New routing algorithms for heterogeneous exaflopic supercomputers

John Gliksberg

THESE DE DOCTORAT

NNT : 2022UPASG068

New routing algorithms for heterogeneous exaflopic supercomputers

Nouveaux algorithmes de routage pour supercalculateurs exaflopiques hétérogènes

Thèse de doctorat de l'université Paris-Saclay et de l'université de Castilla–La Mancha

École doctorale n°580 : sciences et technologies de l'information et de la communication (STIC)
Spécialité de doctorat : Informatique
Graduate School : Informatique et sciences du numérique
Référent : Université de Versailles Saint-Quentin-en-Yvelines (UVSQ)

Thèse préparée dans l'unité de recherche LI-PARAD (Université Paris-Saclay, UVSQ) ;
sous la direction de Devan Sohier, Professeur HDR,
la co-direction de Pedro Javier García García, Professeur équivalent HDR,
et le co-encadrement de Antoine Capra, Ingénieur recherche.

Thèse soutenue à Guyancourt, le 25 novembre 2022, par

John Gliksberg

Composition du jury

Membres du jury avec voix délibérative

Maria Engracia Gómez Requena Professeure des universités, UPV	Présidente
Brice Goglin Directeur de recherche, Inria Bordeaux - Sud-Ouest	Rapporteur & Examinateur
Enrique Vallejo Maître de conférence, Université de Cantabria	Rapporteur & Examinateur
Marc Pérache Directeur de recherche, CEA DAM	Examinateur
Emmanuelle Saillard Chargée de recherche, Inria Bordeaux - Sud-Ouest	Examinatrice

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Title: New routing algorithms for exaflopic heterogeneous supercomputers
Keywords: Routing, Algorithms, HPC, Interconnection networks

Abstract: Building efficient supercomputers requires optimising communications, and their exaflopic scale causes an unavoidable risk of relatively frequent failures. For a cluster with given networking capabilities and applications, performance is achieved by providing a good route for every message while minimising resource access conflicts between messages. This thesis focuses on the fat-tree family of networks, for which we define several overarching properties so as to efficiently take into account a realistic superset of this topology, while keeping a significant edge over agnostic methods. Additionally, a partially novel static congestion risk evaluation method is used to compare algorithms. A generic optimisation is presented for some applications on clusters with heterogeneous equipment. The proposed algorithms use distinct approaches to improve centralised static routing by combining computation speed, fault-resilience, and minimal congestion risk.

Titre: Nouveaux algorithmes de routage pour supercalculateurs exaflopiques hétérogènes
Mots clés: Routage, Algorithmes, HPC, Réseau d'interconnexion

Résumé: La construction de supercalculateurs performants nécessite d'optimiser les communications, et leur échelle exaflopique amène un risque inévitable de pannes relativement fréquentes. Pour un cluster avec un réseau et des équipements donnés, on améliore les performances en s'assurant que l'on sélectionne une bonne route pour chaque message tout en minimisant les conflits d'accès aux resources entre messages. Cette thèse se concentre sur la famille des réseaux fat-trees, pour laquelle nous donnons quelques grandes caractéristiques afin de mieux prendre en compte une classe réaliste de cette topologie, tout en conservant un avantage par rapport aux méthodes agnostiques. De plus, une approche d'évaluation statique partiellement nouvelle du risque de congestion est utilisée pour comparer les algorithmes. Une optimisation générique est présentée pour certaines applications sur des clusters avec des équipements hétérogènes. Les algorithmes proposés forment le résultat de plusieurs approches distinctes pour apporter des contributions dans le domaine du routage statique centralisé, en combinant rapidité de calcul, résilience aux pannes, et minimisation du risque de congestion.

Titulo: Nuevos algoritmos de encaminamiento para superordenadores heterogéneos a exaescala
Palabras clave: Encaminamiento, Algoritmos, HPC, Red de interconexíon

Resumen: La optimización de las comunicaciones es un requisito fundamental para la construcción de superordenadores eficientes. Por otra parte, la escala en el ámbito del "exaflop" de estos superordenadores conlleva un riesgo inevitable de fallos relativamente frecuentes. Para un clúster de computación con determinadas capacidades de red y aplicaciones, un buen rendimiento se consigue proporcionando rutas eficientes y minimizando los conflictos de acceso a los recursos entre los diferentes mensajes. Esta tesis se centra en la familia de topologías "fat-tree", para la que definimos varias propiedades generales considerando un superconjunto realista de esta topología, manteniendo una ventaja sustancial sobre métodos agnósticos. Además, se utiliza un método relativamente novedoso para la evaluación del riesgo de congestión estática de los diferentes algoritmos. Se presenta una optimización genérica para algunos tipos de aplicaciones en clústeres con equipos heterogéneos. Los algoritmos propuestos utilizan enfoques distintos para mejorar el encaminamiento estático centralizado combinando velocidad de cálculo, tolerancia a fallos y minimizando el riesgo de congestión.

Contents

Acknowledgements

I must give mille mercis to my director, Devan, who was key throughout the process and pushed me towards an academic rigour that can be lacking in this specific domain, and that I wasn't naturally striving to uphold more than necessary. You endeavoured to learn quite a bit about a subject that otherwise may have seemed rote and undignified, for that I am grateful and I suspect you found a few interesting surprises on the way. My co-director, Pedro, deserves heart-felt gracias as well for his continued sympathy and attention to details; never have I received such amicable praise accompanied with a thoroughly blood-drenched page of corrections. I hope to enjoy many more meals and matches in your company. Antoine, you weren't just a reviewer but also a great inspiration, and along with Alexandre I have fond memories of the three of us getting Dmodc working; thank you both. I have received quite a bit of help from the people at LI-PaRAD, and I must single out Thomas and Pablo for their time and care. I must thank both reviewers for their helpful remarks and kind words, and the rest of the jury members for their efforts. On another note, I would be remiss to forget to acknowledge the character-building challenges brought on by the administrative entities of both universities. I can only recommend the combination of industrial doctorate and international cotutelle to students and advisors of tenacious will.

I must thank my mum profusely for the numerous indispensable nitpicks throughout the manuscript. Please read it only superficially from now on, or you'll find yet more mistakes. Thank you, yiddishe papa, for driving me to the interview back in 2016 and then leaving me to work without any pressure. Thank you Daphné for your infinite patience and loving application of motivational threats. You had to bear with my occasional defeated self ever since we were together.

My colleagues at Atos have always been very motivated to see me go through this doctorate (sometimes more so than myself); and I must thank each of them, both for their work and their fraternity. Alain, Pierre, Safae, Ravaka, Ben, Zakaria, Marc, Marwa, Jean-Yvon, Bruno G, Bruno F, etc, etc, thank you. I must thank Jean-Noël and Pierre V in particular, my first two advisors at Atos. You placed all the necessary blocks for this thesis, and your back-breaking work on BXI helped me learn a lot of my trade.

The times I spent in Albacete (no hay ningún sitio mejor que Albacete) were wonderful, and the people at and around the lab were great to me. Many thanks to Rocher, German, Juanje, Jesús, Pedro Yébenes, Blas, Raúl, Antonía, Hugo, Ester, etc, etc.

Finally, I must shout out to all friends and family, who've been hearing a whole lot about this; now we'll have to find other things to talk about. Thank you, in no particular order, to Sam, Alex, Guillaume, Sarah, Vincent, Louis-Daniel, Laure, Matthieu, Marion, Simon, Léopold, Marie-Liesse, and all the others. And of course, I am eternally indebted to Patate.

Translated introduction

Introduction in French

Les supercalculateurs actuels les plus puissants construits pour le Calcul Haute-Performance (HPC) sont des grappes de matériel commercial haut-de-gamme, combinant des millions de cœurs, coûtant des dizaines de millions d'euros, et consommant plusieurs mégawatts [81] : une telle puissance de calcul demeure prohibitive en termes de coût et de consommation énergétique. Le réseau d'interconnexion (l'*interconnect*) d'un tel système permet aux applications d'être distribuées sur de nombreux nœuds pour accélérer leur temps d'exécution, ou pour augmenter leur échelle ou leur précision. Les performances globales du système dépendent fortement des performances de l'interconnect [24, 3]. En conséquence, les interconnects HPC sont prévus pour assurer une quasi-totale absence de pannes et minimiser l'utilisation de mécanismes de résilience aux pannes. Ce réseau finit donc par représenter une part conséquente de la puissance et du matériel alloués aux supercalculateurs modernes. Néanmoins, l'évolution en termes de puissance et d'échelle de ces systèmes amène un risque inévitable de pannes matérielles. Un volume important de la recherche dans le domaine vise à minimiser le coût et la consommation de ces réseaux tout en maximisant leurs performances et leur résilience aux pannes. Le routage joue un rôle vital dans l'interconnect HPC, qui constitue dans son ensemble un système compliqué dont les choix architecturaux et l'utilisation impactent le coût, les performances et la résilience aux pannes. Cette thèse se concentre sur des améliorations de techniques de routage pour interconnects HPC avec la conception et l'analyse statique de nouveaux algorithmes de routages, et des modifications applicables à des algorithmes existants. Cette décision se place à contre-courant de la majorité de la recherche contemporaine qui se concentre sur des études dynamiques d'aspects plus avancés des interconnects HPC (tels que le routage adaptatif, l'ordonnancement dynamique d'applications, ou autres optimisations dynamiques basées sur le comportement réel des applications) : une partie des soucis de performance qui justifient ces méthodes pourraient être évités en amont au niveau du routage statique. Les techniques de routage proposées offrent des améliorations de performances et/ou de résilience aux pannes, tout en relaxant partiellement les contraintes topologiques de la conception du réseau.

Cette recherche a été motivée par un contexte industriel alors que Bull construisait un interconnect interne pour ses nouveaux supercalculateurs Bull eXascale Interconnect (BXI), avec un nouveau *fabric manager* (le logiciel qui observe et configure le réseau à un haut niveau, ce qui inclut le calcul d'un routage fonctionnel et performant). Les switches BXI sont conçus avec un CPU

ARM embarqué et un réseau de gestion séparé de l'interconnect, ainsi que d'autres spécificités de conception. Pour s'adapter à ces aspects, et profiter de l'occasion de repartir de zéro, le fabric manager BXI (BXI FM) est organisé différemment des *subnet managers* de ses concurrents. Certains algorithmes et techniques de routage ont été conçus pour ce système pour utiliser pleinement les caractéristiques de switches. Pour améliorer ces techniques, et avec l'intuition que cette nouvelle architecture de matériel pouvait mener à une approche différente dans la recherche, le sujet de thèse initial a été planifié. Cette thèse est menée avec le LI-PaRAD (Laboratoire d'Informatique, Parallélisme, Réseaux, Algorithmes Distribués) à l'UVSQ (Université de Versailles Saint-Quentin-en-Yvelines). Cette thèse est également menée avec le laboratoire RAAP (Redes y Arquitecturas de Altas Prestaciones) à l'UCLM (Universidad de Castilla-la-Mancha) en Espagne, au travers d'une cotutelle internationalle. Le contexte industriel a fortement impacté le déroulement de cette thèse, lors de laquelle plusieurs inventions ont été brevetées et intégrées dans le produit commercialisé par Atos. En revanche, un effort a été fourni pour étudier des cas généraux et les résultats sont, dans l'ensemble, applicables à d'autres situations.

Introduction in Spanish

Los actuales supercomputadores empleados para la computación de altas prestaciones (High-Performance Computing, HPC) están formados por un conjunto de componentes comerciales, incluyen millones de núcleos de cómputo (cores), tienen un coste de decenas de millones de euros, y consumen algunas decenas de megavatios [81]; en resumen, puede decirse que conseguir altas potencias de cómputo sigue siendo caro, y consume muchos recursos. La red de interconexión de estos sistemas permite repartir aplicaciones entre muchos nodos del sistema para acelerar su tiempo de ejecución, o para aumentar su escala, resolución, o precisión. Las prestaciones del sistema en su conjunto dependen enormemente de la red de interconexión [24, 3]. En consecuencia, a menudo las redes de sistemas HPC se han sobredimensionado para no fallar prácticamente nunca y para minimizar los mecanismos de recuperación, pero este enfoque hace que la red represente una fracción muy relevante de los componentes y del consumo del sistema supercomputador. La importancia de este problema ha crecido con el incremento de la escala de estos sistemas (imprescindible para conseguir mayor potencia de cómputo), lo que además ha ocasionado que la aparición de fallos en los componentes sea casi inevitable. Por ello, buena parte de la investigación en este campo se orienta a minimizar el coste y consumo de estas redes, mientras se intenta optimizar sus prestaciones y su resiliencia ante posibles fallos. En este entorno, el encaminamiento es un factor esencial en el funcionamiento de la red, cuyo diseño es clave de cara al coste, las prestaciones y la resiliencia. Esta tesis se centra en mejorar el

encaminamiento para redes de sistemas HPC mediante el diseño y análisis de nuevos algoritmos estáticos de encaminamiento, y mediante extensiones de otros ya existentes. Este enfoque se basa en la observación de que una gran parte de la investigación actual en redes de interconexión para sistemas HPC se orienta hacia estudios de técnicas "dinámicas" tales como encaminamiento adaptativo, planificación dinámica de procesos, y optimizaciones basadas en trazas, mientras que muchas de las prestaciones que se pretenden conseguir con estas técnicas podrían abordarse mejor de antemano mediante algoritmos de encaminamiento estáticos. Las propuestas de esta tesis ofrecen mejoras bien respecto a prestaciones, bien respecto a resiliencia ante fallos, o respecto a ambas, relajando a la vez ciertas restricciones del diseño de la topología.

La investigación recogida en esta tesis se inició en un contexto industrial, al estar Bull diseñando su nueva tecnología de red de interconexión propia, llamada BXI (Bull eXascale Interconnect), junto con un nuevo software (*fabric manager*) para monitorizar y configurar la red a alto nivel, incluyendo el mantenimiento funcional y el encaminamiento eficiente. Los conmutadores BXI integran una CPU de ARM, y se conectados a una red de control y administración paralela y separada de la red de interconexión del sistema supercomputador. Para adaptarse a este entorno, y también para partir de cero, sin ceñirse a modelos previos, el software de control BXI se organiza de forma distinta a los equivalentes en otras tecnologías de red. Algunas técnicas y algoritmos de encaminamiento se diseñaron para este sistema, de cara a optimizar el uso del diseño de los conmutadores BXI. El tema principal de esta tesis surgió con la intención de mejorar estas técnicas, y al intuirse que la nueva arquitectura de hardware BXI permitiría desarrollar la investigación de forma diferente. Esta investigación se ha desarrollado conjuntamente con el laboratorio LI-PaRAD (Laboratoire d'Informatique, Parallélisme, Réseaux, Algorithmes Distribués) de la Universidad de Versailles Saint-Quentin-en-Yvelines (UVSQ) en Francia. La tesis también se ha desarrollado conjuntamente con el grupo de Redes y Arquitecturas de Altas Prestaciones (RAAP) de la Universidad de Castilla-La Mancha (UCLM) en España, mediante una cotutela internacional. El contexto industrial ha tenido una gran influencia en la investigación desarrollada, y varias contribuciones de la tesis se han patentado e integrado en productos comerciales. Sin embargo, se ha tenido especial cuidado en estudiar casos genéricos, y por tanto en general los resultados del trabajo son aplicables a otros entornos.

1 — Context and state of the art

1.1 . Introduction

Today's most powerful supercomputers built for High-Performance Computing (HPC) are clusters of high-end commodity hardware, composed of millions of cores, costing tens of millions of euros, and requiring multiple megawatts [81]: computing power remains prohibitive and resource-consuming. The interconnection network (*interconnect*) which connects such a system allows applications to be distributed across many nodes to speed up their run time, or increase their scale, resolution, or precision. The performance of the system as a whole is highly dependant on the performance of the interconnect [24, 3]. Correspondingly, the HPC interconnect is over-engineered to almost never fail so as to minimise layers targeting resilience, and this network represents a significant fraction of the power and hardware allocated to modern supercomputers. However, the increase in power and scale of these systems has caused hardware failures to become unavoidable. A significant part of the research in this area aims to minimise the cost and consumption of these networks while maximising their performance and fault resilience. Routing plays a vital role in the HPC interconnect, which is overall an intricate system whose design and usage affect cost, performance, and fault resilience. This thesis focuses on improving routing techniques for HPC interconnects by designing and statically analysing new static routing algorithms and extensions to existing ones. This choice comes from an observation that current research is often largely about dynamic studies of advanced aspects of HPC interconnects (such as adaptive routing, dynamic job scheduling, and trace-based optimisations), whereas many of the underlying performance issues justifying these methods could be better addressed by the static routing algorithm beforehand. The proposed routing techniques offer improvements in either or both performance and fault resilience, while partially relaxing a common design constraint (network topology).

This research was initiated in an industrial context as Bull was designing an in-house interconnect for its new Bull eXascale Interconnect (BXI) clusters, alongside a new *fabric manager* (software to monitor and configure the network at a high level, which includes maintaining functional and efficient routing). BXI switches are designed with an integrated ARM CPU and a management network separate from the interconnect, as well as other design specificities. To adapt to these aspects, and use the occasion to start from a clean slate, the BXI fabric manager (BXI FM) is organised differently from competitors' subnet managers. Some routing techniques and algorithms were designed for this system to make full use of the switch design. Out of desire to improve on these techniques, and an intuition that research could be conducted dif-

ferently with this new hardware architecture, the initial thesis subject was drawn out. Research is conducted in conjunction with the LI-PaRAD laboratory (Laboratoire d'Informatique, Parallélisme, Réseaux, Algorithmes Distribués) at UVSQ (the Université de Versailles Saint-Quentin-en-Yvelines) in France. Research is also conducted with the RAAP laboratory (Redes y Arquitecturas de Altas Prestaciones) at UCLM (the Universidad de Castilla-la-Mancha) in Spain, through an international cotutelle. The industrial setting has strongly influenced the research, and several inventions have been patented and integrated in the commercial product during the doctorate. As a whole, however, care has been taken to focus on general cases and the resulting work is largely applicable to other settings.

The rest of this chapter covers elements of contexts for HPC interconnects, routing, and topologies. Firstly, Section 1.2 introduces general elements of HPC interconnects and their routing. Section 1.3 presents topologies used in HPC interconnects, as well as an overview of their current usage by the main HPC vendors, in order to define the target topologies of this thesis as precisely as possible. From there, Section 1.4 studies several existing fat-tree-specific static routing algorithms on which this thesis is based. Section 1.5 finally lays out the problem statement more precisely in the given context, and plans out the following chapters of contributions accordingly.

1.2 . Problem model

1.2.1 . HPC interconnects

Interconnection networks are designed to allow various devices to communicate. To achieve this goal, *switches*[1] are needed to forward *messages* from their sender to their recipient. The devices are called *nodes*[2], or *endnodes* in some literature so as to distinguish them from switches. Switches communicate between one another and with nodes via bidirectional *links*. Links (either copper or optical) are plugged into switch *ports* (via a transceiver pod for optical links). There may be multiple links connecting the same two switches; multiple ports of a switch leading to the same *remote* switch are part of a *port group*. All the links in each considered network are considered equivalent, in terms of latency and bandwidth. In networks where some switches are not directly connected to nodes (*indirect networks*, see Section 1.3.2), those connected to at least one node are distinguished by being called *leaf switches* or simply *leaves*. Switch ports leading to nodes are called *endports*.

The target communications of the network are those going from node to node (that is, from endnode to endnode). In the context of BXI, where switches are managed *out-of-band* (through a separate network), switches are never the

[1]Switches differ from *routers* because switches connect nodes of the same *subnet*, while routers connect different subnets together—only single subnets will be considered here.

[2]Types of nodes include compute blades, service nodes, I/O nodes, and accelerators.

source nor the destination of traffic. This makes the node/switch distinction clear cut, and it will be taken into account in our network model and its properties. In existing literature, switches are at times considered as nodes like any other, to the point where many articles do not specify the nature of graph vertices. Classical graph theory and its results generally do not distinguish vertices either.

Taking into account this distinction in the network model, we hereafter note N the set of nodes, and S the set of switches. Each node has a unique identifier (its *NID*), decided automatically or manually. A link is defined by the nodes or switches it connects, and a locally unique identifier corresponding to the port number, or port *rank*. We model links with E, the set of directional edges; E is a subset of $(S \cup N)^2 \times \mathbb{N}$ since no link can connect more than two elements, and several links can join the same two switches: this network model is a multigraph, noted $G(S,N,E)$. Elements of E are noted $((i,j),k)$, denoting links from i to j, locally indexed k. This is the only difference with the host-switch graph model [89], if we add the restriction that each node is connected to the rest of the graph by one link only. This restriction is valid throughout this thesis, though there also exists research targeting multi-ported host-switch graphs [90]. From this model we can deduce the set of leaves, noted L:

$$L = \{s \in S \mid \exists\ n \in N,\ ((s,n),0) \in E\}$$

This model of the network is called its (multi)graph or *topology*; though the word topology is also used interchangeably with classes of topologies, discussed in Section 1.3.

Some (classes of) topologies are explicitly defined using *up* and *down* directions, possibly through the definition of non-negative integer *levels* of switches. From this, we prepend an up, *same*, or down directional prefix to the ports, port groups, links, or neighbouring switches of a switch where it applies. These are used extensively in indirect topologies, defined in Section 1.3.2. Nodes and switches are physically organised in racks, often with a top rack (which may or may not correspond to the topological top level) to facilitate inter-rack cabling. The physical organisation often directs or constrains the topology, though sometimes it is the topology design which causes physical organisation choices.

Interconnection systems such as InfiniBand, Cray and BXI implement some form of *channels*, also referred to as *virtual channels* (VCs), or *virtual lanes* (VLs), depending on the context; wherein a single physical port can be used for multiple traffic flows without affecting one another. VCs require sharing the port's buffer space and temporal link usage, either statically or dynamically. VCs are primarily used for quality of service (QoS), wherein different classes of messages are guaranteed not to interfere with one another. They are also used for deadlock avoidance, as mentioned in Section 1.2.7, though the work in this thesis relies on simple edge-level deadlock avoidance. Furthermore, they

are also used to optimise quality of routing for some topologies [8]. Finally, they are also often used by queueing schemes for congestion avoidance and management as well [33, 91]. VCs are largely out of the scope of this thesis, and simple links will be modelled, though there will be several references to VCs. VCs are defined by an edge and a non-negative integer identifier, the channel rank. They can be modelled by C, the set of channels, a subset of $E \times \mathbb{N}$, itself a subset of $(S \cup N)^2 \times \mathbb{N}^2$. Other literature often models topologies as $G(S,N,C)$.

Specific notations used in this thesis are provided at the end of this Chapter, in Table 1.1.

1.2.2 . Network definitions

We will now go over several network definitions which will be useful later on to assess network characteristics for the context and for some contributions.

Definition 1. *A path from node a to node b is a sequence of edges $(e_1, e_2, \dots e_n)$ such that the end of e_i is the start of e_{i+1}, and such that e_1 starts at a and e_n ends at b. The number n of edges in the path is its length, or number of hops.*

Definition 2. *The* distance *between nodes a and b, noted $d(a,b) = d(b,a)$, is the shortest length attainable by paths from a to b (and conversely b to a, since links are bidirectional). Any path whose length is minimal, or equal to the distance, is called a* shortest path.

This distance verifies the triangular inequality.

Definition 3. *The* diameter *of a network G is the largest distance between all pairs of nodes: $D(G) = \max\{d(a,b) \mid a,b \in N\}$.*

For example, the diameter of a fully-connected network is 1 and the diameter of a bidirectional ring with n nodes is $\lfloor \frac{n}{2} \rfloor$.

Definition 4. *A partition of N is a couple (A,A') such that $A \subset N$, $A' \subset N$, $A \cup A' = N$, $A \cap A' = \emptyset$.*

Definition 5. *A cut (A,B) of $G(S,N,E)$ is a partition of $S \cup N$. The cut-set of (A,B) is the set $\{(u,v) \in E \mid u \in A, v \in B\}$ of edges that have one endpoint in A and the other endpoint in B. The node-partition of (A,B) is the partition $(N \cap A,\ N \cap B)$ of N.*

Definition 6. *The* bisection *(A,B) of a network is a cut whose node-partition (N_A, N_B) verifies $|\#N_A - \#N_B| \leq 1$.*

Informally, a cut is a bisection if it splits endnodes evenly. Note that in graph theory in general, where switches and endnodes are not differentiated, a bisection splits all nodes evenly. This is one of many cases where we diverge from general graph theory due to the switch/endnode distinction.

Definition 7. *The* bandwidth *of a cut is the number of edges in its cut-set.*

In general, bandwidth is computed as the sum of individual link bandwidths, but we instead assume that all links are equivalent.

Definition 8. *The* bisection bandwidth *of a network, or* cross-bisectional bandwidth *(abbreviated CBB), is the minimum number of links of all the bisections of a network, defined as:*

$$\min\{\#E_C \mid E_C \textit{ the cut-set of a bisection of } G\}$$

For example, the CBB of a fully-connected network with n nodes is $\left\lfloor \frac{n}{2} \right\rfloor \times \left\lceil \frac{n}{2} \right\rceil = \left\lfloor \left(\frac{n}{2}\right)^2 \right\rfloor$ and the CBB of a ring is 2. The CBB is a theoretical worst-case scenario bandwidth of network-wide group communications assuming perfect path selection. However, the actual path selection might result in significantly lower effective bandwidths [40].

Definition 9. *The* blocking factor *of a network is the ratio between the greatest and smallest bandwidths of its bisections, defined as:*

$$\frac{\max\{\#E_C \mid E_C \textit{ the cut-set of a bisection of } G\}}{\min\{\#E_C \mid E_C \textit{ the cut-set of a bisection of } G\}}$$

Informally, the blocking factor represents the maximum variation in worst-case scenario bandwidth of network-wide group communications, depending on communication pattern.

Definition 10. *A network has* constant bisection bandwidth *(or* constant CBB*) if its blocking factor is equal to 1.*

This definition means that all the bisections of a network with constant CBB have the same bandwidth.

HPC cluster interconnect design often relies on networks being connected according to specific topologies, which will be discussed in Section 1.3. The aim is generally to guarantee beneficial network properties such as high CBB and low diameter, while simplifying routing design for various criteria.

1.2.3 . Routing for HPC interconnects

Routing is the exercise of assigning correct paths to all messages. It is different from *switching*, which describes how messages are processed at a lower level. To provide basic functionality, routing should ensure that messages will reach their destination in finite time. Firstly, this means that every message must be directed to its correct destination (and naturally avoid infinite loops). Furthermore, *deadlocks* must be avoided: a deadlock situation occurs when there is a cycle of packets which can never advance because they each require the next channel to be freed [14], as shown in Figure 1.1. Deadlocks

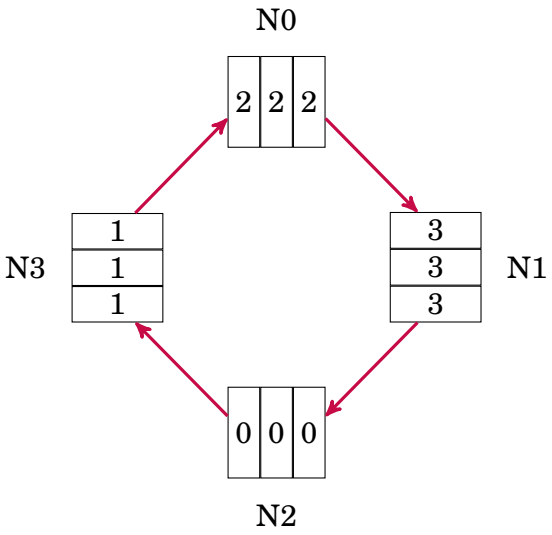


Figure 1.1: Example of a deadlock, with four nodes N0 to N3, each sending a message to the opposite node, and with every buffer full.

are particularly problematic in *lossless* networks (in which packets cannot be dropped), which form the majority of HPC networks. More on deadlock avoidance can be found in Subsection 1.2.7.

