually required to first calculate and maintain the list of unavailable up-ports for each switch. This is required to avoid migrating traffic to an unavailable link. (We present the full algorithm details in [24].)

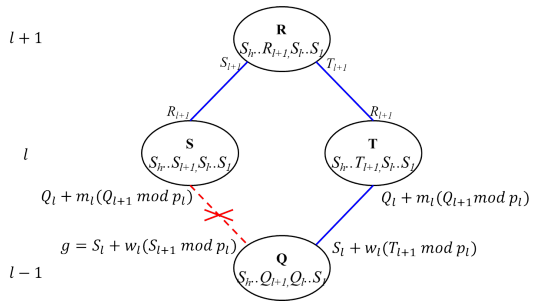

Fig. 7. A QFT sub-graph around a failing link between switches S and Q. Most of the tuple digits are related by the connection rule. So an alternate switch T can be found given the remaining available ports on the destination switch Q.

Algorithm 1 QFT fault-tolerant routing

```
Get2LevelsDownDescendantSwitch(R,d)
  find the switch Q by the following tuple:
    (l ,d_h,..d_{l-1},r_{l-2},..r_1) when l = l_r

ReRouteQFTOnFaults()
collect missing up-ports on each switch into port-mask
for each source switch R in level l and destination d
  if d is a QFT descendant of R
    if p_{l-1} > 1 (R is on a merge level)
      Q = Get2LevelsDownDescendantSwitch(R,d)
      Q_mask = MissingPortMask(Q)
      R_mask = TranslateLowerMask(Q_mask)
      select an existing port out of R_mask
    else
      nothing can be done, no alternate paths down
  else
    if (p_{l+1} = 1 or
        destination is child to same l+1 parent)
      Q = SameLevelAccessibleBySParentOfD(R,d)
      N-port-mask = Q-port-mask & R-port-mask
    else
      N-port-mask = R-port-mask
    select a random port of N-port-mask
```

B. QFT Fault-Tolerant Routing

QFTs extend fat trees with some extra links. This means that the fault-tolerant routing algorithm of fat trees is sufficient to recover routing in QFT, and therefore we could stop here. However, the additional cross-connections of QFT also provide an opportunity (not available for fat trees) to route around a missing down-link fault. The fault-tolerant routing for QFT is presented in Algorithm 1. The main deviation from the fat-tree algorithm is the section dealing with levels where multiple paths to descendants are provided by the cross-connections.

Figure 7 illustrates the use of alternate paths in a given sub-graph of QFT. To resolve faults by utilizing alternate paths on a merge level $l+1$, Algorithm 1 relies on obtaining the switch at level $l-1$ that is on the way to the destination (using the function Get2LevelsDownDescendantSwitch of Algorithm 1). The figure shows a missing link and one of its alternate links. As can be observed, most of the tuple digits of the involved switches are inter-related by the QFT connection rule. Since the destination d is a descendant of Q, the QFT descendant criterion applies. Moreover, since Q is below the level with cross-connections, the criterion just requires all digits of Q above level l to be identical to those of d. As described in Figure 7, the relations between R and Q show that the lower digits of Q are the same as those of R. Therefore:

$$Q = d_h, ..., d_{l_r-1}, r_{l_r-2}, ...r_1 \tag{19}$$

TABLE I. ROUTING ALGORITHMS RUNTIME

Topology		Routing (sec)	
Definition	Hosts	*ftree*	*qft*
PGFT(3;18,9,36;1,9,18;1,2,1)	5832	4	1
QFT(3;18,9,36;1,9,18;1,2,1)	5832	NA	1
PGFT(4;18,3,18,36;1,3,18,18;1,6,1,1)	34992	478	18
QFT(4;18,3,18,36;1,3,18,18;1,6,1,1)	34992	NA	17

VI. EVALUATION

We now want to evaluate the correctness and performance of our suggested algorithms. To do so, we encode the QFT (as well as PGFT) algorithms described in this paper in a new routing engine named *pqft*, in the *de-facto* standard InfiniBand subnet manager OpenSM. In InfiniBand networks, the subnet manager is somewhat similar to an SDN (Software-Defined Networking) controller, and is in charge of configuring the forwarding tables on all network switches.

Correctness: We start by testing the correctness of the implementation on many different 3- and 4-level PGFT and QFT topologies. Given a single job spanning the entire network, our tests verify that without faults, there is no contention for all shift permutations. In addition, we later remove links and switches to test fault-tolerance. Again, we verify that the connectivity between all hosts is maintained by the routing engine. We have run this verification on 20 different sizes of 3- and 4-level QFTs with cluster sizes from 32 nodes to 11664 nodes.