It is not sufficient for routing to be functional, it must also result in low congestion for the supercomputer to perform efficiently. Different implementations of routing, or *routing functions*, may affect network performance in various ways. Network latency may be affected by paths being longer than necessary (a routing function is *minimal* if it guarantees shortest paths, and *non-minimal* otherwise). Furthermore, the routing function may cause *contention* of network resources, increasing latency and/or reducing throughput. Sustained contention, or *congestion*, is often shown to take the form of points of contention causing buffers to progressively saturate in routes crossing these points. These trees of congestion are called *congestion trees*, and they evolve towards sources of contending packets. This is sometimes distinguished into low-order head-of-line (HoL) blocking for so-called *hot* message flows directly causing congestion versus high-order HoL blocking for *cold* message flows which happen to go through an already congested edge [46].

In commonly used topologies, path length is constrained. Furthermore, path length has lesser repercussions on application latency than other parameters such as network congestion. For these reasons, the design of high-quality routing algorithms mostly aims to reduce or minimise congestion risk, generally by spreading competing traffic flows across the available resources, or *load balancing*. Depending on the granularity, these resources might be switch-paths, links, or virtual channels. Splitting competing traffic into different virtual channels reduces head-of-line blocking [24]. However, designing and

using more VCs increases latency and hardware complexity, and is only useful when it provides greater benefits than its own cost.

Other mechanisms attempt reactive congestion management by throttling (temporarily slowing down) offending message producers [30, 42] with hard-to-tune results that generally do not scale [102, 31]. However, Cray claims to have implemented a stable and scalable reactive throttling congestion management mechanism [13, 18].

The study of quality of routing is approached in various ways in existing research, where techniques are more often than not studied in experimental and quantititave approaches, and precise characterisation of HoL blocking is not practical. Some techniques are studied in general approaches, such as approximation of effective CBB based on common usage scenarios [40]. Other techniques are studied under specific use cases (for individual target clusters, with Deimos [19] being one example among many others) either by experimentation with sample applications (such as specific benchmarks like the NAS Parallel Benchmark suite in that same cited work, once again with numerous other examples) or by simulation of synthetic traffic (with hotspot traffic [70] being a common example to study congestion) or communication traces from real application runs [9]. The goal is either to directly compare application run times or to estimate effective network metrics potentially transferable to other use cases. Common metrics studied using experimentation or simulation are effective throughput and latency as functions of offered communication load, from which can generally be interpreted comparable characteristics such as saturation load and corresponding max throughput.

A simpler approach is to count the number of paths at each edge, or port, of the network. The maximum value, called edge-forwarding index, reflects the maximum load of the network under sustained communication between all nodes [39].

Other techniques taking into account more information such as job placement, communication pattern matrices, or fault tolerance, usually integrate the corresponding data points in their study. Chapter 2 presents the approach used in this thesis to study quality of routing which is useful for the scope of the proposed techniques. A specific use case will be presented in Section 1.2.8.

In case of equipment failure or other unexpected behaviour, routing must be fault-resilient to remain functional. Fault resilience will be presented in Section 1.2.9. Contexts for these upcoming subjects are impacted by the types of routing implementations, which will therefore be presented beforehand in Sections 1.2.4 and 1.2.5, followed by a categorisation of approaches to programming routing algorithms in Section 1.2.6.

1.2.4 . Centralised static routing

Static—or *deterministic*—routing is the simple and efficient type of routing that is most widely available in HPC. Such a routing can be characterised by a function of the form $R : N^2 \times S \to E$, where a message coming from and going to nodes (in N^2) via a switch (in S) is routed via an output edge, which must naturally start from the considered switch. In the majority of applications of static routing, however, routing is source-independent, and can be characterised instead by a function of the form $R : E \times N \to E$, where each input port (in E) and each destination (in N) are associated with a corresponding output port. In practice, this function is implemented using Linear Forwarding Tables (LFTs, often referred to as *routing tables*), with generally one entry per switch and destination, valid for all the input ports of the switch (with the shape $R : S \times N \to E$). Edges are used in each model for simplicity, though channels are considered in other works, by substituting E with C. Interconnection networks for HPC usually have a centralised static routing mechanism wherein a subnet manager (responsible for its connectivity) computes LFTs off line and uploads them to the switches. Static routing guarantees in-order delivery in lossless HPC networks.

Static routing is *oblivious* insofar as it does not take into account the current state of the network. However, there are also some *dynamic* (i.e. *multi-path*) routing techniques which are oblivious [24, 17, 36]. The only dynamic oblivious techniques commonly found in the existing literature involve a random selection: take for example Valiant routing which chooses a random intermediary switch to send messages, later performing minimal routing from that switch to the destination [82].

1.2.5 . Adaptive routing

An *adaptive* routing algorithm, unlike oblivious ones, takes into account the state of the network to make routing decisions [24]. Figure 1.2 summarises algorithmic categories described up to now. This network information may be local information like port/VC congestion or the number of available credits, or global information like switch congestion across the network. A good adaptive routing technique overcomes the inflexibility of static routing by making use of under-used resources to alleviate congestion. However, adaptive techniques that eagerly spread traffic onto a large proportion of network resources tend to increase congestion [70, 71]. Adaptive routing breaks the guarantee of in-order delivery within communication flows [34].

BXI implements adaptive routing for traffic (marked as authorised for out-of-order delivery) with a search of lowest contention (measured as buffer usage across VCs) between the deterministic route and a random set of alternative routing tables. The set size and activation threshold are configurable. The alternative routing tables are computed by the routing algorithm and uploaded by the fabric manager.

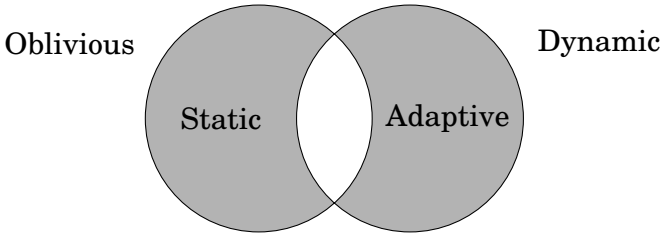


Figure 1.2: Categories of routing algorithms

Distributed applications that take advantage of BXI NICs indirectly rely on the Portals interface [4]. Some communications are implemented using PtlPut, for which some layer must provide ordering between packets and messages in each flow. Offloading the Portals matching engine to the NIC imposes restrictions on the amount of memory available per flow, and reordering at the destination becomes untenable. Instead, these messages are marked at the source as non-adaptive, so as to benefit from the in-order delivery guaranteed by static routing. This is one reason why good quality static routing was deemed critical and motivated the subject to be explored in this thesis.

1.2.6 . Approaches to programming routing algorithms

Several categories of approaches to programming routing algorithms are presented below, with references to corresponding example routing techniques. These are not strictly speaking algorithmic categories, and there is some overlap, but they do outline to some extent the space of possibilities available to the programmer.

Dynamic programming and greedy algorithms A dynamic programming technique breaks down the problem into a sequence of subproblems of the same type, each easier to solve than the full problem. A greedy algorithm progressively builds a global solution through a series of locally optimal choices. These approaches might be intuitive insofar as they could generally be described with the same steps a person might take to solve the problem by hand, such as assigning routes to each destination one after the other, based on previous decisions like Dijkstra or Bellman-Ford algorithms. Greedy algorithms are difficult to distribute or parallelise while also keeping guarantees about global results. Examples include Ftree [97], SSSP [41] (and its slower but deadlock-free counterpart DFSSSP [19]), and Nue [21].

Closed-form algorithms Some algorithms determine each element of the solution using a closed-form operation, i.e. based on input alone and no dependency to other elements of the solution. These methods might require a preprocessing phase to prepare the information each element requires, such as topological addresses. This phase can be made more efficient when topological properties are regular. Such a formulation allows implementing a perfectly parallel computation phase. Examples include Dmodk [95] and Smodk [61, 73].

Randomised algorithms Routing algorithms are sometimes based on random choices as a means to approximate agnostic load balancing or congestion avoidance. Examples include Valiant's algorithm [82] and Greenberg's RANDOM [37].

MILP programming Some routing methods describe the constraints of the problem as a set of linear relationships. This way, load balancing explicitly becomes an optimisation problem. This approach is used mostly for communication-pattern-aware routing. Generic Mixed Integer Linear Programming methods can then be used, with potential tweaks corresponding to the specific nature of the problem space, such as BSOR, BSORM [49], and some with topology-specific tweaks to achieve better routing run time [66]. BeFS [72] is another discrete method, using branch-and-bound.

Distributed algorithms In some cases it may be possible to build a globally correct or optimal solution with no central computation unit. In such situations, distributed computation units can communicate with one another and use local information to make decisions. Some routing techniques, especially adaptive ones, rely on switches computing routes based on local information. This might provide fast reaction time to network changes or congesting traffic. Examples include the Chaos router [50] and UGAL [80], two techniques for adaptive routing.

Centralised algorithms are beneficial, because they are generally easier to design than fully distributed ones, even when multithreaded. Like the majority of similar products, BXI indeed uses only centralised algorithms to compute routing tables.

1.2.7 . Deadlock avoidance

Deadlock avoidance is a major aspect of the co-design of switches, topologies and routing algorithms. Packet dropping is a simple mechanism to guarantee deadlock-freedom (wherein a packet is simply discarded if it does not advance fast enough), but HPC interconnects are lossless to improve performance, and are therefore at real risk of deadlocks.

Most approaches used to guarantee deadlock-freedom rely on the absence of channel dependency cycles. A channel dependency $a \rightarrow b$ corresponds to the potential for channel b to be used by a message after it has used channel a.

Definition 11. *The* channel dependency graph *(CDG), for a given interconnection network, I, and routing function, R, is a directed graph, $CDG(I,R) = G(E,D)$. The vertices of $CDG(I,R)$ are the edges of I, and its edges are the pairs of edges connected by R:*

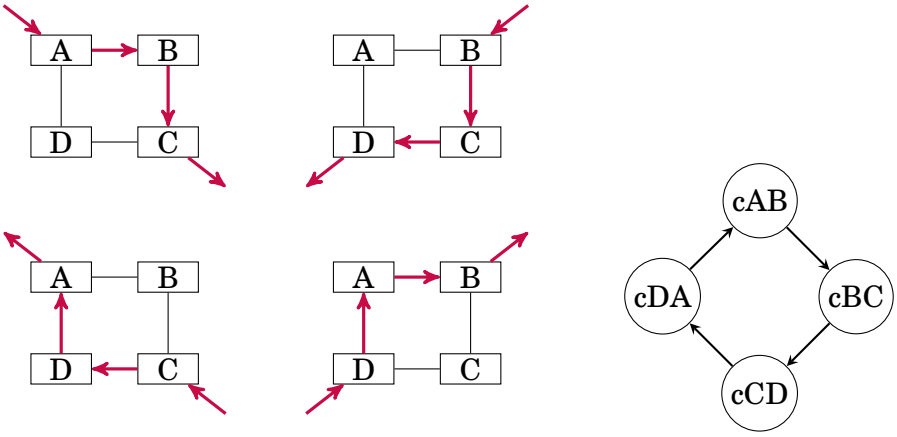
$$D = \left\{(e_i, e_j) \in E^2 \mid \exists\, n \in N,\ R(e_i, n) = e_j\right\}$$

Since there is no route from an edge to itself, there is no 1-cycle in CDG.

Definition 12. *Channel dependencies are connected in CDG paths as follows:*

$$[e_0, e_1, \ldots] = [\ldots, (e_i, e_{i+1}), (e_{i+1}, e_{i+2}), \ldots], \quad \forall\ i,\ (e_i, e_{i+1}) \in D$$

Dally proved that cycles in the CDG were a necessary condition for deadlocks to arise [14]. Figure 1.3 provides an example potential deadlock situation and the corresponding CDG. From this, it is sufficient to guarantee the absence of cycles in the CDG to prove deadlock-freedom. This only requires a static understanding of the routing function, but ignores some deadlock-free situations that arise dynamically. Guaranteeing cycle-free CDGs is therefore relatively simple, and is generally implemented through directional routing restrictions, as will be explored further in Section 1.3. Some research goes further and explores deadlock-freedom in routing functions which contain cycles in the CDG [78].



(a) A ring of four switches with routes in red

(b) The cycle in the corresponding CDG

Figure 1.3: Example of a potential deadlock situation. Deadlock occurs if all the buffers in the cycle become full.

1.2.8 . Node-type heterogeneity

Many large supercomputers are used for varied applications, with varied resource needs. To provide adequate resources, these supercomputers are often designed with multiple kinds of available equipments, whose usage impact network usage. In practice the type of equipments encountered are among the following:

- Compute nodes, with powerful central processing units (CPUs);
- Storage nodes, with slower remote access to large regions of memory than local memory;
- General purpose graphic processing units (GPGPUs), with fast parallel units for specialised SIMD (single instruction multiple data) operations;
- Field-programmable gate array units (FPGAs), with reprogrammable gate logic for fast processing of general purpose applications;
- Service nodes, operating network management or interacting with external networks for user access.

These might be connected directly to other components, or to the interconnection network (in which case they are seen as endnodes). There are approaches to extend usability of directly connected equipment to other nodes, such as rCUDA for GPUs [26]. That is outside the scope of the interconnection management layer of the network. On the other hand, use of equipment visible as endnodes can be strongly affected by the network management layer.

In particular, one case in which the network might provide inadequate performance as a result of node-type heterogeneity is that of applications in which communications are organised in separate time frames for separate types of equipment. Observation shows that purely static (and therefore oblivious) routing in such cases might restrict traffic to each node-type group to a subset of available network resources and result in lower performance. Such a situation, which happened during the course of this thesis, is studied in more detailed in Chapter 3.

1.2.9 . Fault resilience

Network topologies and corresponding routing are often defined statically, while real operation may require replacing or extending network components. Furthermore, as the sheer amount and power of equipment increases in current and future supercomputers, the average rate of equipment failures increases accordingly [76, 20, 88]. Failures are especially frequent with optical links [12] typically used to cover long distances in large scale clusters. For these reasons, fault resilience becomes important to avoid frequent application interruption. A centralised subnet manager (or fabric manager) can react to equipment failures that do not break graph connectivity by uploading updated routing tables. In order to do this, it requires a fault-resilient routing algorithm capable of rapid re-routing. The challenge is to provide these characteristics while maintaining high-quality static load balance.

There are various approaches to guaranteeing deadlock-freedom during transition phases in case of routing change [25, 53], but when possible, a simple approach is to prove that every possible routing function is part of a *super function* which is itself deadlock-free, thus guaranteeing that any transition step between any two possible routing functions is also deadlock-free [68]. This is possible for example when a direction restriction applies uniformly to any degraded state of a network, as will be the case in Section 1.3.2.

More context regarding topology-specific fault resilience will be provided in Section 1.4.6.

1.3 . Topologies

Topologies are ubiquitous in the design of supercomputers. They help provide guarantees about performance, resilience, or other design goals through their network characteristics. For example, a topology with a lower diameter tends to constrain application latency more, and a topology with a higher CBB tends to accommodate a higher bandwidth. Other aspects can be considered, such as required hardware availability and cost, ease of expansion, and ease of deadlock-free routing.

The purpose of this Section is threefold. Firstly, it should help justify the focus on fat-trees, which are only one among many other usable topologies.

Secondly, it should give some context to help appreciate the pros and cons of fat-tree-specific routing. Lastly, it should define to some extent what the specific target topologies are, given the wide array of topologies called fat-tree in the existing literature.

1.3.1 . Direct topologies

In *direct* topologies, every switch is connected to nodes (through *internal* links) to give them access to the rest of the network (via *external* links). In such a case, $S = L$. By considering each switch and its directly connected nodes as one vertex of the network graph, this allows simplifying the graph to its switches and external links. The number of nodes connected to each switch is sometimes specified in the topology definition. For a given switch radix and number of nodes, the external connection scheme affects the corresponding number of switches and links required, directly impacting network cost. Performant direct topologies generally result in networks in which it is difficult to integrate new equipment. Hop count is given only across external links, therefore artificially reducing the diameter by 2 hops, assuming nodes are connected with the same links as inter-switch connections. Examples of direct topologies are provided below.

Definition 13. *A* ring *topology is a connected graph where each vertex is connected to exactly two different vertices with one link each. A* bus *topology is a ring with one link removed.*

Both of these topologies are extremely cheap in terms of link count: with $N = n$ switches, only n or $n-1$ links are required. However, they have low CBB (1 or 2) and high diameter ($n-1$ or $\lfloor \frac{n}{2} \rfloor$), making them perform poorly for concurrent applications. Furthermore, rings are easily affected by deadlocks: guaranteeing cycle-free CDG in a ring or bus requires either direction restrictions or using multiple VCs.

Definition 14. *A* mesh *topology is a two-dimensional square grid of vertices, where non-diagonal neighbours are connected with one link. A* torus *topology is a mesh where the first and last vertices of each row and column are connected with one link.*

Meshes are also cheap: with $N = n \times n$ switches, only $2(N-n)$ links are required. They provide a slightly higher CBB of $2\lceil \frac{n}{2} \rceil$ and a slightly lower diameter of $2(n-1)$, but still result in poor performance. To avoid deadlocks, either a turn order model [35, 60] or separation into distinct VCs is required [15]. Torus topologies cost only slightly more than meshes ($2N$ links), cut the diameter down to $2\lfloor \frac{n}{2} \rfloor$, and raise the CBB to $4\lceil \frac{n}{2} \rceil$.

Definition 15. *A* Hypercube *topology is a higher-dimension 2-by-2 mesh. A* k-ary n-cube *topology is a generalisation of a torus with n dimensions and k vertices in each dimension.*

Note that hypercubes are 2-ary n-cubes, and that toruses are k-ary 2-cubes. k-ary n-cubes provide competitive topological characteristics [16] and are commonly used in massively parallel processors (MPPs) and HPC interconnects (such as the Tofu interconnect, used in the K computer and the Fugaku supercomputer, with each being the most performant at the network-intensive HPCG benchmark at the time of their installation [23]), despite difficulties in cabling, scalability, and deadlock-freedom.

DragonFly (DF) topologies [48] are a more recent field of research. Specific DFs are defined by specifying groups of switches and the intra-group and inter-group connection schemes. A common example is a DF with a flattened butterfly intra-group scheme and a pruned Hamming graph inter-group scheme. Real DFs have diameters ranging from 2 to 5. Routing DFs requires special attention in order to guarantee deadlock freedom [32, 57, 56]. Furthermore, DFs provide low performance for adversarial inter-group traffic patterns unless either fine-tuned non-minimal[3] adaptive routing techniques [48, 44, 67] or group-spreading job placement policies (RDR or RRR in [43]) are used. Similarly, when jobs are placed contiguously, all links connecting groups assigned to different jobs are left unused in case of minimal routing; these links are referred to as *dark* links [22].

Other large low-diameter direct topologies have been studied in detail [47]. Designing a diameter-2 topology with high bandwidth, path diversity, and node count results in networks with structurally guaranteed low congestion and latency, but requires careful attention and generally requires complicated routing methods.

1.3.2 . Indirect topologies

An *indirect* topology contains non-leaf switches, wherein $L \subsetneq S$. This section will present the only common type of indirect topology, *Multistage Interconnection Networks* (MINs).

Definition 16. *A* non-blocking *telephone network is always capable of accommodating any new call (from an available caller to an available recipient), regardless of existing traffic.*

Definition 17. *A telephone network is* rearrangeably non-blocking *if for a given set of calls, there exists a routing function for which the network will not block.*

Clos designed indirect topologies for point-to-point telephone networks (using crossbar switches) to guarantee non-blocking switching at reasonable cost [11]. These unidirectional multistage interconnection networks (MINs, or UMINs) are composed of one level of input switches, an odd number of levels of intermediary switches, and finally one level of output switches. Depending on

[3] In DFs, *minimal* and *non-minimal* often refer to use of intergroup links only, or not, respectively.

the numbers of input ports, output ports, and input switches, a Clos network can be non-blocking, or only rearrangeably non-blocking.

Unlike UMINs, bidirectional multistage interconnection networks (BMINs), are composed of bidirectional links and have all leaf switches on one side (the bottom one, with levels ordered vertically). The height (in number of levels) of a BMIN is noted h.

Definition 18. *In a BMIN, a node is* down-accessible *from a switch if it can be reached with a path made up only of downward hops.*

Definition 19. *The* direction *of an edge e in a BMIN, noted* $\mathrm{dir}(e)$*, is equal to the difference of levels incurred by following the edge.*

For example, an edge leading from level 4 to level 1 has a direction of -3.

Definition 20. *In a BMIN, a* topological group *is a subgraph connected to the rest of the graph by up-edges only (with* $\mathrm{dir} > 0$*), which lead only to switches at a level greater than all those in the subgraph.*

Definition 21. *An* up–down *path* $(e_1 \dots e_n)$ *is such that:*

$$\begin{aligned} \exists\ 1 < i \leq n, \quad \forall\ j < i, \quad & \mathrm{dir}(e_j) > 0, \\ \forall\ j \geq i, \quad & \mathrm{dir}(e_j) < 0 \end{aligned}$$

Definition 22. *An up–down routing function is made up only of up–down paths.*

Property 1. *Any up–down routing function is deadlock-free in BMINs.*

This advantageous property (applicable to fat-trees, which are described in Section 1.3.3), is claimed in existing literature [64, 24], and Duato extends the notion of strict global ordering of links sufficient to prove deadlock-freedom in UMINs, to a strict local ordering of links in BMINs. A new proof, which doesn't rely on extending the UMIN property, is provided here.

Proof. In a BMIN, edge directions are either upwards, samewards, or downwards, and each channel dependency in its CDG has a pair of directions based on starting and ending edges. In an up–down routing function, no such edge is samewards, and valid pairs of directions can only be up–up, up–down, down–down (and not down–up). From this, CDG paths (see Definition 12) can only be formed as a concatenation of a CDG subpath with only upward edges and one with only downward edges, each of length 0 or more. Deadlock-freedom can be proven by showing that no valid path in the CDG contains a cycle, as seen in Section 1.2.7. To have a cycle, there must exist one valid CDG path verifying:

$$i < j, \quad e_i = e_j$$

If e_i is upwards, then e_j must occur in the upward CDG subpath; however, this CDG subpath is strictly monotonous in terms of levels and cannot contain a cycle. The same applies symmetrically if e_i is downwards. Therefore, there cannot be any cycle in the CDG. □

As described in Section 1.2.7, and defined in [68], up–down super routing functions in BMINs are deadlock-free, thus guaranteeing transitively deadlock-free fault-resilient algorithms.

Property 2. *Any up*–same?–down* routing function is deadlock-free in BMINs.*

Note that * means "any number of times" and ? means "at most once", similarly to regular expressions. Informally, this describes routes with at most one samelink between the upward and downward phases. A novel proof, using the proof for Property 1, is provided here.

Proof. In an up*–same?–down* routing function in a BMIN, there is no down–up, down–same, same–up, or same–same dependency in any CDG path. We will prove deadlock-freedom by showing that there is no cycle in any valid CDG path. If such a CDG path contains no same edge, then the proof for up–down routing functions provided in Section 1.3.1 applies, and there cannot be any cycle. Otherwise, the sameward edge in the CDG path is preceded only by upward edges, and followed only by downward edges. By virtue of strict monotonicity, neither side contains a cycle. By virtue of uniqueness (and lack of any 1-cycle), the sameward edge cannot induce a cycle. Therefore, there cannot be any cycle in the CDG. □

1.3.3 . Fat-tree topologies

A majority of current leading network topologies for HPC clusters are fat-tree variants: the five most powerful clusters of the June 2019 Top500 list [81] had fat-tree topologies. The name fat-tree has been used to describe many different topologies since the original CM-5 version. Despite many differences in definition, fat-trees are always BMINs with at most one downward switch-path from any top switch to any leaf switch. Fat-trees are usually regular and symmetrical. Their "fatness" corresponds to increasing bandwidth when going up levels, and this increase is often designed to provide a constant CBB. Many studies associate fat-trees with this property while others distinguish non-slimmed and slimmed (or oversubscribed) fat-trees. The other important beneficial property is that minimal routing in fat-trees is up–down and therefore deadlock-free.

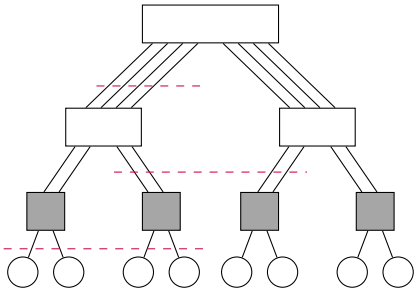


Figure 1.4: Three level binary Leiserson fat-tree. Switches are rectangles (with leaves in grey) and nodes are circles. Several bisections are shown in dashed red lines, illustrating constant CBB.

Leiserson fat-trees

Fat-tree topologies were introduced by Leiserson [51] in the CM-5 for their capacity to accommodate any virtual network for a given size. These fat-trees are organised like regular trees (with a constant number of connected links across switches, or *arity*, k), but with aggregate link bandwidth increasing by a factor k at each level. Note that in their original declination, this rule was only precisely defined for so-called *universal* fat-trees; however, literature generally considers these only when referring to Leiserson fat-trees. Levels go up from the leaves to the top switches. An example with $k = 2$ and $n = 3$, n being the number of levels, is shown in Figure 1.4.

Massive Parallel Systems such as the CM-5 implemented fat-trees using efficient on-chip networks with so-called concentrator switches. Leiserson fat-trees cannot be implemented using the crossbar switches found in clusters because they would require an $(\#N)$-port top switch. Were such a switch available, connecting it to all nodes directly would be the best and cheapest possible network.

A short section regarding routing in Leiserson fat-trees is provided in annex Section 1.

K-ary n-trees

K-ary n-trees were subsequently formalised by Petrini [64] and describe an implementation of fat-trees using fixed-radix crossbar switches organised in *complete bipartite* subgroups (wherein two sets of switches in adjacent levels are fully connected, but only across levels). The diameter of a k-ary n-tree is twice its number of levels. Bisection bandwidth is constant, which is often described as being rearrangeably non-blocking. That description relates to the potential capacity for such a network to have a non-blocking routing function for any communication permutation[4], though that is not possible using static routing, and non-permutation communications are to be expected as well. The

[4]The notion of non-blocking routing functions dates back to circuit-switching networks, in which only permutations can be performed

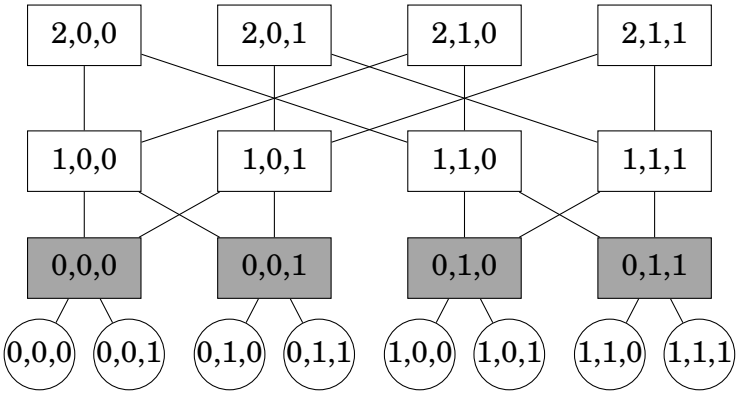


Figure 1.5: A 2-ary 3-tree (with $2^3 = 8$ nodes). Each switch is required to have 4 ports except for the top switches (which have only 2 ports).

connection scheme is defined formally using two rules based on switch and node addressing, both n-tuples.

Definition 23. *In a k-ary n-tree, switches $(l, w_0, \ldots, w_{n-2})$ and $(l', w'_0, \ldots, w'_{n-2})$ (of level l and l' respectively, with $l' \geq l$) are connected if and only if $l' = l + 1$ and $w'_i = w_i, \forall i \neq l$. Leaf switch $(0, w_0, \ldots, w_{n-2})$ and node $(w'_0, \ldots, w'_{n-2}, p)$ are connected if and only if $w_i = w'_i, \forall i \in [0, n-2]$, with $p \in [0, n[$.*

When routing k-ary n-trees, every pair of nodes has multiple nearest common ancestors (NCAs). Optimal routing then comes down to distributing NCAs via which to route to avoid network-congestion from happening in the first place. Deterministic and adaptive routing algorithms have been provided for k-ary n-trees [36].

GFTs, XGFTs, m-port n-trees

Generalised Fat-Trees (GFTs), introduced by Ohring [61], describe a more general class of topologies for which the upward and downward arities can differ.