Runtime: Next, we evaluate the complexity of our routing algorithms. Specifically, we compare the runtime of the existing *ftree* [2] algorithm to our new algorithm implementation, denoted *pqft*. Table 1 illustrates the results for two cluster sizes, both for PGFT and QFT. We can see that our algorithm implementation runs faster.

Performance: We further compare the different routing algorithms by inspecting the latency-versus-throughput curves, which are drawn by changing the offered load from each host. We perform this comparison by simulating a $QFT(3;18,9,28;1,9,18;1,2,1)$ of 4536 hosts. The simulation platform is an OMNeT++ [25] based flit-level model of InfiniBand networks, including credit propagation times and switching based on virtual-output-queues [26]. This publicly-available simulation model was already used for example in [11], [27], [28]. Two types of traffic patterns are evaluated: a random sequence of shift permutation patterns, and a uniform-random destination. These traffic patterns were generated to represent the two different cluster usage models: a single job spanning the entire cluster, or multiple jobs randomly placed (in contiguous fragments). The compared routing algorithms are those available in OpenSM: $ftree, updn + scatter$, and our new qft. The $ftree$ algorithm follows [11], which fails to avoid congestion for the global shift traffic over the QFT. The $updn$ [29] routing applies strict up down turn limitation rules and avoids traffic-polarization effects by applying port-ordering randomization. The simulation uses a switch model utilizing 50KB of input buffer per port, an ability to concurrently send 3 packets from each input port, and an output

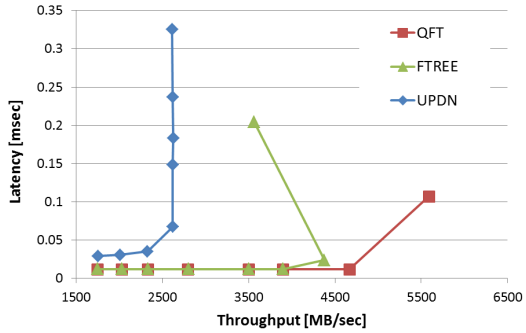

Fig. 8. Latency-versus-Throughput curve for a single job executing random shift permutations on a QFT of 4536 hosts with 56Gbps InfiniBand links. The routing algorithms $updn + scatter$, $ftree$ and qft show saturation at 2620MB/s, 4400MB/s, and the full link bandwidth 5700MB/s respectively.

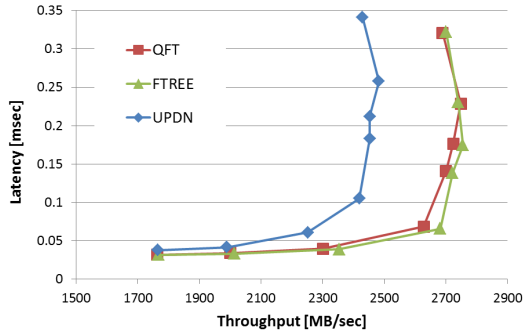

Fig. 9. Latency-versus-Throughput curve for multiple jobs placed randomly on a QFT of 4536 hosts with 56Gbps InfiniBand links. The routing algorithms $updn+scatter$, $ftree$ and qft show saturation at 2480MB/s, 2740MB/s and 2740MB/s respectively.

arbitration model that is similar to iSLIP [30] (with a single match per device clock cycle).

Figure 8 depicts the results for the single job case. As can be seen, the throughput saturates first for the *updn* algorithm at 2620MB/s, then for the *ftree* at 4400MB/s. Note how due to the lossless-network congestion spread, the average throughput is reduced as the network reaches saturation. The full bandwidth of 5.6GB/s is maintained by our new algorithm.

Figure 9 illustrates the performance of these three routing algorithms for many concurrent jobs, by applying a random-destination traffic pattern. As can be observed, both the $ftree$ and qft algorithms reach the same saturation throughput of 2740MB/s, with a slightly smaller latency at that point for $ftree$. Therefore, our evaluation shows that the new qft algorithm results in superior performance for the single job running MPI collectives, and comparable performance for many jobs placed randomly.

VII. Conclusion

In this paper, we formally defined QFT, a commonly-used cluster topology that provides advantages over PGFT for many concurrent jobs. We further introduced a formula for contention-free routing for all global shift permutations. Given our introduced closed-form QFT routing and fault-tolerance, it appears that there is no advantage to using PGFT instead of QFT when supporting a single large job. Hence, QFT should be preferred over PGFT for fat-tree computing clusters.

References

[1] T. Hoefler *et al.*, "Optimized routing for large-scale InfiniBand networks," in *17th IEEE HOTI 2009*, 2009, pp. 103–111.

[2] E. Zahavi *et al.*, "Optimized InfiniBand fat-tree routing for shift all-to-all communication patterns," *Concurrency and Computation*, 2010.

[3] B. Prisacari *et al.*, "Bandwidth-optimal all-to-all exchanges in fat tree networks," in *ICS*, 2013, pp. 139–148.

[4] K. Kandalla *et al.*, "Designing topology-aware collective communication algorithms for large scale InfiniBand clusters," in *IEEE IPDPS*, 2010.

[5] A. Mittal *et al.*, "Collective algorithms for sub-communicators," in *ICS*, 2012, pp. 225–234.

[6] H. Subramoni *et al.*, "Design and evaluation of network topology/Speed aware broadcast algorithms for InfiniBand clusters," in *CLUSTER*, 2011.

[7] A. Jokanovic *et al.*, "Impact of inter-application contention in current and future HPC systems," in *IEEE HotI*, 2010, pp. 15–24.

[8] ——, "Effective quality-of-service policy for capacity high-performance computing systems," in *HPCC-ICESS*, 2012, pp. 598–607.

[9] C. NCAR, "Yellowstone." [Online]. Available: http://www2.cisl.ucar.edu/resources/yellowstone

[10] Top500, "NCEP Tide," Jun. 2013. [Online]. Available: http://www.top500.org/system/177838

[11] E. Zahavi, "Fat-tree routing and node ordering providing contention free traffic for MPI global collectives," *JPDC*, Nov. 2012.

[12] C. E. Leiserson, "Fat-trees: universal networks for hardware-efficient supercomputing," *IEEE Trans. Comput.*, Oct. 1985.

[13] S. Ohring *et al.*, "On generalized fat trees," in *IPPS*, Santa Barbara, CA, USA, pp. 37–44.

[14] J. Duato *et al.*, *Interconnection Networks: An Engineering Approach*. Morgan Kaufmann, 2003.

[15] W. J. Dally and B. P. Towles, *Principles and Practices of Interconnection Networks*. Elsevier, Mar. 2004.

[16] J. A. Pascual *et al.*, "Job scheduling strategies for parallel processing." Springer-Verlag, 2009, p. 138156.

[17] X.-Y. Lin *et al.*, "A multiple LID routing scheme for fat-tree-based InfiniBand networks," in *IEEE IPDPS*, Apr. 2004.

[18] C. Gomez *et al.*, "Deterministic versus adaptive routing in fat-trees," in *2007 IEEE IPDPS*, Long Beach, CA, USA, Mar. 2007, pp. 1–8.

[19] W. Nienaber *et al.*, "Improving performance of deterministic single-path routing on 2-level generalized fat-trees," in *IEEE IPDPS*, 2011.

[20] M. Valerio *et al.*, "Recursively scalable fat-trees as interconnection networks," in *IEEE Intl Phoenix Cfce on Comp. and Comm.*, 1994.

[21] F. Petrini and M. Vanneschi, "k-ary n-trees: high performance networks for massively parallel architectures," in *IPPS*, 1997, pp. 87–93.

[22] J. Arndt, "Mixed radix numbers," in *Matters Computational*. Springer Berlin Heidelberg, Jan. 2011, pp. 217–231.

[23] Z. Ding *et al.*, "Level-wise scheduling algorithm for fat tree interconnection networks," in *ACM/IEEE SC*, 2006.

[24] E. Zahavi *et al.*, "Quasi fat trees formulation, fault-resilient closed-form routing," in *TR14-03*. [Online]. Available: http://technion.ac.il/~ezahavi/papers/tech_rep_14_03_qft.pdf

[25] G. Pongor, "OMNeT: objective modular network testbed," in *Proceedings MASCOTS*, ser. MASCOTS '93, San Diego, CA, USA, 1993.

[26] E. Zahavi, "Infiniband simulation macro model macro." [Online]. Available: http://www.omnetpp.org/omnetpp/doc_details/2070-infiniband

[27] E. G. Gran *et al.*, "On the relation between congestion control, switch arbitration and fairness." IEEE, May 2011, pp. 342–351.

[28] E. Zahavi *et al.*, "Distributed adaptive routing for big-data applications running on data center networks," in *ANCS*, 2012, pp. 99–110.

[29] W. J. Dally and C. L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Trans. on Comp.*, 1987.

[30] N. McKeown, "The iSLIP scheduling algorithm for input-queued switches," *IEEE/ACM ToN*, 1999.