Definition 24. *A graph defined by the formula*

$$GFT(h;\ m;\ w)$$

is a GFT with h levels, with each switch connected to m switches below and w switches above. It is constructed recursively from m distinct copies of $GFT(h-1;\ m;\ w)$. Each of these fat-trees, used to build the complete graph, is referred to as a topological group.

This allows creating oversubscribed fat-trees with smaller and cheaper networks and only partly reduced overall performance. This process of reducing the bisection bandwidth of a network is often called *pruning*. A GFT has m^h

nodes, and $w^l m^{h-l}$ switches at level l. As a result, the total number of switches is:

$$\sum_{l=0}^{h-1} w^l m^{h-l} = m^h \sum_l \left(\frac{w}{m}\right)^l = m^h \frac{1-\left(\frac{w}{m}\right)^h}{1-\frac{w}{m}} = \begin{cases} m\frac{m^h-w^h}{m-w}, & \text{if } m \neq w \\ hm^h, & \text{if } m = w \end{cases}$$

In practice, assuming uniform port count for all switches in a network, GFTs are detrimental since top-switches necessarily have unused ports. This issue and others can be solved by *Extended Generalised Fat-Trees* (XGFTs), introduced alongside GFTs, which provide a per-level definition of arities.

Definition 25. *A graph defined by the formula*

$$XGFT(h;\ m_0,\ldots,m_{h-1};\ 1,w_1,\ldots,w_{h-1})$$

is an XGFT with h levels, with each switch of level i connected to m_{i-1} switches below and to m_i switches above. It is constructed recursively from m_{h-1} distinct copies of $XGFT(h-1;\ m_0,\ldots,m_{h-2};\ 1,w_1,\ldots,w_{h-2})$, its topological groups.

This definition provides even more flexibility in terms of oversubscription. For example, one might decide to provide more bisection bandwidth in the lower levels than in upper levels if few applications are expected to span across topological groups. As a practical example, this is the case in some BXI clusters, connected as fat-trees pruned only in the top level. Experience at Atos has shown that other than benchmarks, few current user applications make strong use of the total bandwidth provided by the interconnect.

XGFTs also allow designing fat-trees with both constant CBB, often referred to as rearrangeably non-blocking, and fully-plugged fixed-radix switches. These graphs respect the following formula:

$$XGFT(h;\ k,\ldots,k,2k;\ 1,k,\ldots,k)$$

The same topology is also called m-port n-tree [52], with $m = 2k$.

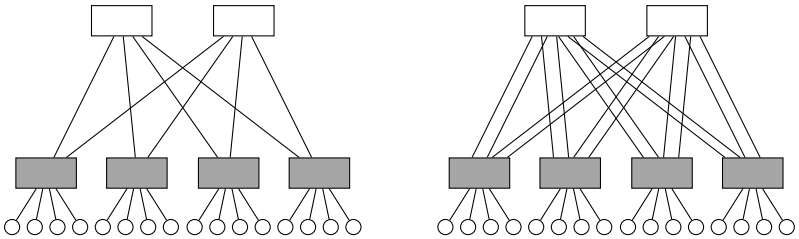
PGFTs, RLFTs

Parallel Ports Fat-Trees, or *Parallel Generalised Fat-Trees* (PGFTs), are an extension to XGFTs allowing for multiple interlinks [95]. The definition allows for one value of "parallel ports" per level:

Definition 26. *A graph defined by the formula*

$$PGFT(h;\ m_0,\ldots,m_{h-1};\ 1,w_1,\ldots,w_{h-1};\ p_0,\ldots,p_{h-1})$$

is an h level PGFT, with each switch of level i connected to m_{i-1} switches below (using p_{i-1} interlinks) and to m_i switches above (using p_i interlinks).



(a) $XGFT(2;4,4;1,2)$ (b) $PGFT(2;4,4;1,2;1,2)$

Figure 1.6: Best efforts to build a constant-CBB network with 16 nodes and fully-plugged 8-port switches using XGFT and PGFT topologies.

This allows more link redundancy, but more importantly greater flexibility in network design. In particular it allows describing networks with constant CBB for more configurations than XGFTs can. A simple example is provided in Figure 1.6, which illustrates that a constant-CBB network with 16 nodes and fully-plugged 8-port switches can be designed using PGFT notation, but not using XGFT notation.

Real-Life Fat-Trees (or RLFTs) are defined alongside PGFTs to account for several constraints often expected of fat-trees in real life [95].

Definition 27. Real-Life Fat-Trees *(or RLFTs) are constant-CBB PGFTs with all switches having the same radix and being fully plugged, and each node being connected to one leaf through one link.*

This is identical to the requirements of m-ports n-trees, with the added possibility of specifying the number of parallel ports at each level. In some cases, however, RLFTs are equivalent to m-ports n-trees. An RLFT with h levels, K-port switches, and a vector p describing parallel links, has the corresponding PGFT definition:

$$PGFT\left(h;\quad \frac{K}{2},\ldots,\frac{K}{2p_i},\ldots,\frac{K}{p_{h-1}};\quad 1,\ldots,\frac{K}{2p_i},\ldots;\quad \ldots,p_i,\ldots\right)$$

The PGFT in Figure 1.6.b is an 8-port 2-level RLFT with $p = [1,2]$.

In this thesis, the names *fat-tree*, or *regular fat-tree*, will be used for any topology that can be described as a PGFT. This includes Leiserson fat-trees, k-ary n-trees, GFTs, XGFTs, RLFTs.

1.3.4 . Irregular fat-trees (IFTs)

Terrain observation of real systems shows that clusters labelled as fat-trees often break properties usually associated with fat-trees. This happens mainly either because fat-tree connection rules are relatively difficult to respect, or in order to attempt improving the performance-to-cost ratio of the network. In this thesis, fat-tree-like networks are categorised as IFTs (a novel nomenclature) if any of the following irregularities applies:

- Multiple down-switch-paths between a top switch and a leaf;
- Variations in arity between switches in the same level (variations due to failures do not strongly apply);
- Connections within a level or skipping a level;
- Leaves in levels higher than others.

Two formally defined topologies which count as IFTs are described below. However, the general case of IFTs does not correspond to formally described topologies, and instead is at best described as fault resilience, as shown in Sections 1.2.9 and 1.4.6.

QFTs

Zahavi defined a slight variation to PGFTs that allows building networks with wider subgroups for a given number of nodes [98]. This *Quasi Fat-Tree* (QFT) topology is defined similarly to PGFTs, but with the p vector referring to cross-connections instead of interlinks: $p_l = 2$ means that at level l, pairs of groups are linked together. QFTs are defined with the assumption that there is only one level l at which $p_l > 1$, with generally $l = 1$. Connecting two groups together means that every lower element in that group has two distinct up-switch-paths to every top switch, instead of only one. This breaks a core property often associated with fat-trees, making load balancing routing more difficult to express. On the other hand, this is presented as an opportunity for greater fault-tolerance through greater switch-path diversity.

MegaFly

An indirect variant of the DragonFly, called *MegaFly* (or *DragonFly+* by Mellanox), uses complete bipartite graphs for the intra-group cabling [10, 79, 28]. The inter-group cabling rules are not stricter than for DFs (there must be at least one direct link connecting each pair of groups). Two advantages of the MF topology over DFs are the increased scalability and path diversity, at the cost of twice the number of switches and an increased average hop count. In this topology, non-leaf switches (called roots) are connected together through samelinks. This topology is a BMIN, and therefore accommodates up*–same?–down* routing in a deadlock-free manner (see Property 2). Minimal routing is

up*–same?–down* in MFs, and therefore deadlock-free, unlike minimal routing in DFs.

Similarly to DFs, literature has focused on non-minimal routing to overcome low effective bandwidth in MFs for adversarial traffic (potentially resulting from natural job placement). This requires use of multiple VCs to guarantee deadlock-freedom, since consecutive samelink hops incur chainable channel dependencies within the top level.

An alternative strategy to achieve high performance for various job sizes is for the job placement to scatter nodes between groups while using only minimal routing. This increases the average minimal distance between nodes of a job, and risks increasing inter-job interference for the use of inter-group links, but avoids spreading congestion to other groups.

1.3.5 . Topologies used by interconnect vendors

Other major HPC interconnects nowadays are Infiniband (mostly provided by Mellanox), Cray Slingshot, Intel Omni-Path, and Fujitsu's Tofu. Apart from Tofu, all of the above are actively used for indirect topologies and use some form of static routing combined with some adaptive behaviours. The choice of topology is dictated more by client need than by vendors, and in practice common indirect topologies are FT and MF, and common direct topologies are DF, Torus, and SlimFly. The latter are treated very differently from the work in this thesis. The most powerful supercomputers listed in the Top500 tend to have indirect topologies to accommodate for comfortable bandwidth on shortest-path routing, but trends evolve rapidly when taking into account equipment cost and energy consumption compared with actual application requirements. A majority of the bulk of Top500 supercomputers is accordingly built using commodity Ethernet equipment, with Infiniband still being used in a large portion of the remaining systems, followed by the other HPC interconnect vendors focused on very high end configurations. Data centers arguably now form a more important source of very large systems than HPC, and there is more and more overlap in the resulting technologies, with some progressive convergence of techniques.

1.4 . Routing algorithms for fat-trees

Here are presented several existing *topology-specific* algorithms specifically designed for fat-trees, with their advantages and their limitations. As shown in Section 1.3.2, up–down routing functions in fat-trees are deadlock-free. Some discussion is provided regarding fat-trees with failed links, but a more in-depth overview of fault resilience is provided in Section 1.2.9.

There are other algorithms for fat-trees which can be found but which target other functionalities not specifically focused on here, such as switch-to-switch routing [8], link-weight aware routing [99], or multi-homed fat-trees [69].

1.4.1 . Ftree

Zahavi defines a greedy algorithm to route fat-trees in a perfectly load-balanced manner for shift-pattern traffic [97]. Improvements to its resilience are often considered part of its definition [6, 7]. It is applicable to any BMIN in which every pair of leaves has at least one up–down path. One of its immediate advantages is that it does not require perfect PGFTs to generate valid routing. For this reason, it is implemented in the widely used OpenSM subnet manager (compatible with Infiniband systems) under the name Ftree. Figure 1.7 illustrates Ftree in an example fat-tree and with one link missing.

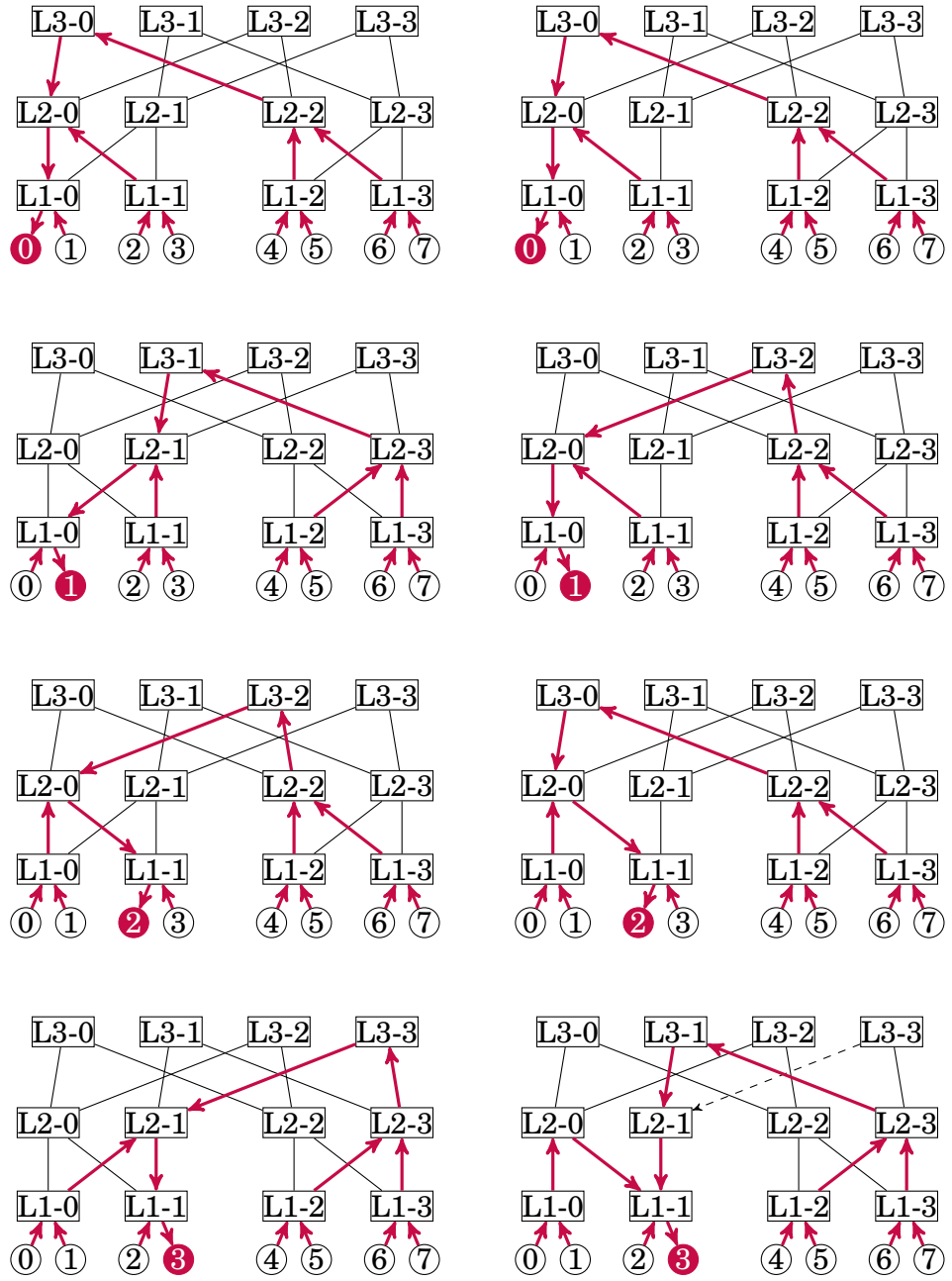
Ftree explicitly coalesces all traffic to the same destination node by traversing a tree within the fat-tree starting at the destination and eventually reaching every other node (viewed as a potential source). Each traversed link is routed in its opposite direction: the routing table of the next switch is updated for the routed destination with its output port into which the step was performed. No switch can be traversed twice thanks to an explicit check of the corresponding entry in its routing table. The pseudo-code is provided in Algorithm 1. As defined in [97], Ftree actually requires every top switch to have a direct down path to every leaf; in an XGFT, this means that any link failure in the top level renders the network unroutable. Here, Ftree is defined closer to its actual implementation in OpenSM (with the Δ counter and the loop at line 10) to consider fault tolerance explicitly.

Load-balance between routes to different destinations is attempted thanks to a usage counter for each upport at each switch. Every time the least used upport of a switch is traversed (in a primary iteration, as defined in line 1), its usage is incremented. For example, in Figure 1.7b, for destination 3, the upport of L1-1 connected to L2-0 is traversed since the other upport does not allow reaching all sources; however its usage is not increased. Alternately, the same behaviour can be implemented using an index corresponding to the least recently used upport at each switch.

The traversal goes as far down as possible before going up to guarantee shortest paths among the traversed switches (in a degraded or perfect PGFT, this is not necessary).

Limitations of Ftree

Ftree generates load-balancing routing in a PGFT and is also applicable to PGFT-like topologies, however the load-balancing procedure does not degrade gracefully to irregular fat-trees, and it is much more susceptible to quality loss when faced with few link losses, than are agnostic algorithms [7]. This is simply due to the load balancing method expecting perfect PGFTs: Ftree does not take into account the specific topology to balance load, only to make sure routing is correct. Figure 1.7b illustrates how a single missing link can result in imperfect load balance. (Moving routes to node 2 through top switch L3-3 instead of L3-0 would have resulted in equal usage of top level links.) Furthermore,



(a) Routes in the complete 2-ary 3-tree.

(b) Routes under failure of L1-0 ↔ L2-1. There are two distinct routes going L3-0 → L2-0 and none going L3-3 → L2-1.

Figure 1.7: Ftree routes in a 2-ary 3-tree towards the first four destinations. Dashed arrows represent “secondary” routes that are computed but never used.

Algorithm 1: Ftree

```
1  foreach upport u of each switch s ∈ S do
2      Reset the usage counter of u
3  foreach leaf switch l ∈ L do
4      foreach endport p of l do
5          n is the node connected to p
6          δ is the NID of n
7          Δ ← #L  /* Remaining number of leaves to route towards
                      δ */
8          Ascend(l,δ)
9  Function Ascend(s,δ)
10     foreach upport u of s in ascending order of usage do
11         a is the remote switch (above s) connected to p
12         if δ ∉ LFT_a then
13             d is the remote (down) port (of a) connected to u
14             LFT_a[δ] ← d   /* Set (down) route of a to δ via s */
15             if in the first iteration of l. 1 then
16                 Increment the usage counter of u
17             Descend(s,δ)
18             Ascend(a,δ)
19             if Δ = 0 then
                           /* All leaves have been routed towards δ */
20                 Break
21 Function Descend(s,δ)
22     if s ∈ L then Δ ← Δ − 1
23     foreach down group g of s do
24         b is the remote equipment (below s) connected to g
25         if b ∈ S and δ ∉ LFT_b then
26             d ← the least used (down) port in g
27             u is the remote (up)port (of b) connected to d
28             LFT_b[δ] ← u      /* Set (up)route of b to δ via s */
29             Descend(b,δ)
30             if Δ = 0 then Break
```

and counter-intuitively, missing nodes also strongly degrade load balance [6]. In oversubscribed fat-trees, Ftree results in overlapping routes to different destinations (with the same overlapping factor as the blocking factor); using multiple VLs helps reduce the resulting pattern-dependent imbalance [38].

1.4.2 . Dmodk

The Dmodk routing algorithm and corresponding PGFT topology are described in detail in [95]. Dmodk is equivalent to Ftree in undegraded PGFTs (except that, unlike Dmodk, Ftree only computes primary routes in an undegraded PGFT). Zahavi later referred to both algorithms as Dmodk. In this thesis, Dmodk refers only to the closed-form algorithm presented here, applicable only to perfect PGFTs, as it was implemented in early versions of BXI FM for example.

The algorithm relies on a closed-form address-based criterion to determine whether a switch S_a is down-reachable from switch S_b and, if so, which ports P_b can be selected as output port:

$$\mathscr{D}_P(S_a,P_b) = \begin{cases} (l,a_h,a_{h-1},\dots,a_l,\dots a_1),\ (k,b_h,b_{h-1},\dots,b_l,\dots b_1,q) \mid l<k \\ \text{and } a_j = b_j\ \forall\ k<j\leq l \\ \text{and } a_k = q \bmod m_k \end{cases}$$

Otherwise, an arithmetic formula defines the upport (with index p) to select:

$$p_{\text{up}} = \left\lfloor \frac{d}{\prod_{k=1}^{l} w_k} \right\rfloor \bmod (w_{l+1}p_{l+1})$$

The level-wide constants (or *arities*) w_l and p_l respectively denote the numbers of uplinks and of interlinks of all switches at level l. With this formula, the routes to each destination are coalesced as early as possible, and routes to different destinations are spread out as much as possible. We can justify this strategy by modelling collisions between different routes to the same destination as being unavoidable by the routing function: it then follows that a reasonable balancing strategy is to avoid collision between routes to different destinations, which requires maximising collisions between all routes to each destination. A more formal justification is provided in Chapter 2. This distribution of routes is illustrated in Figure 1.8; notice how all secondary routes are computed. These closed-form steps rely on a given organisation of addresses of switches and indexing of their ports. Dmodk has very low complexity and is a perfectly parallel routing algorithm for PGFTs, but it is not applicable to degraded PGFTs or irregular fat-trees.

Gomez [36] routes k-ary n-trees with a method which applies bitmasks to the destination number. This method is defined in detail for $k = 2$; a sim-

ilar approach can be extended for higher values of k. This algorithm can be considered as a specialised version of Dmodk routing.

1.4.3 . Smodk

In some systems, the source of each message, or its input port in each switch, is known to the routing function. From this a symmetric alternative to Dmodk can be defined: Smodk, which propagates messages similarly to Dmodk but based on source node ID rather than destination ID [73]. This algorithm concentrates together routes from the same source, thus maximising the number of same-source collisions.

Here we define Smodk using input port notation, with i referring to the index of the input port of a message. For upward routes, the output upport p_{up} is determined using i', the index of i among the ordered set of downports of the switch:

$$p_{\text{up}} = \left\lfloor i' \right\rfloor \bmod (w_{l+1}p_{l+1})$$

Smodk is not defined specifically for PGFTs, so downport selection in the case of parallel ports is not precisely defined. One reasonable method is to select the downport among authorised downports similarly:

$$p_{\text{down}} = \left\lfloor i' \right\rfloor \bmod p_l$$

An example application of Smodk is shown in Figure 1.9.

In cases where communications are symmetrical between patterns with several destinations per source and those with several sources per destination, there is no reason for Dmodk or Smodk to be better than the other. Otherwise there isn't necessarily one choice which is always best, but choosing Smodk for multiple-destination-heavy patterns (and Dmodk for multiple-source-heavy patterns) is a reasonable heuristic [73].

We refer to Dmodk and Smodk together as a class of algorithms as Xmodk for the rest of this thesis.

1.4.4 . Random shortest path routing (*RandSP*)

One approach to balancing the load of deterministic routes in a fat-tree is to randomly choose minimal routes. Random shortest path routing can be used here, see Algorithm 2. As hinted in the algorithm, it is possible to implement a fast criterion for shortest paths in precisely defined fat-trees with known topological addresses: in the same way that there is a simple rule to determine the NCAs between two nodes, it is equally easy to determine either the upports of a switch leading towards the NCAs (in a shortest path) or the downports leading towards the destination (also in a shortest path). This is used in existing literature regarding random routing for, or applicable to, fat-trees [37, 29, 73]. In fat-trees, shortest paths are up–down, therefore random shortest path routing is deadlock-free in fat-trees. Compared with other shortest path

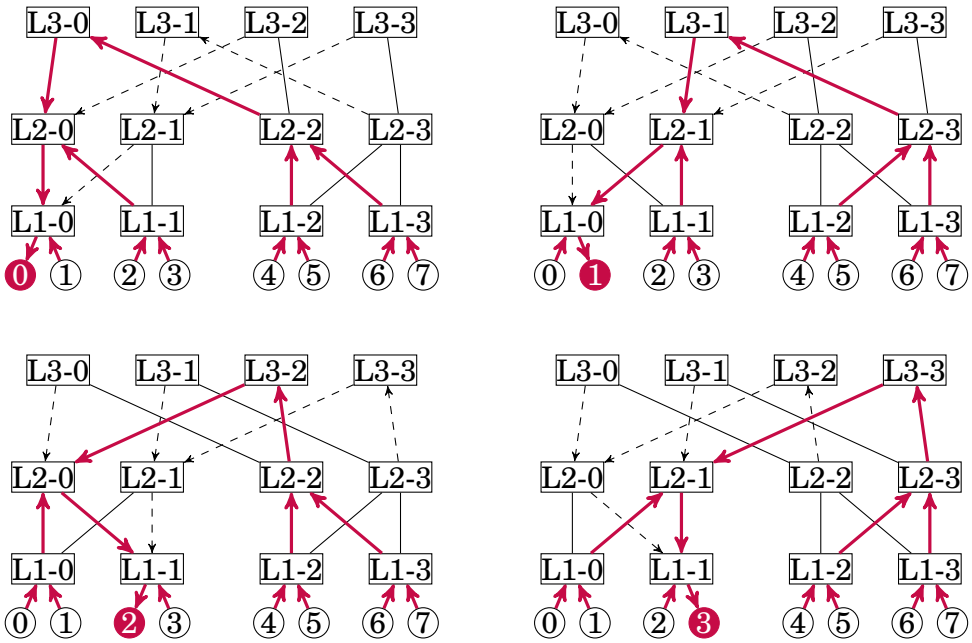


Figure 1.8: Dmodk routes in a 2-ary 3-tree towards the first four destinations. For destination 1 at switch L2-3, the first uplink is selected: this illustrates the pre-modulo integer division ($\lfloor \frac{1}{2} \rfloor \bmod 2 = 0$).

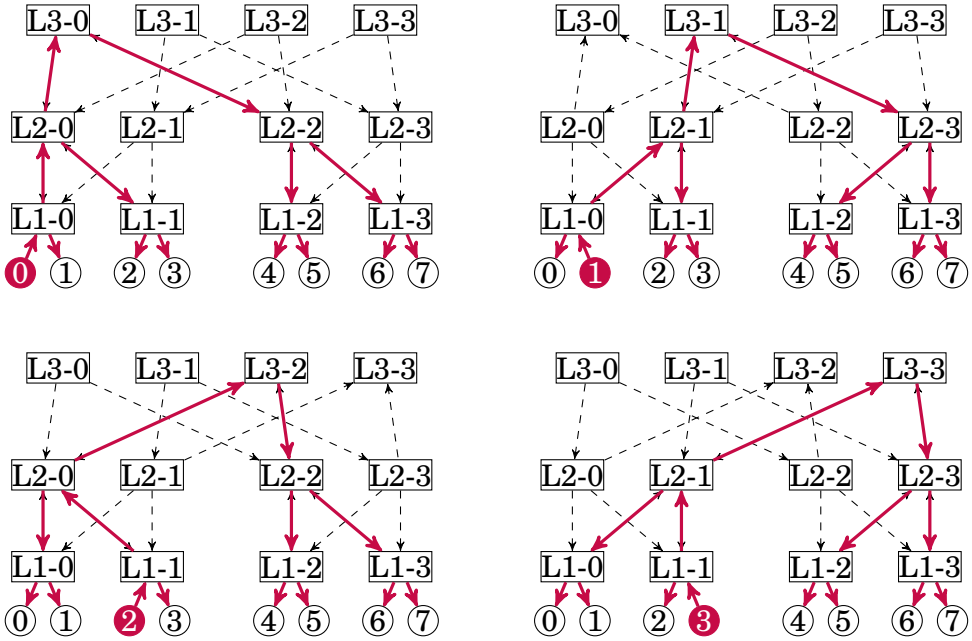


Figure 1.9: Smodk routes in a 2-ary 3-tree from the first four sources.

algorithms, random shortest path routing tends to be simple to implement, fast to execute, and easy to parallelise.

On average, routes are randomly load-balanced: all-to-all traffic will not cause implicit bias towards any part of the topology. Deviations from the average will, however, cause routes to overlap and induce network congestion. Statistically, these deviations are very common and in practice, RandSP provides low-quality routes compared with systematically load balancing routing. This will be studied in more detail in Section 2.

Algorithm 2: Random shortest path routing

```
1 foreach switch s ∈ S do
2     foreach node n ∈ N do
3         P ← ports of s on the shortest path to n (e.g. using topological
            addresses in a fat-tree or the Dijkstra algorithm)
4         δ is the NID of n
5         LFT_s[δ] ← random port among P
```

1.4.5 . Ranking

Fat-tree algorithms require knowledge of switch levels, or at least link direction. In real subnet managers, this information is automatically determined after discovery of the network, in a step sometimes called *ranking* or simply discovery. During this step, it is possible to find topological inconsistencies (forbidden links) in case the manager has specific requirements. Algorithm 3 allows ranking upwards, starting at leaf switches, while checking for topological inconsistencies (for example, if two leaf switches are connected together, the check at line 11 will find the inconsistency). If instead top switches are known, ranking downwards can be done using the same algorithm, by swapping rank for $(l, h-l-1)$ level pairs afterwards.

Generally speaking, there is little discussion in the existing literature regarding ranking. One exception is an article about the application of OpenSM algorithms to degraded fat-trees [7], which mentions how orphaned leaves are virtually flipped into high levels by the ranking of Ftree; the recommended solution is to manually tag each switch so as to force their rank in the tree. Furthermore, the documentation of OpenSM [62] describes the ranking strategies used in its Fat-tree specific algorithms (Ftree and UPDN). These are as follows:

- ranking upwards starting at leaf switches (corresponding to level 0) if the topology is a “pure” fat-tree (defined in terms of coherent arities across levels, which includes QFTs, a type of IFT);
- ranking downwards starting at manually provided top switches;

Algorithm 3: Rank upwards starting at leaf switches

```
1  rank leaf switches to level 0
2  T ← L
3  while T ≠ Ø do
4      N ← Ø
5      foreach s ∈ T do
6          foreach neighbour s′ of s do
7              if s′ is unranked then
8                  rank s′ to level of s + 1
9                  N ← N ∪ {s′}
10             else
11                 check s′ is ranked directly above or below s
12     T ← N
```

- ranking downwards starting at statistically determined leaf switches.

Downward methods become necessary if at least one leaf switch is expected to be higher than others. The latter (statistical) method is defined as follows (note *CA*s correspond to channel adapters, interchangeable with compute nodes, *node*s correspond to switches, and *root node*s correspond to top switches):

> Auto-detect root nodes — based on the CA hop length from any switch in the subnet, a statistical histogram is built for each switch (hop num vs number of occurrences). If the histogram reflects a specific column (higher than others) for a certain node, then it is marked as a root node. Since the algorithm is statistical, it may not find any root nodes. The list of the root nodes found by this auto-detect stage is used by the ranking process stage.

In practice, this statistical method is prone to failure in case of topology irregularity, with a second column appearing as high as the desirable one. A novel resilient alternative is provided in Annex 2.

1.4.6 . Fat-tree-specific fault resilience

Some of the research regarding oblivious fault-resilient routing focuses on techniques that explicitly target degradations to regular fat-trees [98, 68]; there are several re-routing strategies for these techniques. The UPDN [63] and Ftree [97] routing engines available in OpenSM can also be applied from scratch to a degraded fat-tree. PQFT [98] is similar, though it requires a complete list of faults. The combination of Dmodk as an offline algorithm and Ftrnd_diff as an online algorithm (available in older versions of BXI FM [85]) aims at fast reaction to faults with minimal routing changes, by relying on an iterative

list of network changes and an up-to-date view of the network. A very similar method is described by Xu et al. [88], with an observed performance degradation described as graceful but apparently quite significant. Fabriscale [86] also provides fast centralised re-routing of fat-trees, by precomputing alternative routes. Another similar approach through fast hardware reaction is Mellanox's SHIELD technology [58].

The random operation chosen in Ftrnd_diff results in progressive degradation of load balance and incapacity to return to the original routing in case of fault recovery. Ftrnd_diff does manage to recover rapidly from minor failures; however, large numbers of simultaneous changes (which happen for example when entire islets are rebooted) cause computation times to skyrocket in current implementations. The strategies of PQFT, Fabriscale and SHIELD which consist of moving only invalidated routes let one expect somewhat similar load-balancing issues as with Ftrnd_diff. Studies show topology-agnostic routing outperforms fat-tree-specific routing under sufficient topology degradations [7, 20].

1.5 . Problem statement and plan

This overview of context has shed light on several aspects of the design of supercomputers. We find that the interactions between some of these aspects are lacking in the current context. Indeed, we first find that quality evaluation methodologies do not provide meaningful metrics without arbitrary simulation or experimentation. We also find that current resilient fat-tree-specific routing algorithms do not provide high-quality routing under realistic amounts of equipment failure. Furthermore, we find that existing fat-tree-specific routing algorithms do not provide good quality routing in the case of node heterogeneity. Finally, we find that there is no research targeting better-than-agnostic routing for the common case of strongly irregular fat-trees.

Correspondingly, Chapter 2 will present techniques to comparatively estimate the quality of routing algorithms, including a partly novel one. The following sections will each present quality analyses using these tools. Chapter 3 will take on a case study for a specific quality problem observed with the otherwise state-of-the-art routing techniques in some heterogeneous systems, and present a solution which can extend many existing algorithms. Chapter 4 will present a novel approach to high-quality fault resilience in a common target topology. Multiple novel approaches to routing irregular fat-trees will be presented in Chapter 5.

$\mathcal{O}$	is the big O notation
#	denotes cardinality
λ_n	is the (only) leaf switch connected to node n
$\rightarrowtail, \rightarrowtail$	respectively denote down and up links, according to rank
G_s	is the ordered list of port groups of switch s
Ω_g	is the remote switch connected to port group g
	When considering a single topology:
S	is its set of switches
L	is its set of leaf switches ($L \subset S$)
N	is its set of nodes, indexed by NIDs
E	is its set of edges
C	is its set of channels

Table 1.1: Topological notations

2 — Quality comparison of routing algorithms

2.1 . Introduction

Once the routes of a network are guaranteed to be valid and deadlock-free, we can start studying the resulting network performance. This chapter first covers some existing models used for such studies, and then provides some iterative contributions.

Network performance can be defined directly based on the speed of execution of target applications, or using other network criteria such as bandwidth and latency. Specific measures such as the final run time are good measures insofar as they precisely describe a use case's performance and, if the use case is precisely defined (such as a single application or set of concurrent applications), they have final say. In case of strong run time variability, which can arise depending or not on concurrent applications, care should be given to characterising it. In case of concurrency, care should also be given to determining what compound performance metric best reflects desired behaviour (such as overall performance or fairness). Generic measures such as effective network latency and throughput can be seen as attempts to characterise the network independently of a specific use case (either topology details or communication scheme) [24]; they can also correspond to performance targets other than specific application run time, such as overall system efficiency. Literature relating to network design and evaluation often consider specific metrics based on these generic ones, such as *traffic* (measured throughput as a fraction of network capacity) and measured latency-traffic graphs (with applied load as a synthetic input parameter); these allow characterising the network rather than the system as a whole, while being specific to a defined system.

The most generic measures that take into account the routing function are *static metrics*, based on the number of paths going through ports (e.g. the edge-forwarding index [39]). Such a number can be understood as a model for potential load under uniform traffic, and under the assumption of sustained traffic as an estimate of congestion risk.

2.2 . Traffic simulation

Regardless of the type of study considered, choosing the type of traffic to simulate is important since the quality of a routing function is strongly dependent on the traffic then applied. This section presents a few models to simulate traffic commonly found in existing research.

2.2.1 . Static traffic patterns

All to all (A2A) Some research simply considers all routes possible. For example, the original article defining the edge-forwarding index computes it for every possible route in the network. Similarly, the article presenting Nue applies the edge-forwarding index (only on switch–switch ports) for every node–node route in the network [21].

Scatter and gather Applications commonly distribute data from a single node; accordingly they retrieve computation results to the single node. The corresponding pattern matrices are either a single column or row, placed at the index of the single node. This is sometimes called a hotspot pattern for the exacerbated capacity to congest in a single point of the network.

Random A simple model of random communication commonly used is for each node to send messages to a random destination. Note that with this pattern there may be destinations with multiple sources, or incoming flows, and conversely others with no source (i.e. the traffic pattern matrix is not generally a permutation matrix). Another approach could be to use a matrix with ones randomly placed; to the best of my knowledge there is no such communication pattern studied in existing research.

Random bisection One model used to estimate *effective CBB* [40, 75] is by randomly splitting the set of nodes into two equal halves and randomly computing pairs between all elements of both halves. Communications can either be only from one half to the other (unidirectional) or between both halves (bidirectional). In the latter case, the traffic pattern matrix is a permutation matrix. For a network with n nodes, there are $\frac{1}{2}\prod_{i=\frac{P}{2}}^{P} i$ possible bisect communication patterns; this is unmanageably large and random samplings are used instead of exhaustive enumeration. Generally, the sample size is not precisely justified but statistically significant.

Random order ring (ROR) A model of communication which is closer to the behaviour of many applications is to define a ring (or n-cycle) of nodes communicating between neighbours in either (unidirectional) or both (bidirectional) directions. The order of the ring is chosen at random to model systems in which application node order is not optimised to correspond to the routing technique. Unidirectional RORs are permutations.

Shift permutation (SP) Shift permutations are a generalisation of ring patterns[1].

Definition 28. *In the k-shift permutation (SP_k) of a network with N nodes, each source s sends a message to the node with rank k, according to the following function:*

$$SP_k(s) = (s + k) \bmod \#N$$

Shift permutations are particularly interesting because global collectives can be implemented efficiently as a sequence of shift permutations [45]. Furthermore, for networks with rearrangeable optimality (e.g. Clos networks, PGFTs), there is a static routing function which is optimal for all shifts. Shifts are the simplest class of equivalent permutations.

Shift permutations are only interesting given a global node numbering, since shifts on a random numbering are equivalent to random permutations.

Application traffic pattern matrix Instead of standard patterns, custom matrices corresponding to a specific application can be used if they are known in advance, which is only the case for applications with deterministic communication behaviour. Care should be taken to respect the node ordering used by the routing algorithm and the one used by the application's communication layer.

Topology-specific adversarial traffic In certain topologies, some traffic patterns are famously difficult to sustain in a performant way. For example, DFs have few links connecting each pair of groups (in comparison with fat trees), hence a static pattern which stresses this situation is for each node to send to a destination in the next group. With a natural node ordering, these patterns represent a subset of shift permutations.

[1] Unidirectional rings correspond to either SP_1 or $SP_{-1} = SP_{\#N-1}$. Furthermore, shifts are rings *iff* $gcd(k,n) = 1$.

2.2.2 . Dynamic traffic simulation

Another class of traffic simulation takes into account temporal or dynamic behaviour as well. Examples include communication traces recorded from real application runs (temporal order is used to determine message dependencies), as well as synthetic traffic generation schemes such as uniform random traffic from all nodes with a given load ratio, or hotspot traffic (in a gather pattern) to a single destination with a given load ratio as well. Detailed simulations integrate details such as individual message size or target type.

2.3 . Static metric

This thesis focuses mostly on oblivious routing techniques, therefore static metrics seem adapted to describe the quality of routing since they use similar data to that available to the routing function. As described previously, static metrics are based on counting paths crossing each port for a set of routes. The simplest such metric is the edge-forwarding index metric:

Definition 29. *The edge-forwarding index (noted ξ) of a network, is the maximum number of paths crossing any single edge for all routes in a network [39].*

On all switch–node (resp. node–switch) edges, all paths to (resp. from) the node are counted; these values are generally much higher than at any switch–switch edge without being affected by the routing function. This effectively prevents the study from distinguishing routing functions, a simple alternative is therefore to count only switch–switch edges (resulting in a metric noted Ξ). In either case, the complexity of determining this edge-forwarding index is of the order of traversing all routes and the memory requirement is of the order of one integer (large enough to store up to the number of routes considered) for each port.

The number of paths traversing each port models a maximal theoretical load applied to the port in case of sustained message flow without taking into account how the rest of the network would throttle each flow. In this sense, the edge-forwarding index relates to sustained flow situations at a simpler level than queueing, flow, or Markov/Petri models. Choosing to focus on a simple static metric means disregarding performance effects due to dynamic behaviour of communications and local behaviour of congestion propagation. Instead, static metrics represent a simple model of worst-case scenario congestion risk based on maximal theoretical load. Comparing risks for the same traffic patterns on different routing functions provides a simple estimate of contention risk under sustained traffic; in practice, this reflects actual comparative experimentation with representative situations [27]. One approach to explain comparative static metric analysis could be to define a congestive value estimating the ratio between edge capacity over node production or consumption capacity; from this a scenario with a static metric greater than the congestive

value can be generally considered of lower quality than one with a metric below this value (as a simple predictive model of congestion happening at all). This would limit analysis to cases with differing comparisons to the congestive point and be specific to network characteristics, while a simple comparison of static metrics without scale or reference allows comparison between any routing functions. Throughout the remainder of this thesis, a simple static metric applied to two generic traffic patterns (all presented in more detail in the upcoming sections) are used primarily for quality comparisons of routing algorithms on test network topologies.

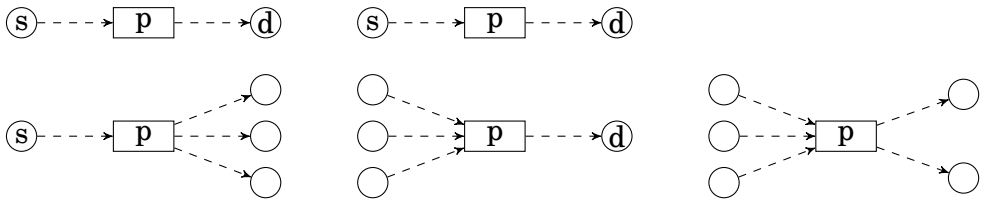
We will note $f_p(P)$ the congestion metric f applied to the edge/port p with the set of paths P. The maximum value across a network $G(S,N,E)$ is $f_G(P) = \max_{p \in E}(f_p(P))$. The set of paths for $A2A$ under the routing function R is $A2A(R)$. The edge-forwarding index previously described is therefore $\xi_G(A2A(R))$. For example, a fully connected network G_F of 5 switches and 10 nodes (with 2 nodes connected to each switch) under shortest path routing R_S has the following edge-forwarding indices:

$$\xi_{G_F}(A2A(R_S)) = \#N - 1 = 9$$
$$\Xi_{G_F}(A2A(R_S)) = \left(\frac{\#N}{\#S}\right)^2 = 4$$

The ξ value is attained at each node–switch (resp. switch–node) edge since all the other nodes are destinations (resp. sources) of paths crossing the edge. The Ξ value is attained at each switch–switch edge since it carries all paths from the nodes connected to the incoming switch, to the nodes connected to the outgoing switch.

Adding to the example a routing function R'_S, defined from R_S by moving a single path to go through any extra switch, the two edges receiving the extra path see their ξ value increased by one. In this case we can see how only Ξ reflects the increased risk of congestion and therefore how it can help compare routing functions better than ξ:

$$\xi_{G_F}(A2A(R'_S)) = 9$$
$$\Xi_{G_F}(A2A(R'_S)) = 5$$



(a) $s = 1$ (b) $d = 1$ (c) $s > 1$ and $d > 1$

Figure 2.1: Cases of s sources and d destinations routed through a port p illustrating the maximal theoretical flow model inducing $\boldsymbol{\mu}_G(P)$.

2.3.1 . The μ static congestion metric

An improvement over the edge-forwarding index metric is the $\min(src, dst)$ metric (here noted $\boldsymbol{\mu}$). It was introduced for a branch and bound routing algorithm [72]. This metric follows the idea that the number of flows estimated by the edge-forwarding index does not correspond to an attainable sustained load when taking into account the production and consumption rates of each node. Under the assumption that all nodes are identical in capability and that their production rate is equal to their consumption rate, the estimate for maximal sustained load, or congestion risk, between s sources and d destinations can be reduced as follows:

- If $s = d = 0$, there is no possible flow. (Note that one cannot be nil while the other is not.)
- If $s = 1$ (Figure 2.1a), then d does not affect the maximal load, equal to the production rate of the source node. Furthermore, assuming uniform production rate and port capacity greater than node production rate, no port subject to only these flows will contend.
- If $d = 1$ (Figure 2.1b), then s does not affect the maximal sustained load, equal to the consumption rate of the destination node. Potential contention in case the combined production rate is greater than the port's capacity can be disregarded since it is lesser than the HoL blocking coming from the destination under sustained load.
- If $s > 1$ and $d > 1$ (Figure 2.1c), the maximal load is greater than each node's capacity. More specifically, the maximal load is $\min(s, d)$ times greater than node capacity.

From this model of flow congestion, we can compute the maximal theoretical load using each port's number of sources and destinations for a given set of paths P (respectively $src_p(P)$ and $dst_p(P)$) as:

$$\boldsymbol{\mu}_p(P) := \min(src_p(P), dst_p(P))$$

Correspondingly for a complete network $G(S,N,E)$:

$$\boldsymbol{\mu}_G(P) := \max_{p \in E}(\min(src_p(P), dst_p(P)))$$

Interestingly, this metric brings a similar benefit to Ξ: switch–node and node–switch edges do not contribute to $\boldsymbol{\mu}_G$. This reflects the idea that potential congestion at these edges cannot result from sub-optimal routing.

A limitation of this metric is that it does not take into account how flows throttled at one port will not contribute as much to other ports' maximal load. Designing a static routing algorithm optimised for this metric is nonetheless a reasonable approach to designing a routing algorithm well-balanced in the worst case for the considered communication pattern. Each port's number of sources and destinations must be known at the same time in order to compute this metric; it therefore requires two passes over every route followed by one pass over every port (to get its minimum), with at least two integers per port. This remains inexpensive compared with the complexity of most routing algorithms.

2.3.2 . Generic static traffic patterns of choice

Without knowledge of real traffic behaviour, a simple approach is to compute $\boldsymbol{\mu}_G(A2A(R))$, considering the worst possible communication pattern. However, real traffic in supercomputers is not well represented by the A2A pattern: since networks are at best designed to accommodate any permutation (like Clos telephone networks), it is strictly more efficient to decompose collective communications into sequences of permutations (using communication trees decomposed in Bruck patterns for example) than to push every point-to-point communication into the network [40]. In practice, this is always done by at least one communication layer. Accordingly, we limit our study of congestion risk to permutation patterns only. For network $G(S,N,E)$, we denote $\mathfrak{S}_N$ the set of all permutation patterns (the symmetric group of N).

Not only is studying permutations more realistic than the all-to-all communication pattern, it is also more useful to understand differences between routing functions. Indeed, A2A cannot be optimised more than spreading traffic in some way, but each permutation might perform very differently depending on routing function.

An immediate collateral benefit to this specialisation is that for any permutation, every node has exactly one source and one destination (potentially itself) and, as a result, we have the following property:

$$\forall P \in \mathfrak{S}_N, \quad \forall p \in E, \quad \boldsymbol{\mu}_p(P(R)) = src_p(P(R)) = dst_p(P(R))$$

This allows simplifying computation of the $\boldsymbol{\mu}$ metric to the same complexity as the Ξ metric.

Unlike A2A, $\mathfrak{S}_N$ is a set of $\#N!$ patterns that cannot be studied exhaustively. This raises two issues: how to sample meaningful patterns, and how to reduce the results to one or few meaningful values.

Random permutations

The first (oblivious) choice of sampling is random sampling. This can be used with the aim of representing typical behaviour, in which case a reasonable approach is to compute statistics: mainly the median value which can be complemented with 1st and 3rd quartiles (or 1st and 39th 40-quantiles to estimate a 95% credible interval). We note the median value for a random sampling $RP(r)$ of r permutations as follows:

$$\boldsymbol{\mu}_G^{RP(r)}(R) := \underset{P \in RP(r)}{\text{median}}(\boldsymbol{\mu}_G(P(R)))$$

Alternately, the maximum $\boldsymbol{\mu}_G$ value for the considered sample could be extracted as a way to estimate $\max_{P \in \mathfrak{S}_N} \boldsymbol{\mu}_G(P(R))$. Its accuracy depends strongly on sampling size and distribution shape, and knowledge of the worst-case permutation does not help understand the behaviour of individual permutations, therefore we do not try to estimate this value.

In many scenarios, applications running on real systems generate traffic which behaves like random permutations. This might be due to job placement strategies progressively degrading to fragmented placements with regards to topology locality, and/or due to non-regular communication patterns (which can be unavoidable when using custom communicators in MPI). Regardless of the cause or how avoidable this behaviour is, it is realistic and should be taken into consideration when evaluating a routing algorithm.

The number of permutations r is hard to choose in advance but can be estimated conservatively large (if available processing power and time allow) based on statistics on similar prior cases. Alternately, a dynamic stop criterion could be computed online based on statistics over the growing set of $\boldsymbol{\mu}^{RP}$ values, such as standard deviation of a statistically significant number of $\boldsymbol{\mu}^{RP}$ values.

Shift permutations

The second choice of sampling is the full set of shift permutations (SP), since some communication layers are expected to decompose collective communications in subsets of shift patterns [96]. This follows the property of rearrangeable optimality described in Subsection 2.2.1 with the convention of targeting the simplest class of equivalent permutations. Global collectives can be decomposed in sequences containing any shift permutation, so congestion risk under shift patterns is characterised using the following single static metric:

$$\boldsymbol{\mu}_G^{SP}(R) := \max_k(\boldsymbol{\mu}_G(SP_k(R)))$$

An interesting use of this metric is that it computationally allows determining the blocking factor of a network for a given routing function, or of the network itself if the considered routing function is optimal for shift permutations. For example, an RLFT has $\boldsymbol{\mu}^{SP}(Dmodk) = 1$, corresponding to its non-blocking nature.

The assumption that traffic actually traverses the network as shift patterns, realistically expands to the following assumptions:

- Communication layers decompose global collectives as shift permutations on the global order,
- Jobs are not fragmented on the global order,
- For programs using MPI, either they do not use custom communicators, or MPI preserves the global numbering for custom communicators.

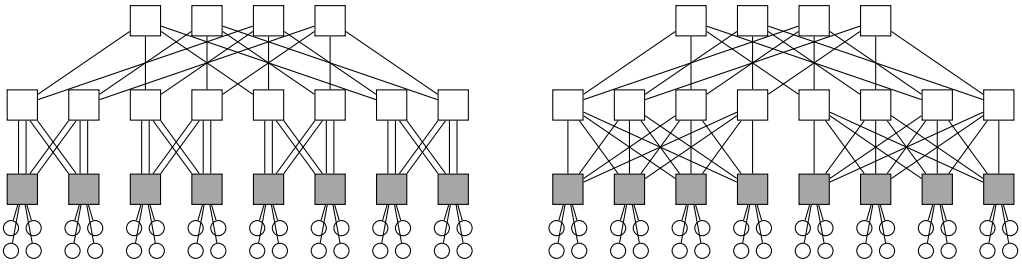
2.3.3 . Effective diameter

The studied metrics only model quality in terms of load or congestion. When possible and useful, this can be complemented by another static metric based on the effective distances between nodes under a given routing function, to model effective uncongested latency (also called *effective diameter*). The effective distance from nodes s to d under routing function R is noted $\boldsymbol{\nu}_{s,d}(R)$ and corresponds to the number of edges in the path from s to d. Much like a network's overall latency for global communications is modelled by its diameter, the effective overall latency for a specific job is modelled by the maximum effective distance between its nodes. However, job placement is not known beforehand in general, and average effective distance $\boldsymbol{\nu}_G$ for all nodes, reflecting effective latency of individual communications, can complement diameter; it is computed as such for a network $G(S,N,E)$:

$$\boldsymbol{\nu}_G(R) = \underset{s,d \in N}{\text{mean}}(\boldsymbol{\nu}_{s,d}(R))$$

This models average uncongested latency well when job placement is uniform and communications are independent from one another. Alternately, median distance could be used to model representative uncongested latency, but it provides little insight over diameter when the distance separating most pairs is the same as the diameter (such as in most fat trees). If job size is known, and if global communications are expected instead of independent ones, then average or median effective diameter for such jobs could be computed instead, in a much more difficult way. However, in the general case, studying diameter complemented with average distance is enough to give insight regarding effective network uncongested latency.

This metric doesn't bring any insight when comparing minimal routing functions in the same network, but it can allow obliviously comparing networks



(a) $G_P = PGFT(3;4,2,4;1,2,2;1,2,1)$ (b) $G_Q = QFT(3;4,2,4;1,2,2;1,2,1)$

Figure 2.2: Example of recabling to reduce effective diameter

or routing functions. As an example, $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$ can be used to show how a QFT allows recabling an existing PGFT with the same load properties but with lower effective diameter: Figure 2.2 shows a PGFT G_P and a QFT G_Q, both non-maximal topologies with 20 8-port switches, 32 nodes, and 80 links, and with half of the ports unused in the third level. This last fact corresponds in both cases to a blocking factor of 2:

$$\boldsymbol{\mu}^{SP}_{G_P}(Dmodk) = \boldsymbol{\mu}^{SP}_{G_Q}(PQFT) = 2$$

We can differentiate the networks using the newly defined average distance metric. In both cases, the routing function is minimal, and we can look at distance instead of effective distance. In G_P, each node is 2 hops from 3 others, 4 hops from 4 others, and 6 hops from 24 others. In G_Q, each node is 2 hops from 3 others, 4 hops from 12 others, and 6 hops from 16 others. As a result, we find the averages:

$$\boldsymbol{\nu}_{G_P}(Dmodk) = \frac{2\times 3 + 4\times 4 + 6\times 24}{3+4+24} \approx 5.35$$
$$\boldsymbol{\nu}_{G_Q}(PQFT) = \frac{2\times 3 + 4\times 12 + 6\times 16}{3+12+16} \approx 4.84$$

In this case, switching to the QFT brings the average distance down by nearly 10% (and doubles the size of diameter-4 groups) in exchange for a more complicated routing algorithm.

Effective diameter is a contribution that can be used to quantitatively compare routing functions and topologies in contexts where they affect path length. A similar concept with the same name already exists in the context of the Internet, but instead describing the minimum distance at which a given proportion of all nodes is attainable from a given node [65].

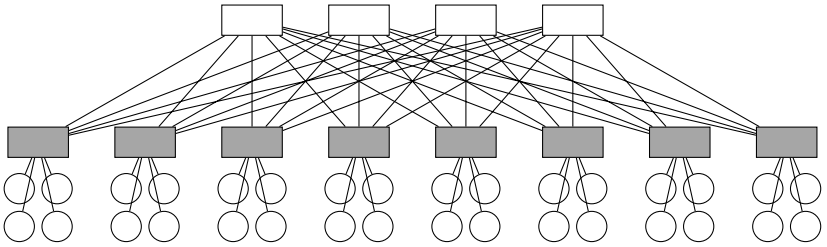


Figure 2.3: Example topology G_2 ($XGFT(2;4,8;1,4)$)

Algorithm	$\boldsymbol{\mu}_{G_2}^{SP}$	$\boldsymbol{\mu}_{G_2}^{RP(1000)}$
Dmodk	1,1,1,1,1,1	5,5,5,5,5,5
Ftree	1,1,1,1,1,1	5,5,5,5,5,5
RandSP	9,7,7,9,7,8	5,5,5,5,5,5
Minhop	1,1,1,1,1,1	5,5,5,5,5,5

Table 2.1: Static metrics applied to G_2

2.4 . Example application of static metrics

As a simple example to illustrate the kind of insight static metrics can provide, let's consider a 2-level 8-switch RLFT topology G_2 (corresponding to formula $XGFT(2;4,8;1,4)$ as shown in Figure 2.3), routed with Dmodk, Ftree, RandSP, or OpenSM's Minhop. As described in Section 2.3.2, $\boldsymbol{\mu}_{G_2}^{SP}(Dmodk)$ is equal to 1 since G_2 is an RLFT. For RP patterns, we arbitrarily sample 1000 random permutations. Since all considered algorithms are minimal, effective diameter metrics are not interesting. OpenSM's Minhop is computed as a black box, testing the hypothesis that it chooses randomly among equivalent paths. The other algorithms were presented in Section 1.4 and are known (Dmodk and Ftree are deterministic and RandSP is non-deterministic). Each algorithm is run and analysed 6 times to determine statistical behaviour for RandSP and Minhop (since they are at least potentially non-deterministic) and to estimate RP stability for the chosen sample size, with results displayed in Table 2.1.

Results show that:

- the chosen sample size appears sufficiently large for the considered topology (since $\boldsymbol{\mu}_{G_2}^{RP(1000)}$ doesn't vary for each run),
- Dmodk and Ftree behave as expected (being optimal and equivalent in PGFTs),
- RandSP behaves poorly for shift permutations,
- Minhop behaves better than RandSP (and is seemingly deterministic) and performs as well as Ftree/Dmodk in this small RLFT,

- all runs resulted in the same median congestion risk for random permutations.

In turn, this leads to the conclusion that any one of these algorithms is adapted to this topology for general use, though RandSP is inadapted for shift permutations.

2.5 . OMNeT++-based simulation

As described in Subsection 2.2.2, the behaviour of a network under communications with known temporal behaviour can be simulated, for example at flit level using OMNeT++ [83, 84]. There are many examples of simulations based on OMNeT++ [59, 54, 5]. Such a simulation is more precise than the previous ones which use static traffic patterns, since it doesn't rely on a sustained load assumption. Instead, it simulates congestion behaviour across all ports at each time step, with an arbitrary precision depending on how simulated decision making precisely reflects reality. The limitation of such a simulation is that it does not reveal global network behaviour, which is instead at best estimated by simile, by comparing simulations across a variety of realistic use cases.

We simulate a use case with OMNeT++ in Section 3.5, as a partial attempt at verifying legitimacy of the static metric used everywhere else. It was outside of the scope of this work to design simulation parameters that reflected a wide array of use cases, and instead an existing simulation tool for HPC systems was used: Sauron [92]. This tool, based on the OMNeT++ framework, has been used in several studies of HPC routing techniques [93, 94], including several combining queueing schemes and adaptive routing [55, 70, 71] and energy saving techniques [101]. Within this framework, simple parameters were chosen to model a network small but large enough to exhibit the behaviour studied in that case.

3 — Routing for heterogeneous fat-trees

Section 1.2.8 presents node-type heterogeneity. For applications which are aware of unusual node types, the implementation of their communications often results in *node-type-specific* communication phases: at any given time, source nodes of a single type are sending messages to destination nodes of a single type. The aim of this chapter is first to show in what ways existing load-balancing routing algorithms (Dmodk/Ftree and Smodk) can result in avoidable congestion during node-type-specific communication phases. Secondly, this chapter proposes a new routing technique extending many existing algorithms, providing the same load-balance for node-type-specific patterns as the base algorithms do for type-unspecific ones.

Section 3.1 describes the existing context in more detail to present common placement strategies. A sample heterogeneous topology is presented in Section 3.2, alongside one of the previously presented common placement strategies. A corresponding adversarial communication pattern is chosen in Section 3.3, and three routing algorithms are then analysed in detail to show how they under-utilise available network resources. Section 3.4 then presents a new technique to use these resources more efficiently without losing properties of the existing algorithms, and provides a static study of this new technique for the considered case study. The study is reproduced with a dynamic simulation in Section 3.5. Finally, Section 3.6 concludes on this chapter.

3.1 . Heterogeneous clusters

Supercomputers are often clusters made of several types of nodes, rather than the common description of a single type of computing nodes. Secondary nodes can include storage nodes for short and long-term data storage; service or management nodes for login, node reservation, deployment, monitoring, fault-tolerance; GPGPU and FPGA nodes for optimised computations.

There are various strategies to place secondary nodes in existing clusters, which are usually not described in research material. In the case of fat-trees, strategies can include placing a constant number of secondary nodes at every leaf, adding an irregular subgroup with secondary nodes connected to the top switches like the other regular subgroups (this generally breaks fat-tree properties), or connecting the cluster to an external topology via routers. This last possibility is common with storage systems such as the Lustre parallel filesystem, where routers can be nodes of the cluster connecting it with an array of storage servers of which the fabric management and routing algorithm are not aware.

As a practical example affecting secondary node placement, first generation

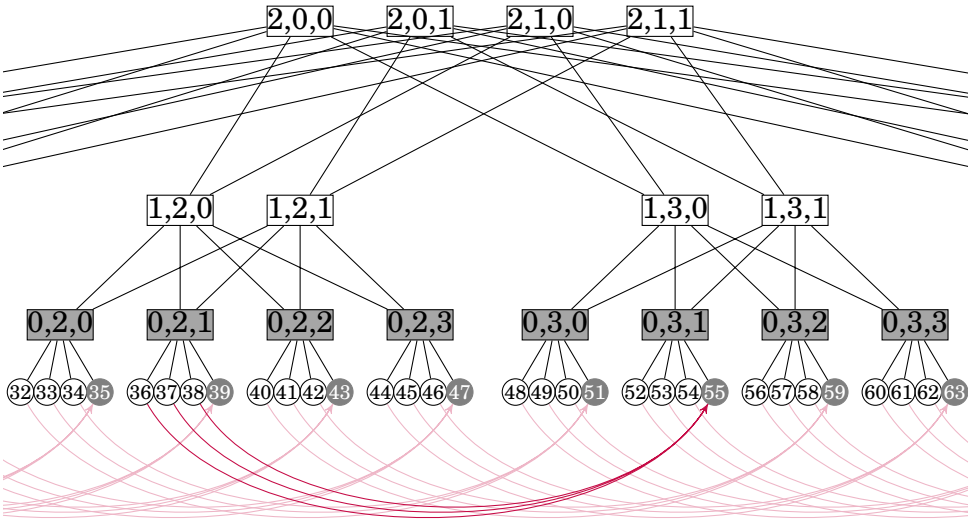


Figure 3.1: Example topology G_3 with storage nodes in grey, and only two out of six groups shown. The pathological case study pattern is shown in red.

48-port BXI leaf switches have either zero or three optical ports. The optical ports are numbered identically on all switches of the same model and are dedicated to nodes physically far within the network; in practice, secondary nodes. As we will see, this impacts both placement and routing, potentially exacerbating performance issues.

3.2 . Case study topology

The canonical situation we are studying here which exacerbates avoidable congestion, as a result of node-type heterogeneity, arises when the blocking factor is greater than 1:1, when all nodes of a given type are connected to the same port numbers on leaf switches, and when applications communicate by node-type-specific phases. Accordingly, we have chosen as a case study to consider a 6-port, 3-level fat-tree topology G_3 with blocking factor 4:1, defined as $XGFT(3;4,4,6;1,2,2)$, shown in Figure 3.1. Nodes are indexed by port rank on their leaf and by leaf address comparison between leaves. The last node connected to each leaf (or more precisely, every node with an NID congruent to 3 modulo 4) is a storage node. Every other node is a computing node.

3.3 . Analysis of a node-type-specific communication pattern

This case study is based on a communication pattern one might find in some distributed memory applications: data collection from computing nodes to storage nodes. In this case, a pathological case with all routes crossing top switches: computing nodes with NID n communicate to the storage node of the fourth next leaf, with NID d:

$$d = \left(\left(\left\lfloor \frac{n}{4} \right\rfloor + 4\right) \bmod 32\right) \times 4 + 3$$

For example, nodes with NIDs 36, 37 and 38 send to the node with NID d':

$$d' = ((9+4) \bmod 32) \times 4 + 3 = 55$$

For a given routing function R, we call $C2IO(R)$ the subset of routes used by this pattern.

3.3.1 . Dmodk/Ftree performance

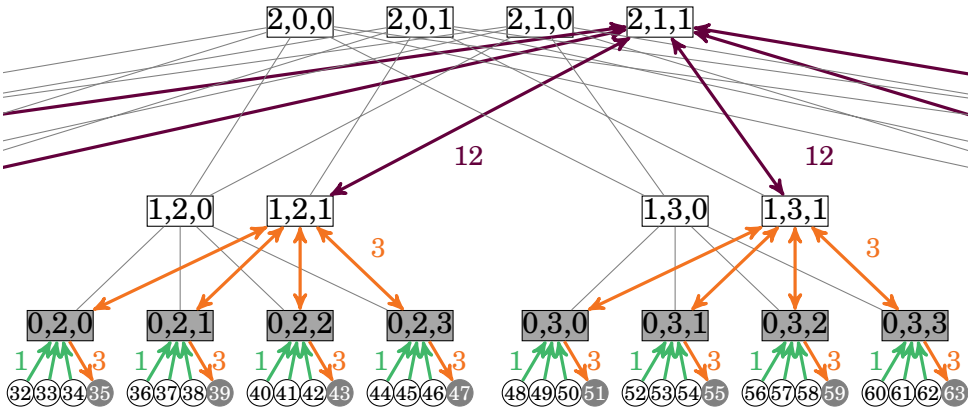
G_3 is an XGFT, therefore Dmodk routing and Ftree routing provide the same result. With both routings, routes going upwards from leaves are computed using the formula $p = d \bmod 2$ equal to 1 for all considered destinations, therefore always using the 2nd up port. Ports are ordered by other-end switch address, and all routes go through switches $(1,*,1)$. Next up, routes going upwards from these switches are computed using the formula $p = \left\lfloor \frac{d}{2} \right\rfloor \bmod (2)$, equal to 1 for all considered destinations, therefore always using the 2nd up port. All routes go through switch (2,1,1). Finally, routes go down to the correct destination through the only shortest path. These routes are shown in Figure 3.2, with corresponding number of sources and destinations at each edge.

The highest congestion risk arises in each edge of 6 top-level links, wherein all 12 computing nodes of a group are sources to all 4 storage nodes of another group. As a result, we have:

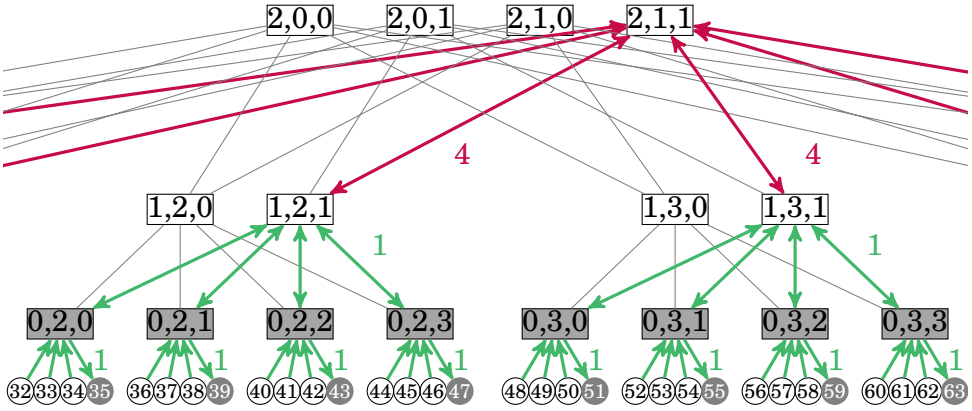
$$\boldsymbol{\mu}_{G_3}(C2IO(Dmodk)) = \boldsymbol{\mu}_{G_3}(C2IO(Ftree)) = 4$$

To reformulate this result: for the given communication pattern, 3 top switches are unused while each port of the remaining top switch is at risk of congestion under the load of 4 distinct flows.

Even though 4 sustained flows on an edge might not induce congestion (depending on characteristics of the considered network), large realistic examples can result in much larger congestion risks. Regardless, this distribution of routes is intuitively sub-optimal.

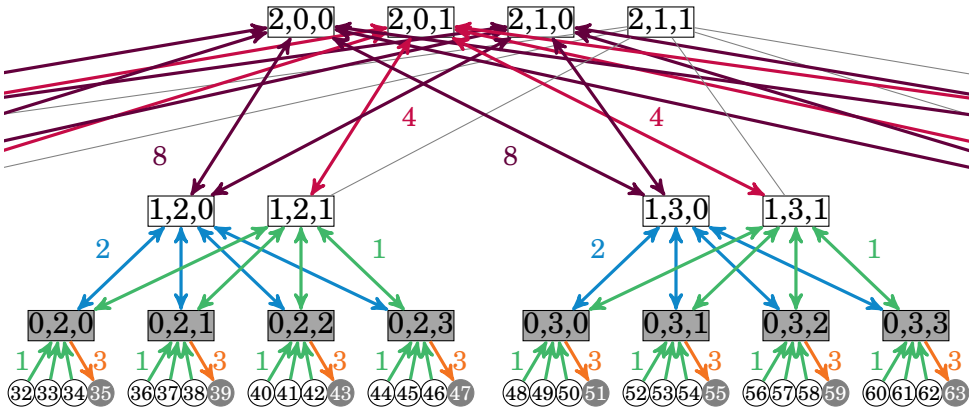


(a) number of sources

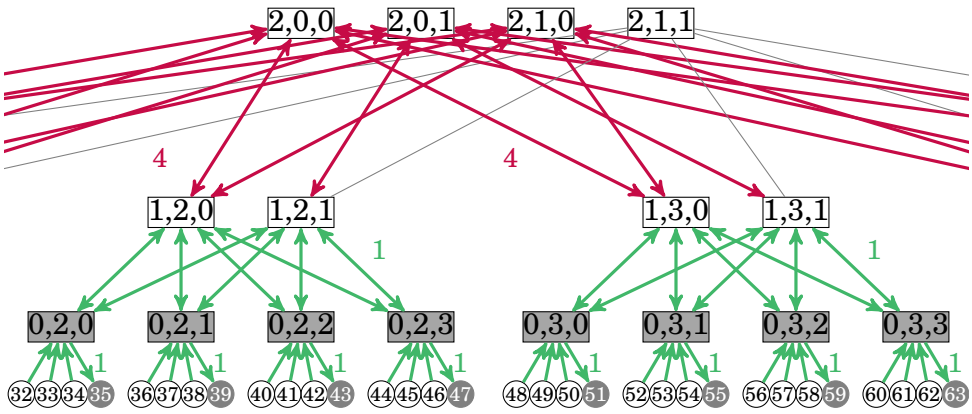


(b) number of destinations

Figure 3.2: C2IO routes in G_3 under Dmodk routing.



(a) number of sources



(b) number of destinations

Figure 3.3: C2IO routes in G_3 under Smodk routing.

3.3.2 . Smodk performance

With Smodk, routes from computing to storage nodes are spread per source. With the same process as Dmodk for computing nodes as destinations, we determine which ports are used with Smodk for computing nodes as sources. These routes are shown in Figure 3.3.

Each leaf switch spreads its 3 routes going upwards among its 2 up ports using the $p = s \bmod 2$ formula. This places 2 routes on the 1st up port and 1 route on the 2nd up port. At the next level, the 3 routes are spread using the $p = \left\lfloor \frac{s}{2} \right\rfloor \bmod 2$ formula over the 3 first up ports of the 2 switches in the group. Downroutes take the only resulting minimal path. With this process duplicated to all leaves, 4 sources and 4 destinations are routed through each of these 3 top-level links. This leaves these links as the highest congestion risk, with 4 distinct flows:

$$\boldsymbol{\mu}_{G_3}(C2IO(Smodk)) = 4$$

3.3.3 . RandSP performance

The previous algorithms make an unnecessarily large number of unrelated routes overlap, because the modulo operation depending on NIDs has no information about the communication pattern. RandSP does not depend on NID; it spreads all routes according to a uniformly random distribution over the available ports and, as a result, every subset of routes is on average spread uniformly. Therefore, $C2IO(RandSP)$ does not have particularly coalesced routes.

However, distributing each set of 12 routes into its corresponding 4 top-ports practically always causes collisions between distinct flows. To estimate the probability, we first define a collision as a port whose $\boldsymbol{\mu}$ value is greater than 1. Then, we notice that the communication pattern is made up of groups of 3 sources sending to 1 destination, from which we deduce that the number of sources is never smaller than the number of destinations at any port. As a result, we can replace the $\boldsymbol{\mu}$ metric with the number of destinations. To keep this metric at 1, each distribution of 12 routes (with 4 destinations) over 4 ports must assign each destination to a separate port. Given that each route is independently chosen at random, the probability of each set of 3 coalesced sources to be assigned the same port is given by $\left(\frac{1}{4}\right)^2$. In turn, the probability of all 4 sets to each be on a single port is given by $\left(\frac{1}{4}\right)^{2\times 4}$. Furthermore, the probability that these ports are distinct from one another is given by:

$$\frac{3}{4} \times \frac{2}{4} \times \frac{1}{4} = \frac{6}{4^3}$$

Altogether, the probability for the metric to be equal to 1 in each subgroup is:

$$\frac{1}{4^8} \times \frac{6}{4^3} = \frac{6}{4^{11}} \approx 1.4 \times 10^{-6}$$

Considering all 6 groups, this comes down to:

$$\left(\frac{6}{4^{11}}\right)^6 \approx 8.6 \times 10^{-36}$$

Therefore, we can safely state that $\boldsymbol{\mu}_{G_3}(C2IO(RandSP))$ is practically strictly larger than 1. Repeated computation of RandSP for the given topology and communication pattern resulted in $\boldsymbol{\mu}_{G_3}(C2IO(RandSP))$ values of either 3 or 4: i.e. rarely better than Xmodk.

RandSP will often give better results than Dmodk or Smodk when the communication patterns have a given bias, but it will always leave some ports with avoidable congestion.

3.4 . Grouped Xmodk

In the previous section we show that the existing routing algorithms do not balance the load correctly when the topology has mixed node types.

Just as Xmodk algorithms aim to compute perfect routes for the general worst-case scenario, we want to compute perfect routes for the node-type-specific worst-case scenario. To improve routing for node-type-specific communication patterns, we can use knowledge of node types and modify Xmodk algorithms accordingly. The aim is to optimise resource usage depending on node type. For example the optimisation should achieve the best throughput for communications towards storage proxies or computing nodes. We suggest balancing each group of nodes separately to improve load-balancing under worst-case node-type-specific patterns.

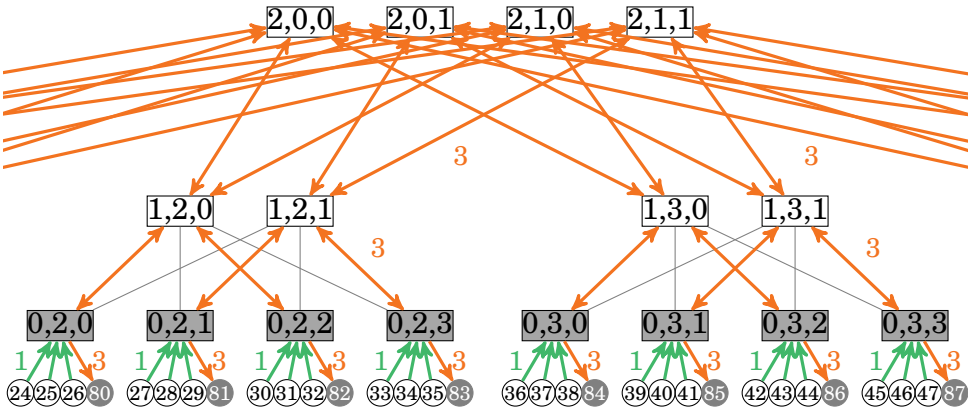
3.4.1 . Reindexing NIDs

Grouped Xmodk algorithms, or *Gxmodk*, consist of preprocessing NIDs. For this we use the type of each node n, noted type(n), and we choose to call gNID(n) its reindexed NID. Knowing the type of all nodes, the algorithms begin by updating the NIDs accordingly, as shown in Algorithm 4.

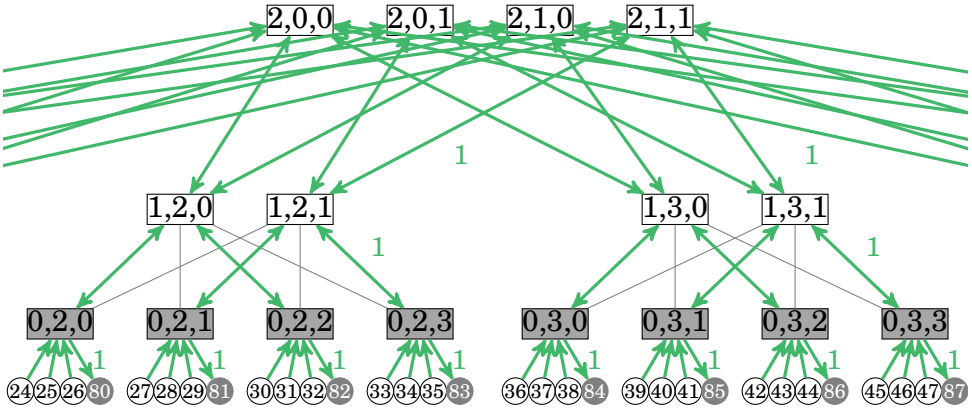
Algorithm 4: Reindex NIDs by type

```
1 x ← 0
2 foreach t ∈ {type(n) | ∀n ∈ N}, in a deterministic order do
3     foreach n ∈ N | type(n) = t, in order of NIDs do
4         gNID(n) ← x
5         x ← x + 1
```

Note that original order is preserved within groups, as defined in line 3. Xmodk is then applied as usual but with gNIDs instead of NIDs.

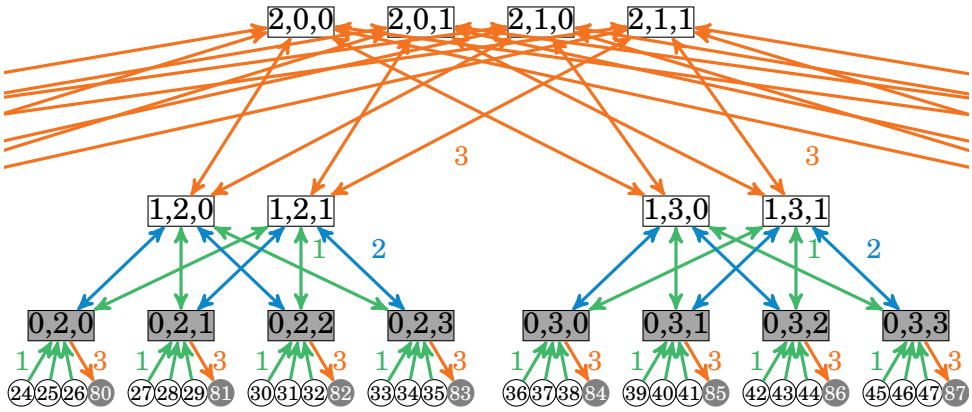


(a) number of sources

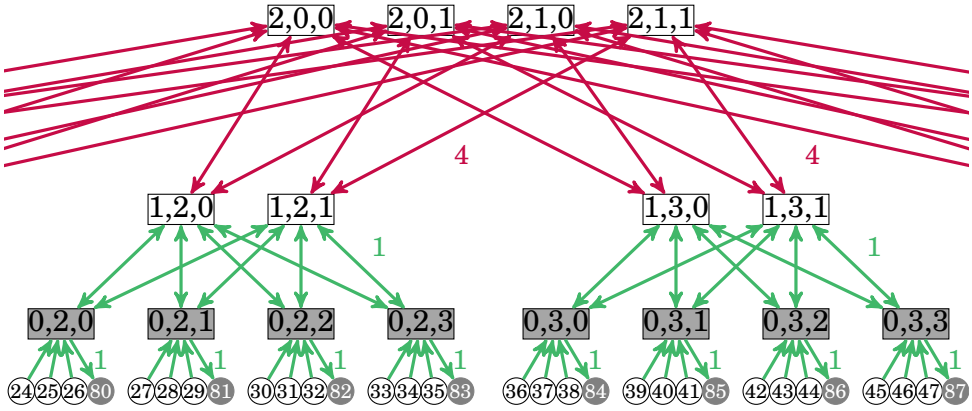


(b) number of destinations

Figure 3.4: C2IO routes in G_3 under Gdmodk routing, with gNIDs.



(a) number of sources



(b) number of destinations

Figure 3.5: C2IO routes in G_3 under Gsmodk routing.

For the given case study, let's suppose that computing nodes are reindexed first: there are 72 so they are assigned gNIDs 0 to 71; storage nodes are assigned gNIDs 72 to 95.

3.4.2 . Gxmodk case study

Gdmodk performance

Using Gdmodk, each subgroup's storage destinations are spread out evenly on the 4 top switches, with 1 destination per top switch. For example, the node with NID 59 and gNID 86 is assigned $(2,1,0)$. Every pair of routes from distinct sources to distinct destinations (within C2IO) therefore uses disjoint paths. Figure 3.4 shows how Gdmok distributes routes optimally when considering this node-type-specific communication pattern.

$$\mu_{G_3}(C2IO(Gdmodk)) = 1$$

Gsmodk performance

Using Gsmodk, the 16 computing sources of each group are spread out evenly on the 4 top switches, with 3 sources per top switch. Figure 3.5 shows how Gsmodk spreads routes efficiently for node-type-specific patterns, though not as efficiently for this pattern as Gdmodk.

$$\mu_{G_3}(C2IO(Gsmodk)) = 3$$

On the symmetrical communication pattern, we would see the same improvement as we do between Dmodk and Gdmodk for the considered communication pattern. In general, if pattern P is symmetrical to Q, we should always find:

$$\begin{aligned}
\mu_G(P(Dmodk)) &= \mu_G(Q(Smodk)) \\
\mu_G(Q(Dmodk)) &= \mu_G(P(Smodk)) \\
\mu_G(P(Gdmodk)) &= \mu_G(Q(Gsmodk)) \\
\mu_G(Q(Gdmodk)) &= \mu_G(P(Gsmodk))
\end{aligned}$$

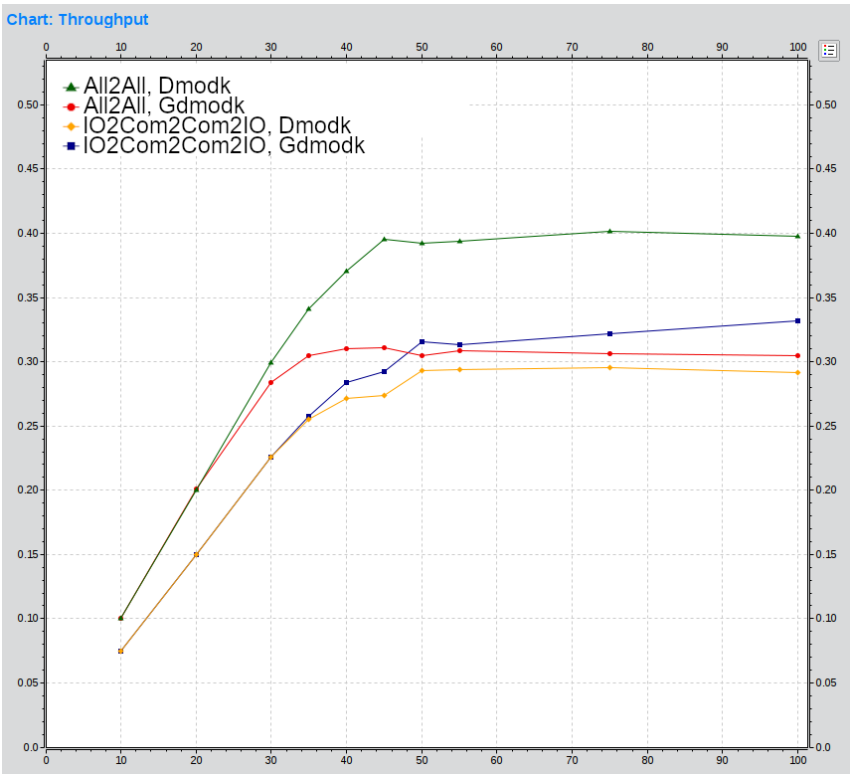


Figure 3.6: Effective throughput in sample pruned fat-tree using Sauron simulation

3.5 . OMNeT++-based simulation of Gdmodk

A similar study was conducted using dynamic simulation with Sauron, described in Section 2.5. The study was conducted on a sample fat-tree $XGFT(3;4,3,2;1,2,2)$ with a blocking factor of 2:1, on which static analysis predicts lower congestion risk for Gdmodk routing on type-based communication patterns. The simulation framework was extended to allow simulation of any PGFT. Both node-type-specific and type-unspecific synthetic random communication patterns were simulated. The node-type-specific communication pattern is made up of a phase from storage to computing nodes, followed by a phase between computing nodes, and finally followed by a phase from computing to storage nodes again; each switch's node with the largest port rank was selected as storage node. On the other hand, the type-unspecific pattern has all nodes communicating randomly regardless of type.

Results are presented in Figure 3.6. Expected results are verified in this simulation: in the sample pruned fat-tree, Gdmodk improves maximal throughput for node-type-specific patterns but strongly reduces maximal throughput for type-unspecific patterns.

3.6 . Conclusions and future works

This chapter provides a realistic node-type-specific communication pattern which is present on production clusters shipped by Atos. Existing solutions are shown to result in unnecessary congestion faced with this real-life scenario. A new method is outlined, providing minimal congestion risk in the example. This method has been published [103], patented [105], and is used in BXI clusters.

The congestion issue of Xmodk stems from nodes of a same type having the same NID, modulo arities. This also affects communications unrelated to node-type, but optimising for these means knowing about application usage. Gxmodk aims only to improve the situation when node-type is known; having early knowledge of applications' communication matrices would warrant writing specific deterministic algorithms.

This preprocessing method is applicable to other arithmetic algorithms targeting IFTs, such as those shown in Sections 4.2, 5.1.2, 9, and 11.

4 — Fault-resilient routing in fat-trees

4.1 . Reconfiguration mechanisms

As presented in Section 1.2.9, current and future large-scale systems are expected to continue working even in the case of unpredictable network changes (failure and restore of any equipment among nodes, switches and links) with potentially low median time between failures (MTBF); though there isn't any known algorithm targeting PGFTs which provides high-quality routing tables with fast enough computation run time. An example degraded topology is shown in Figure 4.1. Proposals addressing this issue must be correct, should be fast, and might result in routing functions with varying levels of quality (as can be studied using the static metrics defined in Chapter 2). A common approach is to use an algorithm which accepts degraded fat-trees (namely OpenSM's Ftree, described in Section 1.4.1) and apply it on the new state of the topology for any network change; this is not only somewhat slow (as will be shown at the end of this Chapter), but also degrades ungracefully with even a few faults [7]. Another existing technique is based on precomputation of all potential tables to avoid spending time computing the correct one upon failure [86]; this is unfeasible when considering large networks (with many potential points of failure) and short MTBF. Alternately, Dmodk (fast but specific to perfect fat-trees, as presented in Section 1.4.2) can be used for the offline phase, with an algorithm moving only invalidated routes for the online phase [98, 85], though this tends towards badly load-balanced routing for realistic network changes over time. Another potential approach is to first compute valid but unbalanced tables fast and then slowly compute high-quality tables; however this also does not scale to networks with short MTBF. Most existing proposals rely on a centralised manager to compute and upload routing tables, while one distributed technique has switches reacting to failures and updating new deterministic paths locally [58], providing fast reaction. Instead, this Section will present and study a fast and high-quality centralised proposal which was developed, implemented, and published during this thesis with the help of both my academic advisors and my colleagues at Atos [104, 106].

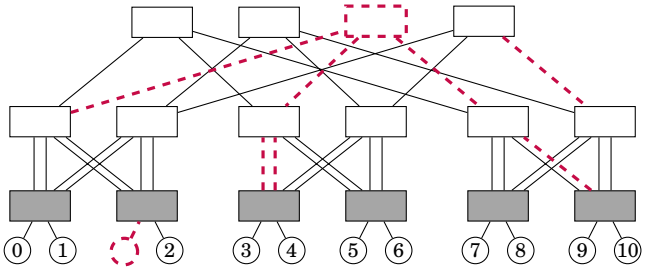


Figure 4.1: Example degraded $PGFT(3;2,2,3;1,2,2;1,2,1)$ with leaf switches shown in grey, and failed/removed equipment in dashed red lines.

4.2 . Dmodc

This novel approach aims to tackle the challenge of routing for PGFTs with fast computation time and low congestion risk even under large-scale equipment failure. It does so by applying the closed-form arithmetic formula of Dmodk while relaxing its topological constraint. For that purpose, it computes shortest paths explicitly rather than relying on an addressing scheme, and it balances load based on locally propagated information rather than relying on level-wide constants. These two goals are addressed together during a partially sequential preprocessing phase.

We call this algorithm Dmodc. The *c* in Dmodc refers to the neighbouring switches explicitly determined to be *closer* to the destination among which paths are chosen. The aim is fast centralised computation of routing tables for degraded PGFTs, providing optimal or well-balanced deterministic routes even under heavy fabric degradation. The algorithm begins with a preprocessing phase (with steps that can be multi-threaded) followed by a strongly parallel computation phase.

4.2.1 . Preprocessing

The aim of the preprocessing phase is to compute equivalents to the values used in the Dmodk formula, with methods that will be detailed in this Subsection. Firstly, the modulo operation (and the selection process in general) requires a set of target ports: a *cost* metric is computed throughout the network to later speed up determination of ports leading to neighbours closer to the destination. Secondly, the division by the number of uplinks at each lower level is replaced by a propagation of *dividers* upwards through the network, attempting to replicate the values which would be found in a corresponding PGFT, even under degradation. Finally, the destination NID is replaced with another number (the *topological NID*), to improve route quality of target communication patterns.

These computations rely on knowledge of levels ranks (or ranking, as described in Section 1.4.5), which can be done with any of the processes defined in Section 1.4.5 and Annex 2. Groups of ports linked to the same switch

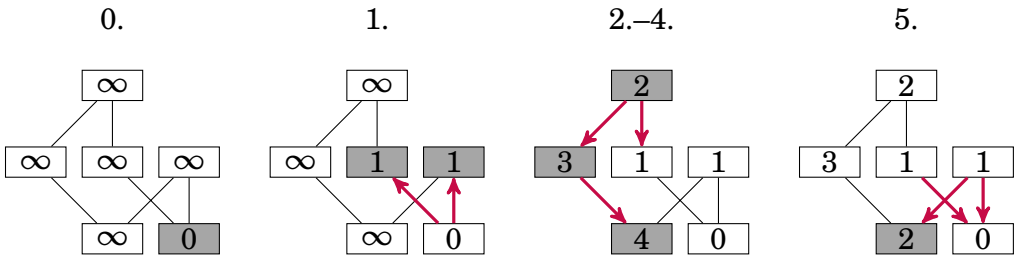


Figure 4.2: Example sequence of cost propagation steps in a degraded part of a network. Costs to the bottom-right switch are shown in switches. At each propagation step, the updated costs are in grey. Note that in steps 3–5, some propagations are interrupted due to the $c_{s,l} + 1 < c_{s',l}$ condition in the procedure. They could have been achieved with a simple $c_{s',l} = \infty$ condition instead; however this would have also interrupted the propagation of 2 in step 5. As a result, the long path on the left would not have been avoided. For PGFTs (degraded or not), such cases are actually impossible and the simple condition would suffice; but it would not guarantee shortest up–down paths in fat-tree-like topologies.

are prepared and sorted by globally unique identifier (GUID) to help with same-destination route coalescing.

Cost and divider (and the complexities of their computation) are described separately, but the divider computation can be integrated in the upward phase of the cost computation. As a result, their combined computation complexity in big O notation is equivalent to that of cost alone.

Cost

We define the cost $c_{s,l}$ of a switch s to a leaf switch l to be the minimum number of hops between one another under up–down restrictions according to rank, as defined in Algorithm 5 and illustrated in Figure 4.2. This later allows us to determine valid paths by exploring neighbouring switches and comparing costs. That exploration could be done at this point to prepare sets of output ports, but it's better to leave it for later since each set is only used once (see Subsection 4.2.2). Other all-pairs shortest paths methods could be substituted here.

Thanks to the up–down restriction, the complexity of this procedure is in $\mathcal{O}(\#E\#L)$. This restriction is only for efficiency, it does not enforce up–down paths (and therefore deadlock-freedom). Some fat-tree-like topologies would result in up–down–up–down paths (if such shortcuts appear in neighbouring switches), since path selection does not distinguish up and down neighbours. Avoiding this requires a slightly different method: an extra integer (downcost) must be stored, similar to cost but only for downpaths (propagated only upwards). A version of this method using only link direction instead of rank is provided in Algorithm 6, and more detail is given in Section 4.2.2.

In our partially parallel implementation, each worker thread obtains a block of switches to propagate with one barrier per-level upwards, then downwards.

Algorithm 5: Compute costs and dividers

foreach $s \in S$ **do**
  foreach $l \in L$ **do**
    $c_{s,l} \leftarrow \infty$
  $\Pi_s \leftarrow 1$
foreach $l \in L$ **do**
  $c_{l,l} \leftarrow 0$
foreach $s \in S$ *sorted in ascending rank order* **do**
  $\pi \leftarrow \Pi_s \times \#\{s' \mathrel{\neg\!\!\downarrow} s\}$
  foreach $s' \mathrel{\neg\!\!\downarrow} s$ **do**
    foreach $l \in L \mid c_{s,l} + 1 < c_{s',l}$ **do**
      $c_{s',l} \leftarrow c_{s,l} + 1$
  foreach $s' \mathrel{\neg\!\!\downarrow} s \mid \Pi_{s'} < \pi$ **do**
    $\Pi_{s'} \leftarrow \pi$
foreach $s \notin L$ *sorted in descending rank order* **do**
  foreach $s' \mathrel{\neg\!\!\uparrow} s$ **do**
    foreach $l \in L \mid c_{s,l} + 1 < c_{s',l}$ **do**
      $c_{s',l} \leftarrow c_{s,l} + 1$

Divider

Dmodc is based on the same arithmetic formula as Dmodk: prior to the modulo operation, it begins with an integer division by the product of $\#\{s' \in S \mid s' \mathrel{\neg\!\!\downarrow} s\}$ (the upward arity of s) of switches at each lower level. This value represents the number of consecutive destinations to route through the same port. It is multiplied when going up levels to mirror the number of consecutive choices by switches below before each switch is chosen again.

To reflect the actual state of the network (in which switches of the same level may have different arities), only local information must be considered; in turn, this operation is based on a *divider* value (noted Π_s), computed using the products of up-to-date counts of upswitches (switches connected above), as defined in Algorithm 5. Each downpath corresponds to a potential divider value, and we choose to keep only the maximum (as illustrated in Figure 4.3). The underlying motivation is to generate the same values as in the non-degraded PGFT, as long as the topological subgroup is not systematically degraded. In general, this results in a little over-aggregation in case of failures, whereas any other choice of downpath would instead result in larger amounts of under-

Algorithm 6: Compute costs c, downcosts c' and dividers π

Data: link directions

foreach $s \in S$ **do**
  foreach $l \in L$ **do**
    $c_{s,l} \leftarrow \infty$
    $c'_{s,l} \leftarrow \infty$
  $\Pi_s \leftarrow 1$
foreach $l \in L$ **do**
  $c_{l,l} \leftarrow 0$
  $c'_{s,l} \leftarrow 0$
repeat
  $\Delta \leftarrow \bot$ /* Was there any propagation? */
  $\pi \leftarrow \Pi_s \times \#\{s' \supset s\}$
  foreach $s \in S$ **do**
    foreach $s' \supset s$ **do**
      foreach $l \in L \mid c_{s,l} + 1 < c_{s',l}$ **do**
        $c_{s',l} \leftarrow c_{s,l} + 1$
        $c'_{s',l} \leftarrow c_{s,l} + 1$ /* Propagated upwards only */
        $\Delta \leftarrow \top$
    foreach $s' \supset s \mid \Pi_{s'} < \pi$ **do**
      $\Pi_{s'} \leftarrow \pi$
until $\Delta = \bot$
repeat
  $\Delta \leftarrow \bot$
  foreach $s \notin L$ **do**
    foreach $s' \subset s$ **do**
      foreach $l \in L \mid c_{s,l} + 1 < c_{s',l}$ **do**
        $c_{s',l} \leftarrow c_{s,l} + 1$
        $\Delta \leftarrow \top$
until $\Delta = \bot$

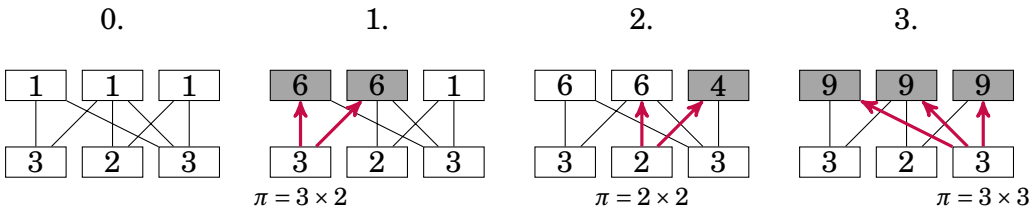


Figure 4.3: Example sequence of divider propagation steps in a degraded part of a network. Dividers are shown in switches. At each propagation step, the updated dividers are in grey. Note that in step 2, the first upswitch is not updated because $\pi = 2 \times 2 \leq 6$. Even though there are multiple degradations in the considered case, all top switches end up with the divider that they would have had in the complete network.

aggregation. Experiments did not provide any strong argument for or against this choice instead of other arbitrary choices of downpath. The complexity of this procedure is in $\mathcal{O}(\#E)$.

Topological NID

The arithmetic nature of Dmodc guarantees load-balancing only if NIDs (on which the modulo operation is applied) are topologically contiguous. We explicitly determine each node's topological NID using previously computed costs in Algorithm 7. In our implementation, finding the l' leaves is accomplished through a qsort[1] call, with the comparison function first comparing cost, then UUID. This choice can be justified by the fact that for every iteration, X_0 is different (and therefore each sort is independent), and because in our case there is no guarantee on the starting order of leaves. Each qsort call covers $\#X$ values, which is decreasing and smaller than $\#L$ after the first iteration, and with the number of iterations being equal to the number of level-1 groups. In the case of a k-ary n-tree, this gives a complexity in $\mathcal{O}\left(\frac{\#N}{k}\#L \log \#L\right)$. This algorithm is multithreaded in our implementation, by splitting X recursively into blocks with progressively decreasing interleaf cost.

Alternately, an implementation using a single qsort call could be efficiently designed by preprocessing group-wise UUIDs and using the comparison rule defined in Equations (4.1) through (4.4):

[1]Qsort is the default array sorting function provided in standard C libraries [87], which is generally single-threaded, in-place but non-stable, and in $\mathcal{O}(n \log n)$ except for worst-cases.

Algorithm 7: Compute topological NIDs

```
t ← 0
X ← L sorted by UUIDs
while X ≠ ∅ do
    l ← X_0
    μ ← min_{l'∈X\{l}}(c_{l,l'})
    foreach l' = l, then l' ∈ X | c_{l,l'} = μ (in order of UUIDs) do
        foreach n ⇀ l' in port rank order do
            t_n ← t
            t ← t + 1
            X ← X \ {l'}
```

$$\forall\ l,l' \in L, \quad \Gamma_{l,l'} = \{l'' \in L \mid c_{l,l''} < c_{l,l'}\} \tag{4.1}$$

$$\forall\ l,l' \in L, \quad \gamma_{l,l'} = \min_{l'' \in \Gamma_{l,l'}} (U(l'')) \tag{4.2}$$

$$\forall\ n \in N, \quad t_n \in [0, \#N[\tag{4.3}$$

$$\forall\ n,n' \in N, \quad \gamma_{\lambda_n,\lambda_{n'}} < \gamma_{\lambda_{n'},\lambda_n} \Rightarrow t_n < t_{n'} \tag{4.4}$$

Note that (4.1) provides a cost-based definition of relative groups, from which (4.2) defines a group-wise UUID. Using this object, (4.3) and (4.4) define a total ordering of topological NIDs between nodes of different leaves. Nodes linked to the same leaf switch are ordered according to their port rank.

4.2.2 . Routes computation

The deterministic output port $p_{s,d}$ and the alternative output ports $P_{s,d}$ of every switch s for every destination $d \in N$ (not directly linked to s) are selected with a closed-form formula based on the results previously determined. First, port groups leading *closer* to λ_d are selected in (4.5) (without taking ranking into account), setting corresponding alternative output ports in (4.6):

$$C_{s,\lambda_d} \leftarrow \left\{g \in G_s \mid c_{\Omega_g,\lambda_d} < c_{s,\lambda_d}\right\} \tag{4.5}$$

$$P_{s,d} \leftarrow \{p \in g \mid g \in C_{s,\lambda_d}\} \tag{4.6}$$

C is an array ordered by the GUID of port groups' remote switches: individual groups are accessed with indices $i \in [0, \#C_{s,\lambda_d}[$ using the $C_{s,\lambda_d}[i]$ notation. Note that it is possible for a switch to have no valid up–down path towards a destination, which corresponds to $\#C_{s,\lambda_d} = 0$, in which case that destination is skipped. From this, the output port group is chosen in (4.7) and the port within that group in (4.8):

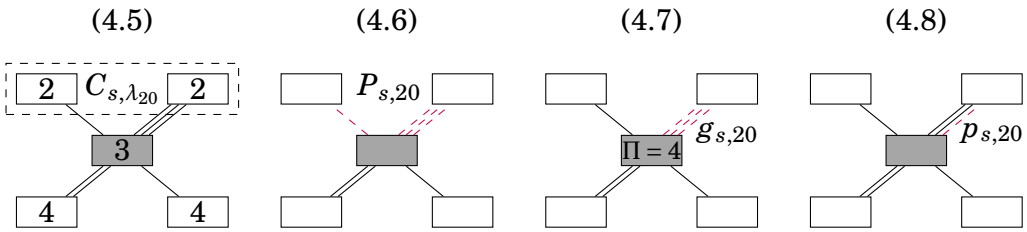


Figure 4.4: Example route computation with s in grey, $\Pi_s = 4$, and $d = 20$. Costs to λ_{20} are shown in switches. Indices are ordered from left to right. The top-right group is chosen as $g_{s,20}$ because $\lfloor 20/4 \rfloor \bmod 2 = 1$, and the right port in $g_{s,20}$ is chosen as $p_{s,20}$ because $\lfloor 20/(4 \times 2) \rfloor \bmod 3 = 2$.

$$g_{s,d} \leftarrow C_{s,\lambda_d}[\left\lfloor \frac{d}{\Pi_s} \right\rfloor \bmod \#C_{s,\lambda_d}] \tag{4.7}$$

$$p_{s,d} \leftarrow g_{s,d}[\left\lfloor \frac{d}{\Pi_s \times \#C_{s,\lambda_d}} \right\rfloor \bmod \#g_{s,d}] \tag{4.8}$$

Routes are computed in a loop over leaves so that $C_{s,\lambda}$ is determined only once for all nodes connected to λ (with $P_{s,d}$ also unchanging $\forall\ d \mid \exists\ \lambda_d$). Figure 4.4 illustrates assignments (4.5), (4.6), (4.7), and (4.8).

The cost variant for up–down restriction described in 4.2.1 requires (4.5) to compare c values for upswitches and the downpath cost value for downswitches. Alternately, the downpath cost value can be omitted by taking ranking into account (and disregarding downgroups if any upgroup is available).

4.2.3 . Primary results

The algorithm was implemented in BXI FM; the same code has been used for validation, simulation and in production.

Validity

Routing is valid for degraded PGFTs if and only if the cost of every leaf switch to every other leaf switch is finite: this reflects every node pair having an up–down path. Our implementation includes a pass through all leaf switch pairs to verify this condition. The up–down path restriction is sufficient to guarantee deadlock-freedom within degraded PGFTs [68].

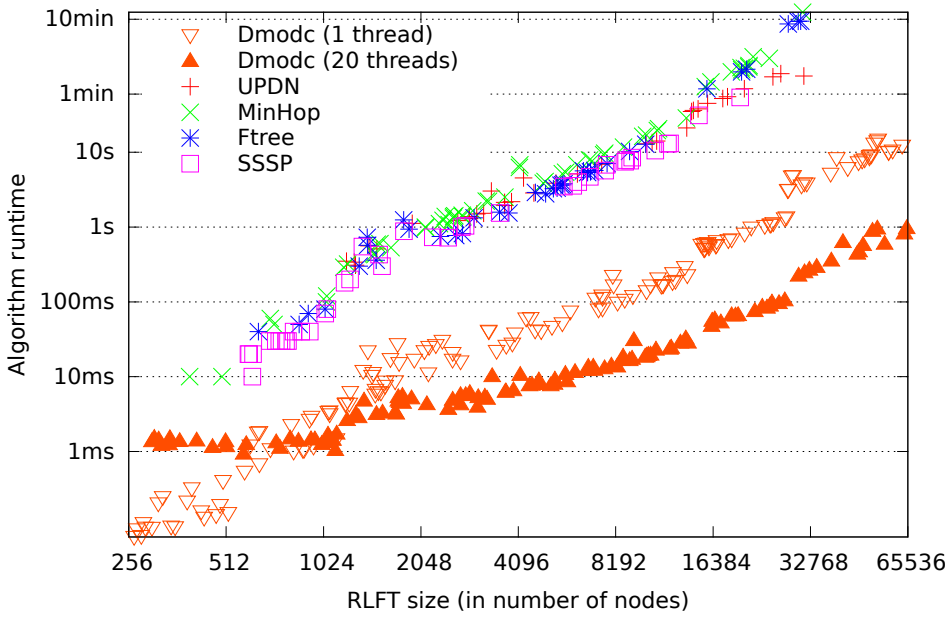


Figure 4.5: Algorithm run time on a 2.50GHz Intel Xeon E5-2680 v3 for Real-Life Fat-Trees of varying sizes (in log–log scale; lower is better).

Run time

Our C99 [2] implementation has computation of cost, divider, and routes spread over POSIX threads [1] fetching work with a switch-level granularity. Figure 4.5 reports complete algorithm execution time. Additionally, we included the routing run times of four applicable routing engines from OpenSM 3.3.21, measured by adding timers in the source code, all running on the same machine. For each number of nodes, a corresponding realistic RLFT is generated, with largely proportional numbers of switches and links. The representation of run time of every algorithm on this log–log scale plot is close to a line (when considering networks with more than a thousand nodes), their complexity is therefore largely polynomial to the amount of equipment (whether it be nodes, switches or links, since they are nearly proportional). Overall, it seems the main difference between these lines is the intercept (hinting at significantly different multiplicative factors) with the slope maybe increasing the distance between these lines as well (hinting at slightly different exponents). Finally, the run time of Dmodc is at least fifteen times faster than all compared OpenSM routing algorithms; with the parallel implementation running approximately an extra ten times faster when executed on twenty cores.

For clusters ranging up to many tens of thousands of nodes, Dmodc provides fast enough re-routing for a centralised fabric manager to react to faults before applications are interrupted.

4.2.4 . Congestion risk as a function of degradation

Analysis of the routing algorithm's quality under failure is undertaken similarly to existing similar research [20]. Random degradation is simulated on an 8640 node PGFT (with blocking factor of 4) using hundreds of throws for each considered routing algorithm and type of equipment to degrade (switches or links). The integer amount of equipment $a \in [0, 2^m[$ to remove at each throw is chosen using a shifted log-uniform distribution. This distribution is chosen to test degradation uniformly across multiple scales and include non-degraded tests; it is defined in the following formula using the uniformly random number $u \in [0, 1]$:

$$a \leftarrow \left\lfloor 2^{m \times u} - 1 \right\rfloor$$

The chosen amount of equipment is then randomly removed from the complete topology. The resulting degraded (or complete) topology is routed at this point, and linear forwarding tables are dumped for analysis.

Evaluation of these tables is performed using static metrics $\boldsymbol{\mu}$, $\boldsymbol{\mu}^{SP}$, and $\boldsymbol{\mu}^{RP(1000)}$, defined in Section 2.3. The effective diameter metric $\boldsymbol{\nu}$ is not studied because it does not change for shortest up–down routing in fat-trees and degraded fat-trees. To justify the choice of 1000 samples for $\boldsymbol{\mu}^{RP}$, we generated one instance of the 8640 node PGFT with 256 switches randomly removed; for which the $\boldsymbol{\mu}^{RP}$ values are the largest considered, generally around 20, and for which sampling variability is the highest. The standard deviation for 100 random runs of $\boldsymbol{\mu}^{RP(1000)}$ was measured at 0.96. This illustrates the stability of these runs and justifies the sampling choice. Justifying this study using standard deviation inspired the proposal to base an automatic stop criterion for $\boldsymbol{\mu}^{RP}$ in the *Random permutations* subsection in Section 2.3.2.

Congestion risk results are shown in Figures 4.6 and 4.7. When considering existing routing algorithms, Ftree provides the best performance for complete PGFTs (especially regarding SP for which the maximum congestion risk approaches theoretical optimal), but SSSP provides better stability under massive degradation, confirming results of the studies mentioned in Section 1.2.9. UPDN and MinHop provide visually identical results in this analysis: in fact, in a full PGFT they are equivalent and vary only slightly under degradation. They both provide comparatively poor results for SP and A2A throughout the observed scale, however for RP they surprisingly improve significantly under massive degradation.

Dmodc provides minimal congestion risk throughout the considered range of degradations when compared with existing oblivious algorithms. In particular, it is even more stable than Ftree for SP under minimal degradation and nearly as stable as SSSP for A2A and RP under massive degradation.

4.3 . Conclusion

The simulation results in Sections 4.2.3 and 4.2.3 show that Dmodc provides high-quality centralised fault-resilient routing for PGFTs at a fraction of the run time of existing algorithms, without relying on partial re-routing. Dmodc is also applicable to fat-tree-like topologies (as mentioned in Figure 4.2) but with lower quality load balancing. As defined here, no effort has been made to minimise the size of updates to be uploaded to switches throughout the fabric.

This method is patented [107]; and has been successfully deployed to an 8490 node PGFT production network in which it helps provide fault resilience, even when faced with thousands of simultaneous changes.

If one wants to minimise size of updates without interstate knowledge, one strategy is to pad NIDs. Existing research instead uses interstate knowledge by explicitely sorting LFTs compared with old ones [100]. Alternately, if the fabric manager is aware of placement of failed nodes, ghost nodes can be inserted to easily obtain the intended behaviour (similarly to place-keeper nodes in Section 3.2 in the original paper detailing Ftree [97]). The latter has been successfully implemented, but not studied for quality under degradation.

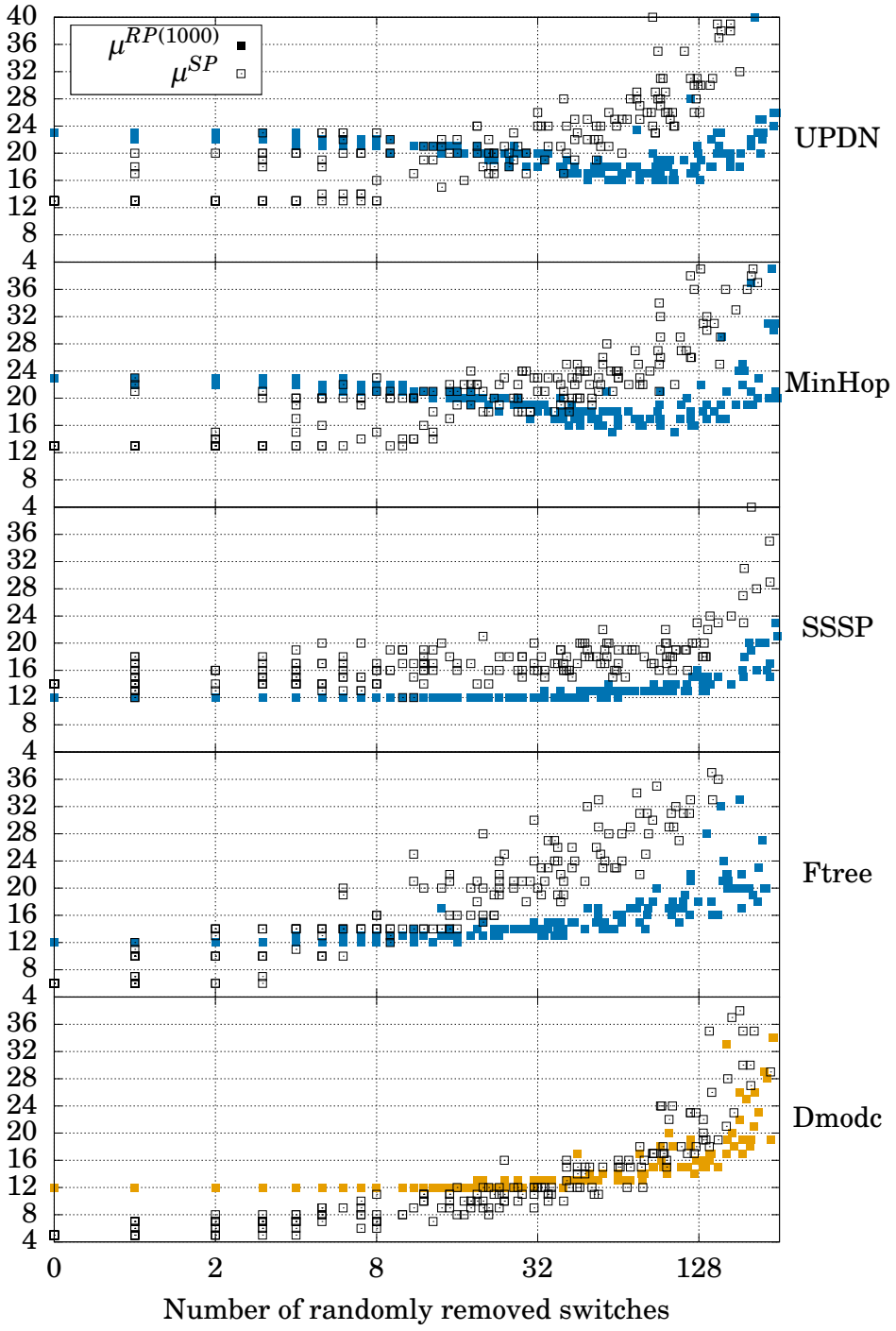


Figure 4.6: Maximum congestion risk in an 8640 node PGFT with blocking factor of 4, under random switch degradation (in log scale; lower is better).

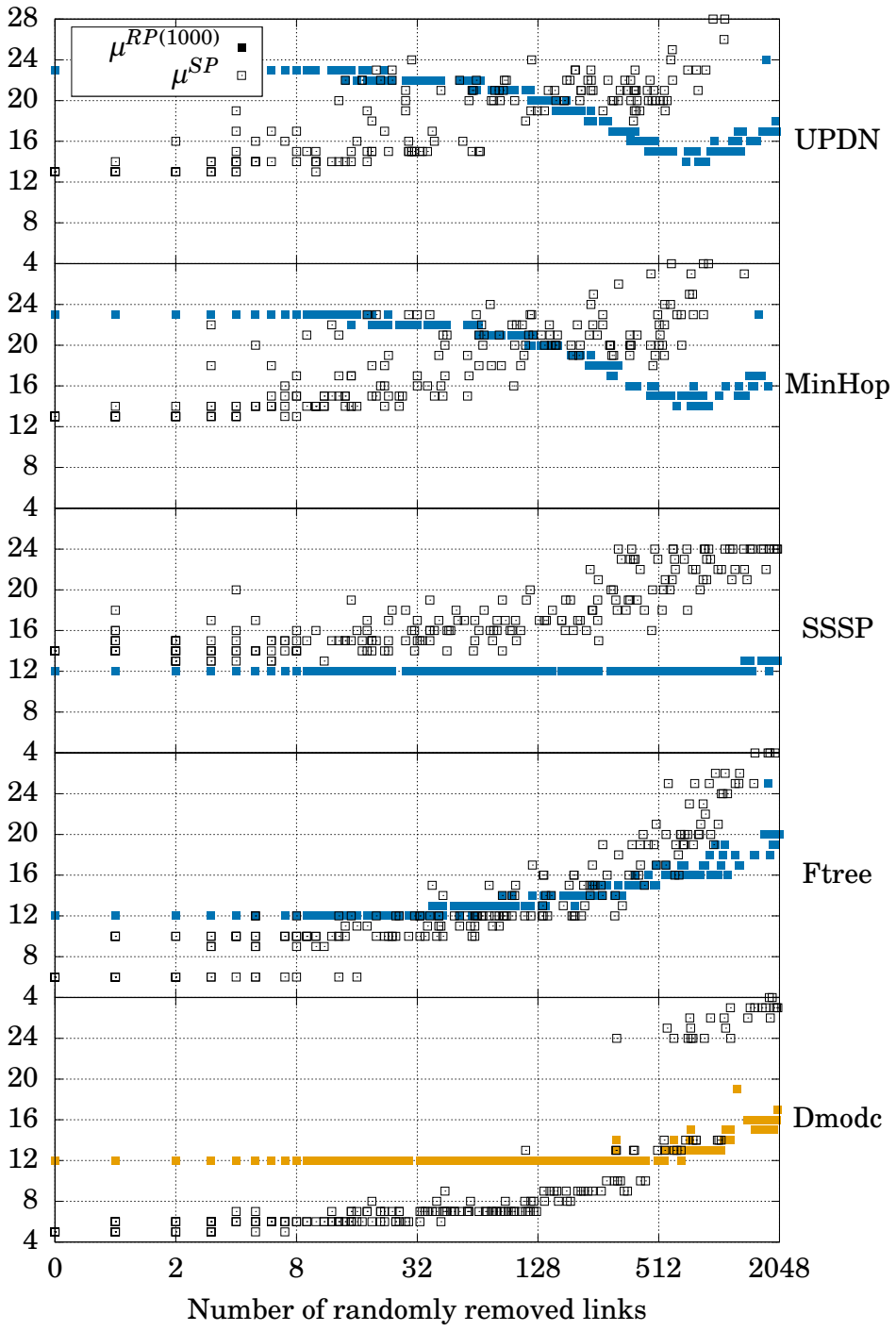


Figure 4.7: Maximum congestion risk in an 8640 node PGFT with blocking factor of 4, under random link degradation (in log scale; lower is better).

5 — Routing irregular fat-trees

As described in Section 1.3.4, there are various common variants to canonical fat-trees, either with precise definitions for a specific purpose (QFTs, MFs), simply taking into account incomplete FTs (due to failure or design constraints), or finally taking into account core design differences. These Irregular Fat-Trees (IFTs) describe variants to PGFTs, whose variations result either from degradation or from structural change to cabling logic. For the sake of formality, IFT-specific routing algorithms must however remain functional for a generalised class of BMINs, wherein leaves can be in any level, as long as every pair of leaves is connected via an up–down path. The more irregularity is introduced, the less optimally balanced IFT-specific routing can be expected. Some of the work presented in Section 1.2.9 regarding fault-resilient routing for fat-trees is applicable, with varying resulting levels of quality, to IFTs. Furthermore, existing algorithms can be extended at little cost to accept IFTs with leaves in levels higher than others, in part by ranking from top switches as defined in Section 1.4.5.

Section 5.1 will first present novel attempts at efficiently routing IFTs. Section 5.2 will then provide some quantitative quality comparisons of these algorithms and existing ones, using the congestion risk metric methodology fleshed out in the previous chapters.

5.1 . Some new IFT-specific routing algorithms

5.1.1 . Ftree-Random

As an alternative to fat-tree specific algorithms, random algorithms can be applied. Some research uses random routing in fat trees [37, 29, 73] and degraded fat-trees as a benchmark. Random routing in general refers to random choices within shortest paths, called RandSP in Section 1.4.4; it is therefore deadlock-free if shortest path routing is deadlock-free (such as in PGFTs wherein shortest paths are up–down and consequently deadlock-free) but not in general. Random routing can also refer to a specific fat-tree implementation where each pair of nodes is routed through one of their NCAs, explicitly selected at random. The benefits of random routing are complete agnosticism, ease of implementation, and speed of execution. The problem in this version of random routing is that congestion is highly probable, due to the extremely probable lack of correlated route coalescing.

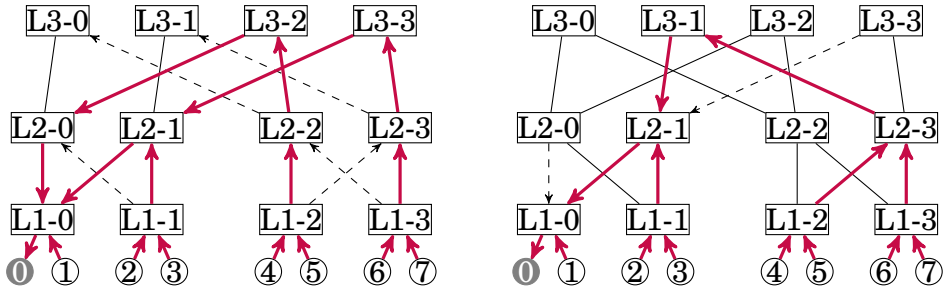
A potential improvement would be to coalesce routes to the same destination, similarly to Ftree and Dmodc, while randomly spreading subtrees to distinct destinations. There is such an algorithm (called Random-NCA-Down [73]), which applies Dmodk after randomizing numbering. This algorithm presents

benefits in terms of performance, but is only defined in the case of undegraded PGFTs since it directly uses Dmodk. Ftree-Random is a similar but novel fat-tree-specific random routing algorithm which trades off some agnosticism in exchange for correlated route coalescing; however, while also providing fault-resilience. This algorithm, defined in Algorithm 8, is a small modification to Ftree, with the same traversal behaviour but using random choices instead of usage counters. It could, additionally, be implemented closer to Random-NCA-Down using the cost pre-computation steps of Dmodc but with a random port selection during the parallel routes computation phase.

Algorithm 8: Ftree-Random

```
1  foreach leaf switch l ∈ L do
2      foreach endport p of l do
3          n is the node connected to p
4          δ is the NID of n
5          LFT_l[δ] ← p
6          Δ ← #L
7          Ascend(l, δ)
8  Function Ascend(s, δ)
9      foreach up port u of s in random order do
10         a is the remote switch (above s) connected to p
11         if δ ∉ LFT_a then
12             d is the remote port (of a) connected to p
13             LFT_a[δ] ← d
14             Descend(s, δ)
15             Ascend(a, δ)
16             if Δ = 0 then Break
17 Function Descend(s, δ)
18     if s ∈ L then Δ ← Δ − 1
19     foreach down port d of s in random order do
20         b is the remote switch (below s) connected to p
21         if δ ∉ LFT_b then
22             u is the remote port (of b) connected to p
23             LFT_b[δ] ← u
24             Descend(s, δ)
25             if Δ = 0 then Break
```

Figure 5.1 shows an example situation to highlight how RandSP might strongly spread out routes to a destination, while illustrating the choices available to Ftree-Random during its selection. Of course, the selected example



(a) RandomSP (b) Ftree-Random

Figure 5.1: Example routes towards the first destination in a 2-ary 3-tree. Discarded choices are shown in dashed arrows, secondary routes are ignored.

for RandomSP has particularly uncoalesced routes to exacerbate the potential congestion risk with the routes to other destinations. Furthermore, note that nothing stops Ftree-Random from selecting overlapping routes going towards the second destination, potentially also resulting in avoidable congestion risk.

Before providing any quantitative analysis of Ftree-Random, we can both expect it to perform better for the target $\boldsymbol{\mu}$ metrics than non-coalescing random algorithms, and also expect it to perform worse than Ftree/Dmodc for $\boldsymbol{\mu}^{SP}$ metrics on regular Fat-trees. Indeed, similarly to the analysis of RandSP in Section 3.3.3, it is nearly impossible for Ftree-Random to perfectly spread subtrees on available resources by chance.

5.1.2 . Ftree2

As described in Section 1.4.1, Ftree is often used in degraded or slightly irregular fat-trees, but it has some limitations in terms of quality. We propose a cheap approach to largely overcome these limitations, by modifying Ftree slightly, taking into consideration remote switch usage before local port usage, as defined in Algorithm 9. This is a novel contribution called Ftree2. Like Ftree, the up port usage counters can be implemented using one least-recently-used up port index per switch. On the other hand, asc-usage and desc-usage counters are remotely accessed from multiple entry points. They must therefore be implemented using individual counters and be explicitly accessed by remote switches. The overall aim of Ftree2 is to provide a compromise between the simplicity of Ftree, and the quality of Dmodc.

Figure 5.2 presents Ftree2 routes in the same case (network G_d) as Figure 1.7b (which presented Ftree routes). This example shows how Ftree2 balances routes better throughout the network by using remote information early: in Figure 1.7b, we can see that 3 distinct routes go from L1-2 and L1-3 to L2-2, while in Figure 5.2, we can see that 2 distinct routes go from L1-2 and L1-3 to L2-2 and L2-3. Formulated differently:

Algorithm 9: Ftree2

foreach *switch* $s \in S$ **do**
    Reset the asc-usage counter of s
    Reset the desc-usage counter of s
    foreach *up port* u *of* s **do**
        Reset the usage counter of u
foreach *leaf switch* $l \in L$ **do**
    foreach *endport* p *of* l **do**
        n is the node connected to p
        δ is the NID of n
        $\mathrm{LFT}_l[\delta] \leftarrow p$
        $\Delta \leftarrow \#L$
        `Ascend`(l,δ)
Function `Ascend`(s,δ)
    $U \leftarrow$ ordered up ports of s, sorted first by remote switches' asc-usage, second by their own usage
    foreach $u \in U$ *in increasing order* **do**
        a is the remote switch (above s) connected to p
        if $\delta \notin \mathrm{LFT}_a$ **then**
            d is the remote port (of a) connected to p
            $\mathrm{LFT}_a[\delta] \leftarrow d$
            Update usage counter of u
            Update asc-usage counter of a
            `Descend`(s,δ)
            `Ascend`(a,δ)
            if $\Delta = 0$ **then Break**
Function `Descend`(s,δ)
    if $s \in L$ **then** $\Delta \leftarrow \Delta - 1$
    $D \leftarrow$ ordered non-endport down ports of s, sorted first by remote switches' desc-usage, second by their own usage
    foreach $d \in D$ *in increasing order* **do**
        b is the remote switch (below s) connected to d
        if $\delta \notin \mathrm{LFT}_b$ **then**
            u is the remote port (of b) connected to d
            $\mathrm{LFT}_b[\delta] \leftarrow u$
            Update usage counter of d
            Update desc-usage counter of b
            `Descend`(b,δ)
            if $\Delta = 0$ **then Break**

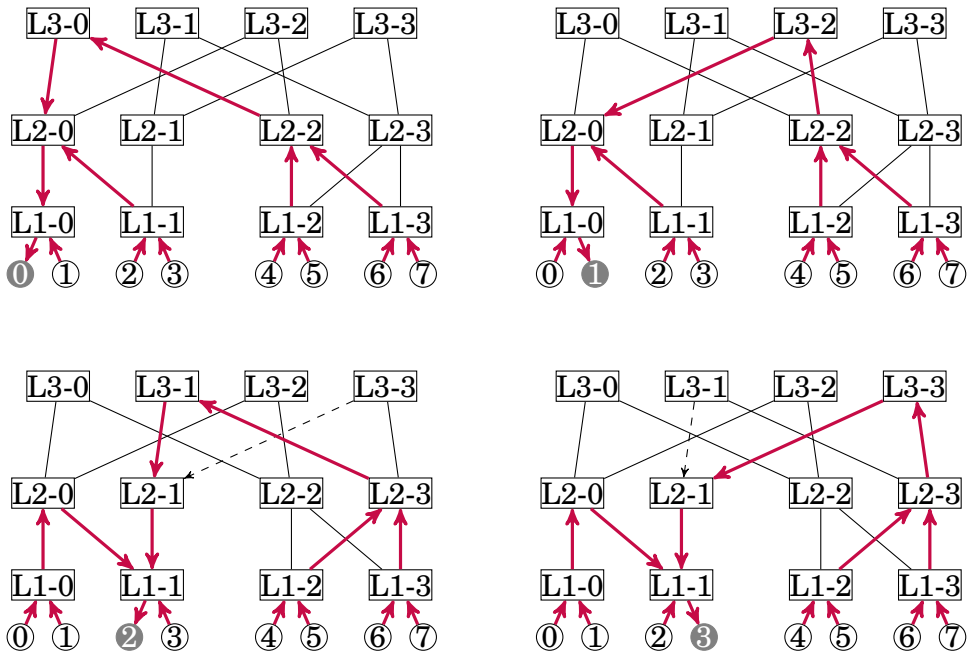


Figure 5.2: Ftree2 routes in a 2-ary 3-tree (under failure of L1-0 ↔ L2-1) towards the first four destinations. Dashed arrows represent "secondary" routes that are computed but never used.

$$\mu_{G_d}(A2A(Ftree)) = 3$$
$$\mu_{G_d}(A2A(Ftree2)) = 2$$

Considering remote switch usage helps to build a more thorough view of the overall available resources, at a controlled cost. It is reasonable to imagine that for fat-trees over 3 levels in height, this approach might be insufficient, and an even more thorough approach considering resource usage one step beyond could be envisioned instead. For the foreseeable future, however, 4-level fat-trees are not being constructed, due to their high diameter and hard-to-justify network cost. In this context, Ftree2 might prove sufficient to improve real-life performance in the majority of actively used fat-tree topologies.

5.1.3 . Up*/Down* implementations

As an alternative approach to relaxing topological constraints in topology-specific algorithms, a generic agnostic routing algorithm can be specialised using existing fat-tree-specific load-balancing techniques.

This agnostic routing algorithm is Up*/Down* [77], which has been extensively studied and modified in existing research. Up*/Down* guarantees deadlock-free routing in any connected network by first constructing a directed spanning tree with an arbitrary switch as root node. Second, any routing function whose paths are up–down with regards to the directed spanning tree is guaranteed deadlock-free. It is up to specific implementations to decide on traversal method, such as breadth-first search (BFS), depth-first search (DFS); and route selection method, such as MinHop.

This section describes novel attempts at incorporating the previously described topology-specific algorithms inside directed spanning trees, with the aim of designing a routing algorithm both providing competitive load balance in fat-trees and degrading gracefully in any irregular topology. The attempts use a BFS spanning tree because it's the most widely used in existing literature and because it preserves BMIN direction and structure throughout most of the network in PGFTs when the root node is selected among leaf switches. Further work could introduce variants using DFS and the heuristics introduced by Sancho et al. [74].

The BFS method that was settled on only keeps strictly oriented links (no same level link), and is defined in Algorithm 10.

All implementations use precomputed costs and downcosts, as was described in Section 4.2.1. The complete method to determine such data is given in Algorithm 6, using link directions provided by the BFS.

BFS-Random The first implementation of Up*/Down* to consider is the most agnostic one, BFS-Random, providing deadlock-free random shortest-path routing. It is an obvious implementation which we present as a base-level alongside the other (novel) implementations of Up*/Down*, in a fair context. BFS-Random is defined in Algorithm 11. Note that it is equivalent to RandSP (defined in Section 1.4.4) in BMINs when the root of the BFS is selected among leaf switches. However, it also provides deadlock-free routing for any topology.

Algorithm 10: Breadth-First Search (BFS)

Data: a root switch L_0 (selected from L)
Result: directions for all links

$D \leftarrow \emptyset$ /* Set of *done* switches */
$T \leftarrow \{L_0\}$ /* Set of *to-do* switches */
while $T \neq \emptyset$ **do**
  $N \leftarrow \emptyset$ /* Set of *next* switches */
  foreach $s \in T$ **do**
    foreach $g \in G_s$ **do**
      if $\Omega_s \in D$ **then**
        Add $s \multimap \Omega_s$ to spanning tree
      else
        Add $s \rightarrowtail \Omega_s$ to spanning tree
        if $\Omega_s \notin T$ **then**
          $N \leftarrow N \cup \{\Omega_s\}$
    $D \leftarrow D \cup \{s\}$
  $T \leftarrow N$

Algorithm 11: BFS-Random

Data: link directions from BFS (Algorithm 10)
Data: costs and downcosts (Algorithm 6)

foreach $s \in S$ **do**
  foreach $d \in N$ **do**
    $C_{s,\lambda_d} \leftarrow \emptyset$
    foreach $g \in G_s$ **do**
      $s' \leftarrow \Omega_g$
      if *($s' \multimap s$ and $c'_{s',\lambda_d} < c'_{s,\lambda_d}$) or ($s' \rightarrowtail s$ and $c_{s',\lambda_d} < c_{s,\lambda_d}$)* **then**
        $C_{s,\lambda_d} \leftarrow C_{s,\lambda_d} \cup \{g\}$
    if $\#C_{s,\lambda_d} > 0$ **then**
      $p_{s,d} \leftarrow$ random port from C_{s,λ_d}

BFS-Dmodc BFS-Dmodc is a novel variant of Dmodc which guarantees up–down paths, and is simple to implement. It is detailed in Algorithm 12 in the context of BFS-computed link directions. Note that this implementation is given instead of just relying on the existing computation phase of Dmodc from Section 4.2.2, because that one does not guarantee up–down paths.

Algorithm 12: BFS-Dmodc

Data: link directions from BFS (Algorithm 10)
Data: costs, downcosts and dividers (Algorithm 6)
foreach $s \in S$ **do**
  foreach $d \in N$ **do**
    $C_{s,\lambda_d} \leftarrow \emptyset$
    foreach $g \in G_s$ **do**
      $s' \leftarrow \Omega_g$
      if *($s' \dashrightarrow s$ and $c'_{s',\lambda_d} < c'_{s,\lambda_d}$) or ($s' \rightsquigarrow s$ and $c_{s',\lambda_d} < c_{s,\lambda_d}$)* **then**
        $C_{s,\lambda_d} \leftarrow C_{s,\lambda_d} \cup \{g\}$
    if $\#C_{s,\lambda_d} > 0$ **then**
      $P_{s,d} \leftarrow \{p \in g \mid g \in C_{s,\lambda_d}\}$
      $g_{s,d} \leftarrow C_{s,\lambda_d}[\left\lfloor \frac{d}{\Pi_s} \right\rfloor \bmod \#C_{s,\lambda_d}]$
      $p_{s,d} \leftarrow g_{s,d}[\left\lfloor \frac{d}{\Pi_s \times \#C_{s,\lambda_d}} \right\rfloor \bmod \#g_{s,d}]$

Ftree-based BFS implementations Ftree, Ftree2, and Ftree-Random follow the same structure which is immediatly compatible with Up*/Down* (since they only use link direction). From this, three novel implementations of Up*/Down* are provided, using link directions from the BFS (Algorithm 10):

- BFS-Ftree (see Algorithm 1),
- BFS-Ftree2 (see Algorithm 9),
- BFS-Ftree-Random (see Algorithm 8).

Annex 3 presents an alternative approach to routing one kind of IFTs, wherein virtually merging switches in a specific way results in regular PGFTs. This approach was left out of this chapter since it is not generalisable and intrinsically leaves out available resources, but it is still novel.

5.2 . Comparison of algorithms

5.2.1 . Progressively degraded fat-tree

Similarly to the comparison work done in Section 4.2.4, BFS algorithms are compared in a sample 8490-node fat-tree topology with a blocking factor of 4, which corresponds to the real system on which the 8640-node PGFT was based, but taking missing nodes into account. These nodes are missing from the real system due to restrictions on ports imposed by that generation of BXI switches, but similar situations are found in many other real fat-tree systems [6]. Here, the comparison was done on the same progressively randomly degraded network, and not on individually degraded networks at random amounts (as in Section 4.2.4); this choice is discussed below. Twelve algorithms are considered in this study, including 3 from OpenSM, shown in blue in the top row, and 9 from BXI FM, shown in yellow (including the 7 presented in this Chapter). Figures 5.3 and 5.4 show static metrics for the computed routes on the same network on which switches are randomly removed one by one. Figures 5.5 and 5.6 show static metrics for the computed routes on the same network on which links are randomly removed one by one.

The choice of computing congestion risk as degradation is progressively applied was made in order to improve intuitive understanding of the simulation behaviour. One downside to this method is that the analysis might be biased due to the specific random sequence considered. However, there isn't any specific bias known to favour any algorithm, and a second random sequence was added, in order to give some weight to the idea that comparison is fair even if the specific evolution of congestion risk is potentially not representative of the quality behaviour under degradation in general. Indeed, though congestion risk behaves slightly differently from one random sequence to another (e.g. $\boldsymbol{\mu}^{SP}_{G_{FT}}(Dmodc) < \boldsymbol{\mu}^{RP(1000)}_{G_{FT}}(Dmodc)$ holds throughout the second switch-degradation sequence, and not in the first sequence), comparisons between algorithms hold across sequences (e.g. BFS-Ftree-Random performs worse than Ftree-Random for all considered cases). The choice of progressive random degradation versus random-amount random degradation is subjective and the latter disperses bias to some extent, though only a single statistical value per metric (e.g. the median) can be comfortably displayed; while in the former each metric can be displayed with multiple statistical values (e.g. the median with errorbars at chosen quantiles).

The supplementary 1st and 39th 40-quantiles provided in Figures 5.3 to 5.10 show 95% credible intervals for $\boldsymbol{\mu}^{RP}$ metrics. This gives us more detail as to how performance evolves. We can observe that the 1st 40-quantile is at most very slightly lower than the median in all studied cases, while the 39th 40-quantile is sometimes significantly larger than the median. We can conclude that regardless of routing function, random permutations mostly perform homogeneously, with a proportion performing worse, to varying degrees

	best for $\boldsymbol{\mu}^{RP}_{G_{FT}}$	best for $\boldsymbol{\mu}^{SP}_{G_{FT}}$	also good
agnostic	SSSP	BFS-Ftree2	BFS-Random
topology-specific	Dmodc	Dmodc, Ftree2	Ftree (BXI)

	worst for $\boldsymbol{\mu}^{RP}_{G_{FT}}$ and $\boldsymbol{\mu}^{SP}_{G_{FT}}$	also bad
agnostic	BFS-Ftree-Random, BFS-Ftree	BFS-Dmodc
topology-specific	UpDn	

Table 5.1: Summary of results for G_{FT}

depending on routing function.

From Figures 5.3 and 5.4, we infer that in the considered fat-tree with switch degradations only, the algorithms providing best performance even under large-scale degradation are SSSP, Dmodc, BFS-Random, and BFS-Dmodc for random permutations (each with $\boldsymbol{\mu}^{RP(1000)}_{G_{FT}} < 15$ for all considered cases); and Dmodc, Ftree2, and BFS-Ftree2 for shift permutations (each with $\boldsymbol{\mu}^{SP}_{G_{FT}} < 15$ for all considered cases). From Figures 5.5 and 5.6, we infer that in the same case but with link degradations only, the algorithms providing good performance even under large-scale degradation are all but UpDn, BFS-Ftree-Random, and BFS-Ftree for random permutations; and for shift permutations, Dmodc performs significantly better than all the other algorithms in both sequences (with $\boldsymbol{\mu}^{SP}_{G_{FT}}(Dmodc) \leq 10$ in all considered cases).

To summarise these simulations, the all-around best-performing and most resilient algorithms are presented in Table 5.1, as well as the worst-performing and least resilient algorithms. In this table, SSSP is defined as an agnostic algorithm, which is technically untrue, but its results are equivalent to DFSSSP (which is entirely topology agnostic) in the target topologies.

This case study foremost outlines an example use of the congestion metric presented in Chapter 2 in a methodology to perform a quantitative evaluation of potential routing algorithms for a given network under potential failure scenarios. Secondarily, the algorithms performing poorly on this rather simple target network can be written off for the general case right away as not being competitive enough. However, this case study is not meant as a way to determine the best algorithms for the general case based on its specific results.

The worst performing algorithms presented in Figures 5.3 to 5.6 are BFS-Ftree-Random, BFS-Ftree, BFS-Dmodc, and UpDn (which performs particularly poorly in the undegraded network).

	best for $\mu^{RP}_{G_{QFT}}$	best for $\mu^{SP}_{G_{QFT}}$
agnostic	SSSP, BFS-Ftree2	SSSP
	BFS-Random	BFS-Ftree2
topology-specific	Ftree, Ftree2	Ftree, Ftree2

	worst for $\mu^{RP}_{G_{QFT}}$ and $\mu^{SP}_{G_{QFT}}$
agnostic	BFS-Dmodc
topology-specific	UpDn

Table 5.2: Summary of results for G_{QFT}

5.2.2 . Progressively degraded QFT

A second case study was conducted on a QFT called G_{QFT} under random degradation to extend the range of IFTs studied here. This QFT topology corresponds to the network of a real supercomputer constructed by Atos. It was sold as a 3-level fat-tree, and it is based on 2-level fat-trees packaged as 648-port *director switches* used for the higher levels (chosen for their cheaper cost compared with bulk switches and links from the external provider). The interconnection pattern used between the L2 switches and the L1 leaf switches does not respect PGFT logic, since it provides multiple switch-paths between L3 top switches and L1 leaves. Instead, it corresponds to QFT cabling, since the lower level is properly organised in complete bipartite subgroups. This situation is similar to that of the JuRoPA supercomputer [6], and other real-world IFTs designed around hardware constraints.

As discussed in the previous section, the comparison was also done on the same progressively randomly degraded network, and not on individually degraded networks at random amounts. The same twelve algorithms are considered in this study. Figures 5.7 and 5.8 show static metrics for the computed routes on the same network on which switches are randomly removed one by one. Figures 5.9 and 5.10 show static metrics for the computed routes on the same network on which links are randomly removed one by one.

We can first observe that quantiles overall appear to behave identically to the first case study. From Figures 5.7, 5.8, 5.9 and 5.10, we infer that in G_{QFT} with either switch faults or link faults, the algorithms providing best performance even under large-scale degradation are SSSP, Ftree2, BFS-Ftree2, Ftree, and BFS-Random for random permutations (each with $\mu^{RP(1000)}_{G_{QFT}} < 8$ for all considered cases); and Ftree, Ftree2, and BFS-Ftree2 for shift permutations (each with $\mu^{SP}_{G_{QFT}} < 8$ for all considered cases, and a perfect $\mu^{SP}_{G_{QFT}} = 3$ in the complete network).

To summarise these simulations, the all-around best-performing and most resilient algorithms are presented in Table 5.2, as well as the worst-performing and least resilient algorithms.

Among the algorithms which performed particularly poorly for G_{FT}, both BFS-Ftree-Random and BFS-Ftree performed much less poorly on G_{QFT} (though this specific result does not qualify them as viable algorithms in the general case), while BFS-Dmodc and UpDn remained as the only particularly poor performers. Among the Up*/Down* implementations considered, BFS-Random appears to be a reasonable pure agnostic choice, and BFS-Ftree2 as surprisingly competitive with fat-tree-specific algorithms while being functionally topology-agnostic.

5.3 . Conclusion

Real-world interconnection networks blur the precise lines of topology definition, and require more effort to determine adapted routing techniques. This work showcases a methodology to quantitatively compare congestion risk for a set of routing algorithms, given one network. For both an existing and a new selection of agnostic and topology-specific algorithms, we showed how varied congestion risk evolved as random degradation accumulated. This makes it simpler to discern viable routing algorithms by reducing the risk of congestion to only a couple of comparable values. Other aspects might be considered as well, such as execution run time, composability with other techniques, etc.

This methodology allowed us to show how some of the novel algorithms (namely BFS-Ftree-Random, BFS-Ftree, and BFS-Dmodc) did not perform competitively on the realistic case studies, and can therefore be written off from general use. On the other hand, the other novel algorithms (in particular Ftree2, BFS-Ftree2, and BFS-Random) show promising capabilities in the considered case-studies.

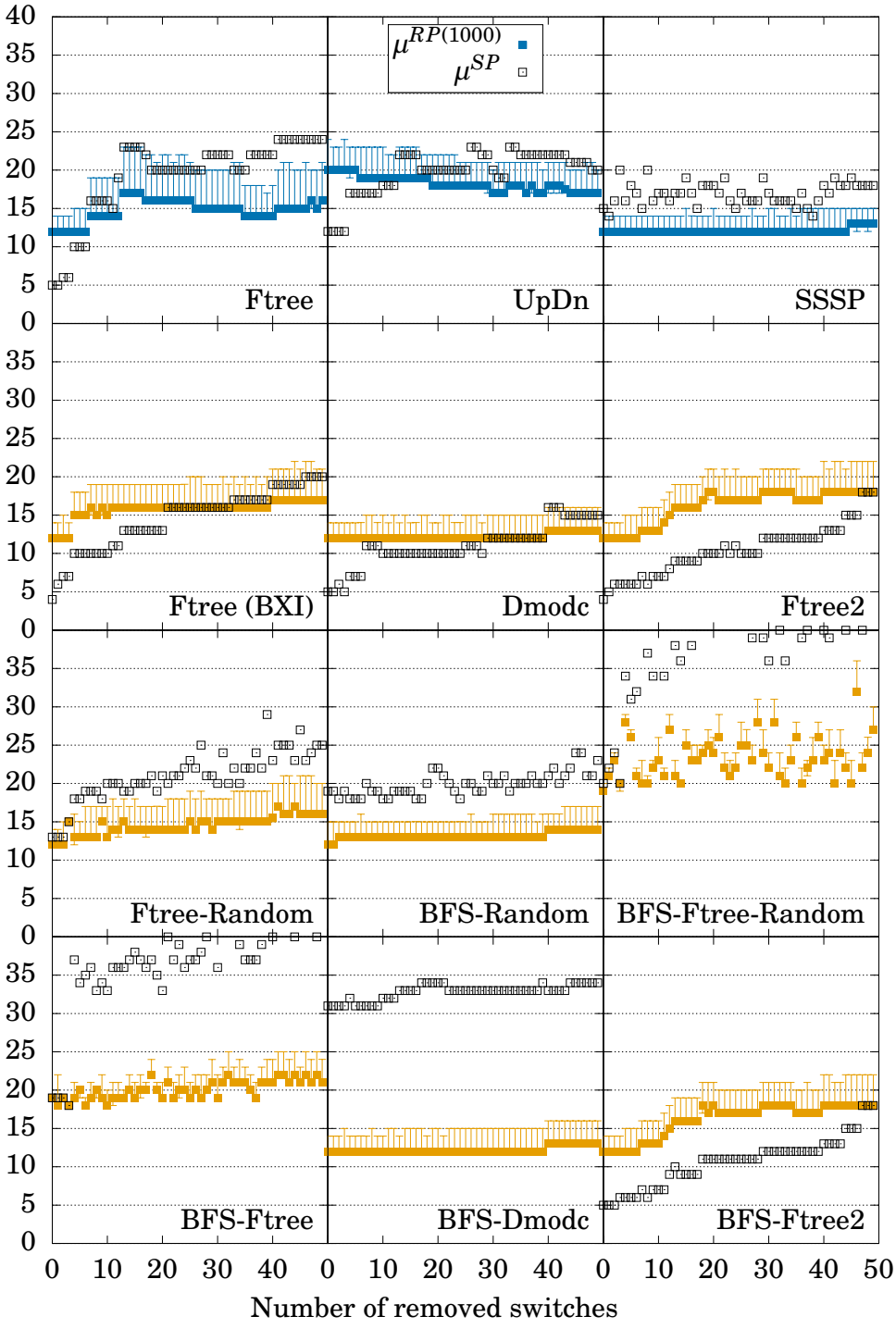


Figure 5.3: First sequence of progressively randomly switch-degraded 8490-node fat-tree (G_{FT})

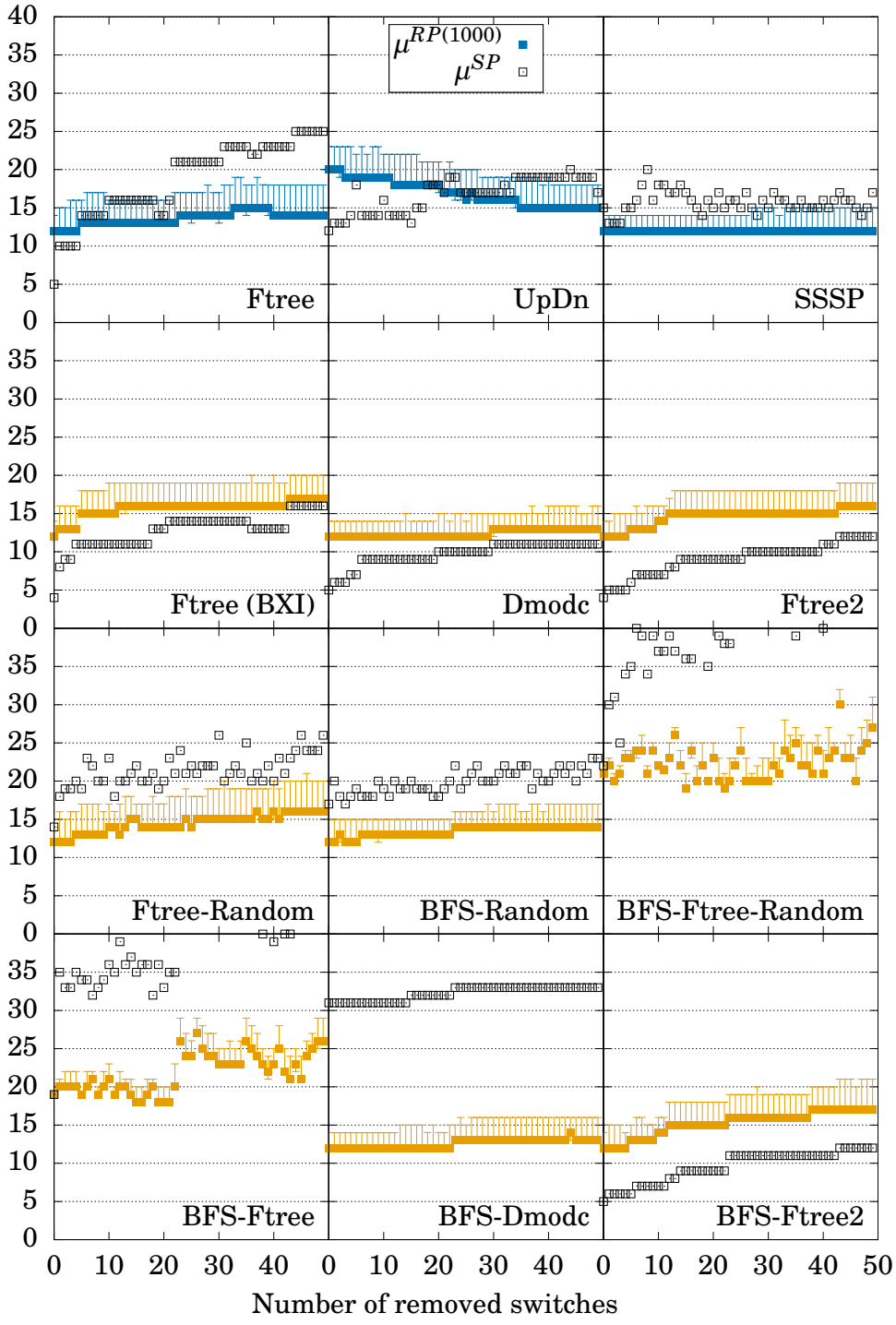


Figure 5.4: Second sequence of progressively randomly switch-degraded 8490-node fat-tree (G_{FT})

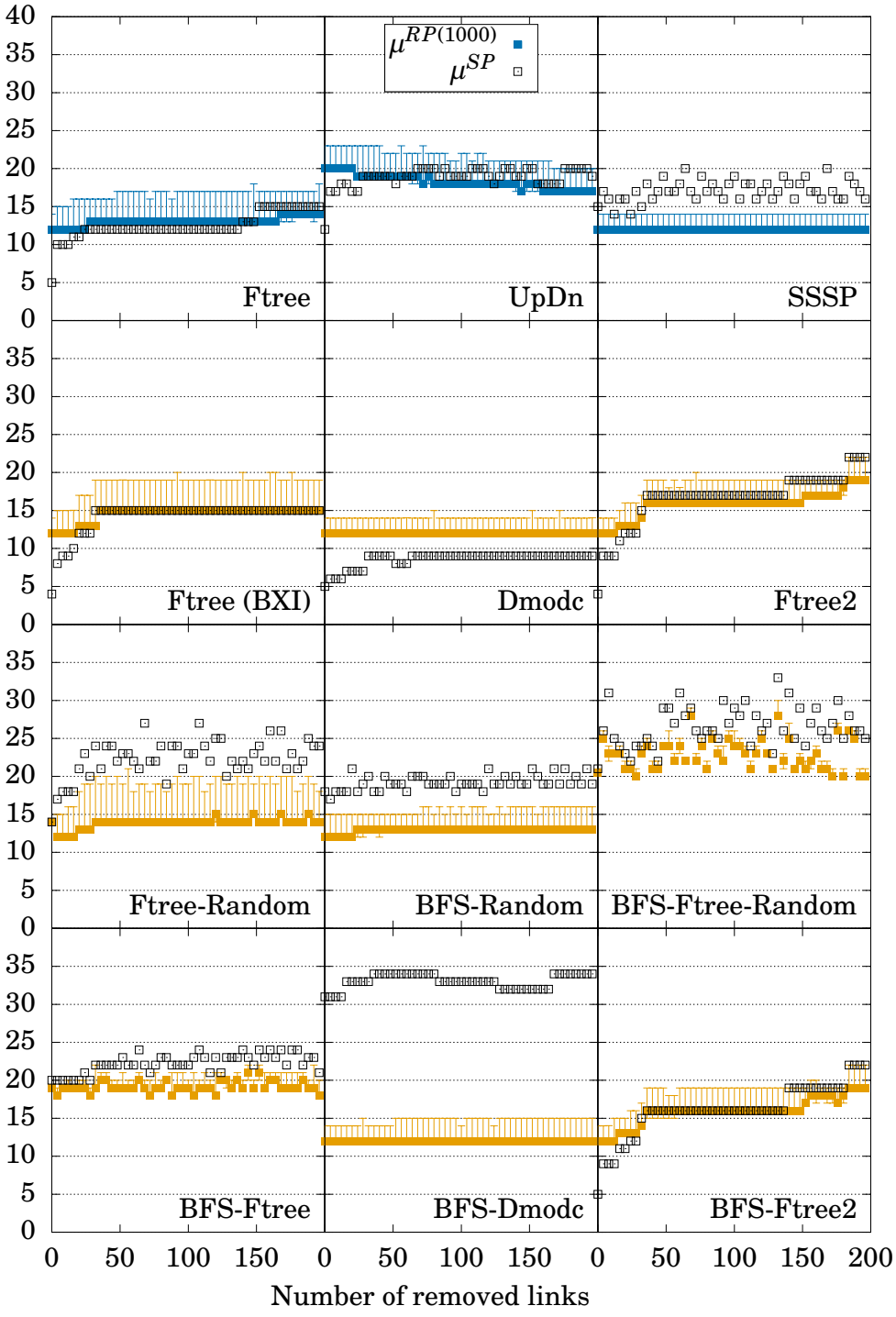


Figure 5.5: First sequence of progressively randomly link-degraded 8490-node fat-tree (G_{FT})

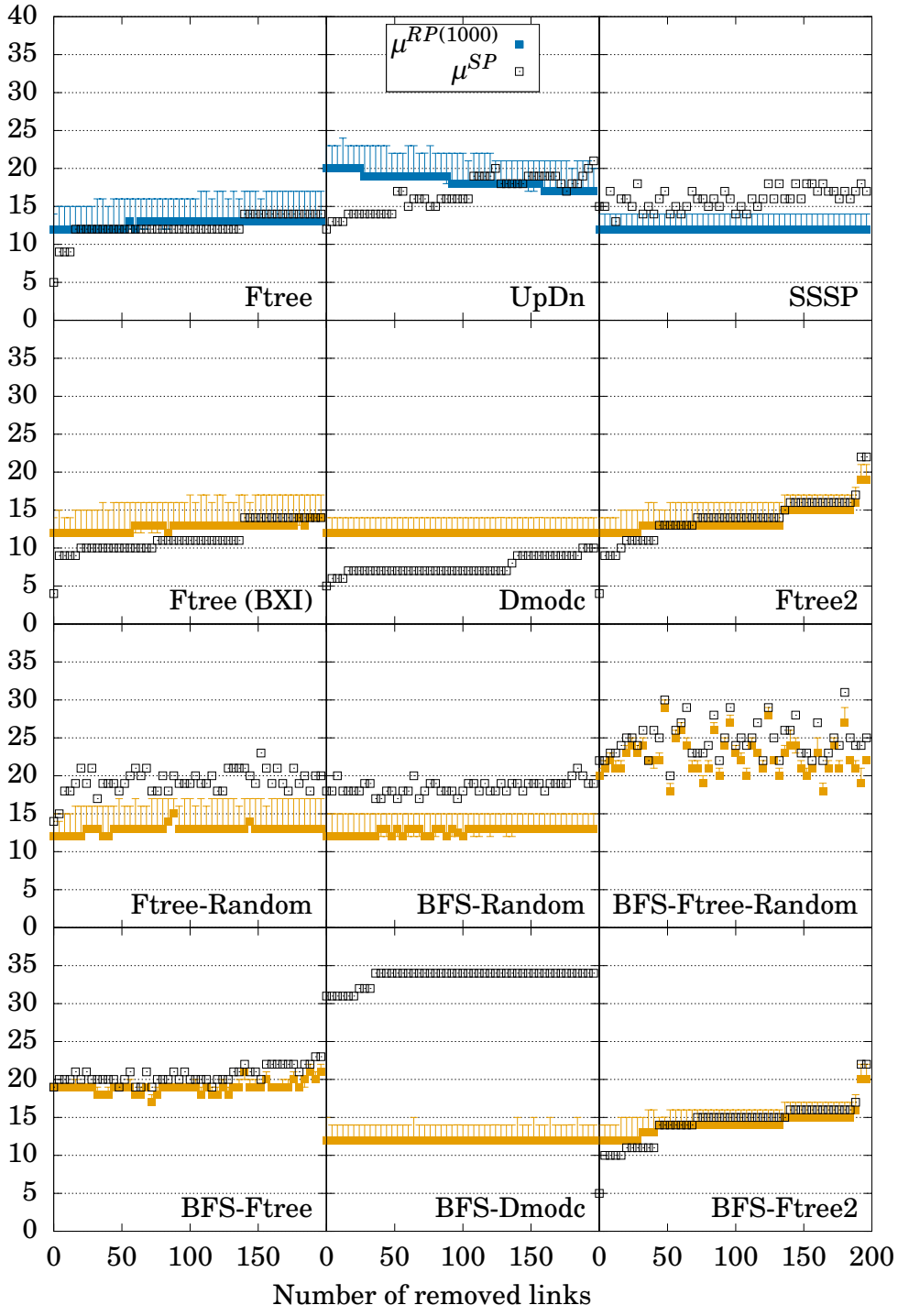


Figure 5.6: Second sequence of progressively randomly link-degraded 8490-node fat-tree (G_{FT})

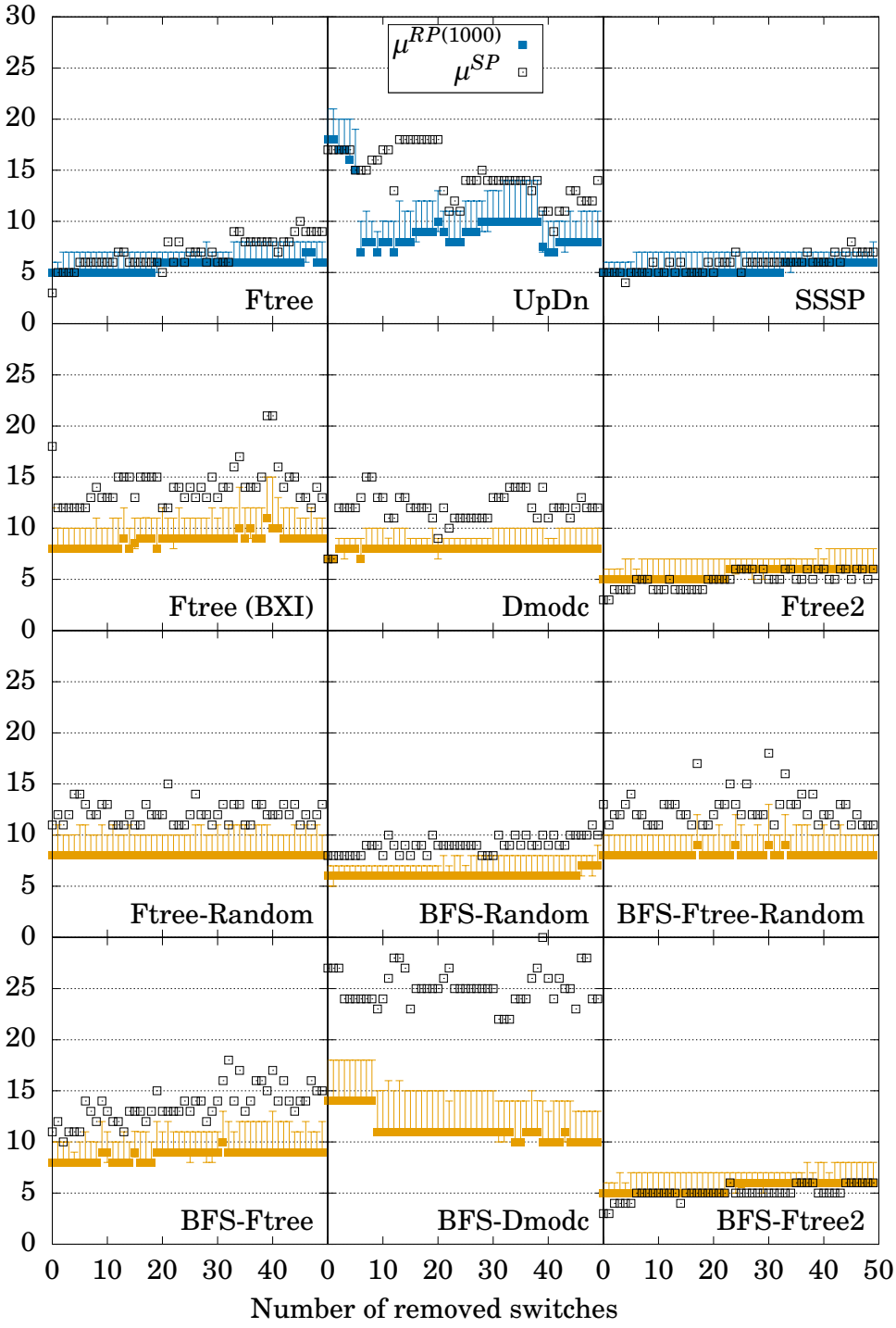


Figure 5.7: First sequence of progressively randomly switch-degraded 1036-node quasi-fat-tree (G_{QFT})

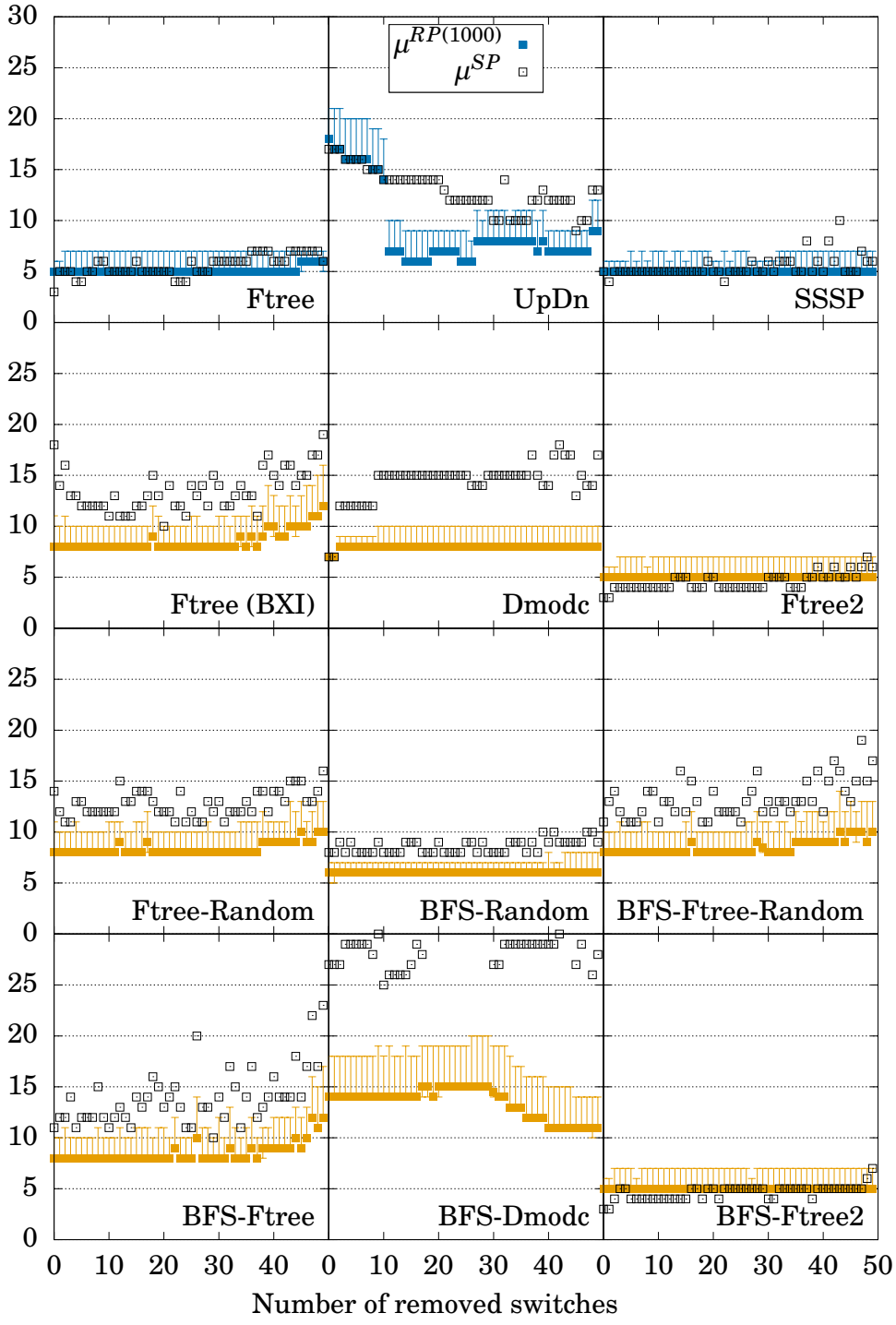


Figure 5.8: Second sequence of progressively randomly switch-degraded 1036-node quasi-fat-tree (G_{QFT})

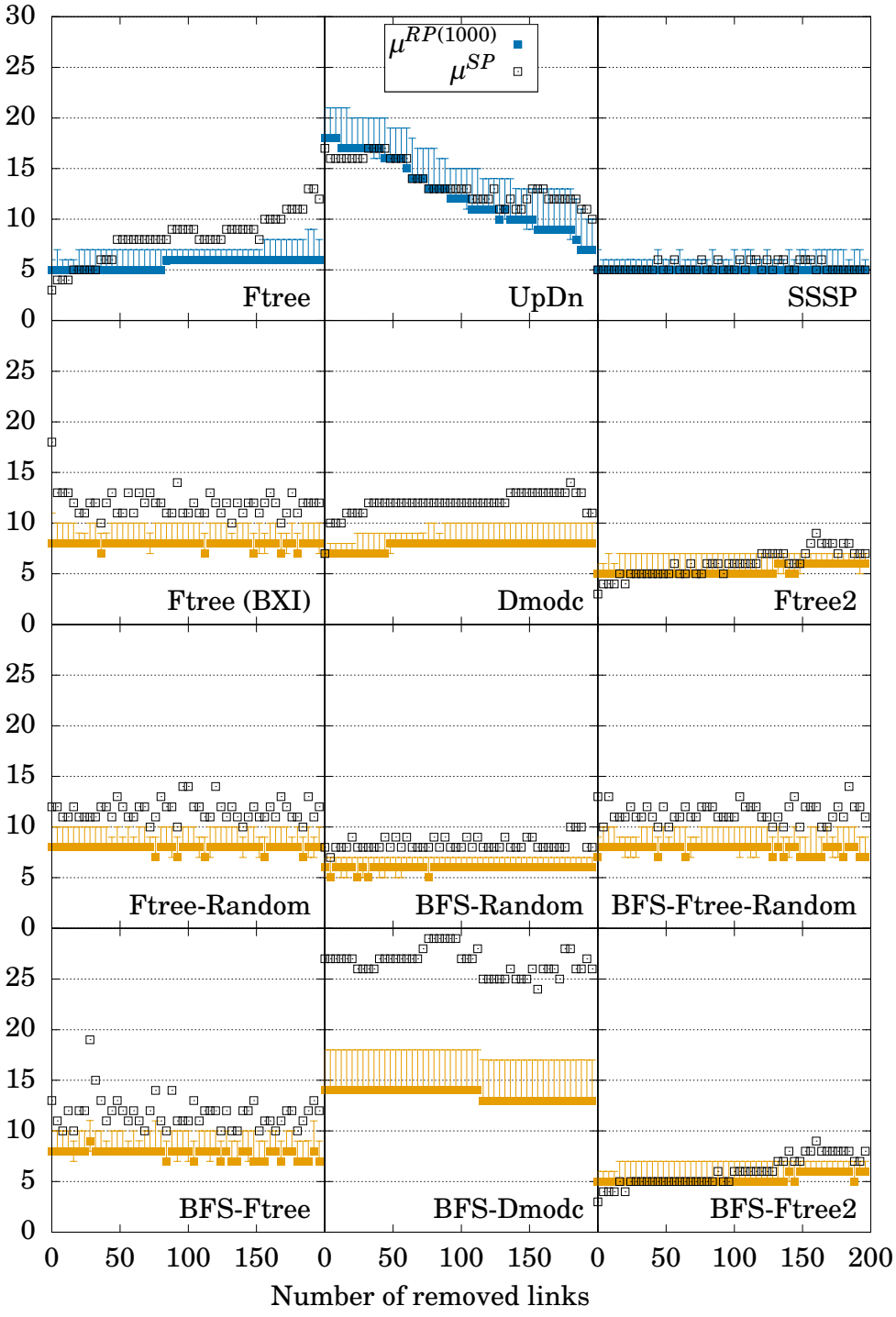


Figure 5.9: First sequence of progressively randomly link-degraded 1036-node quasi-fat-tree (G_{QFT})

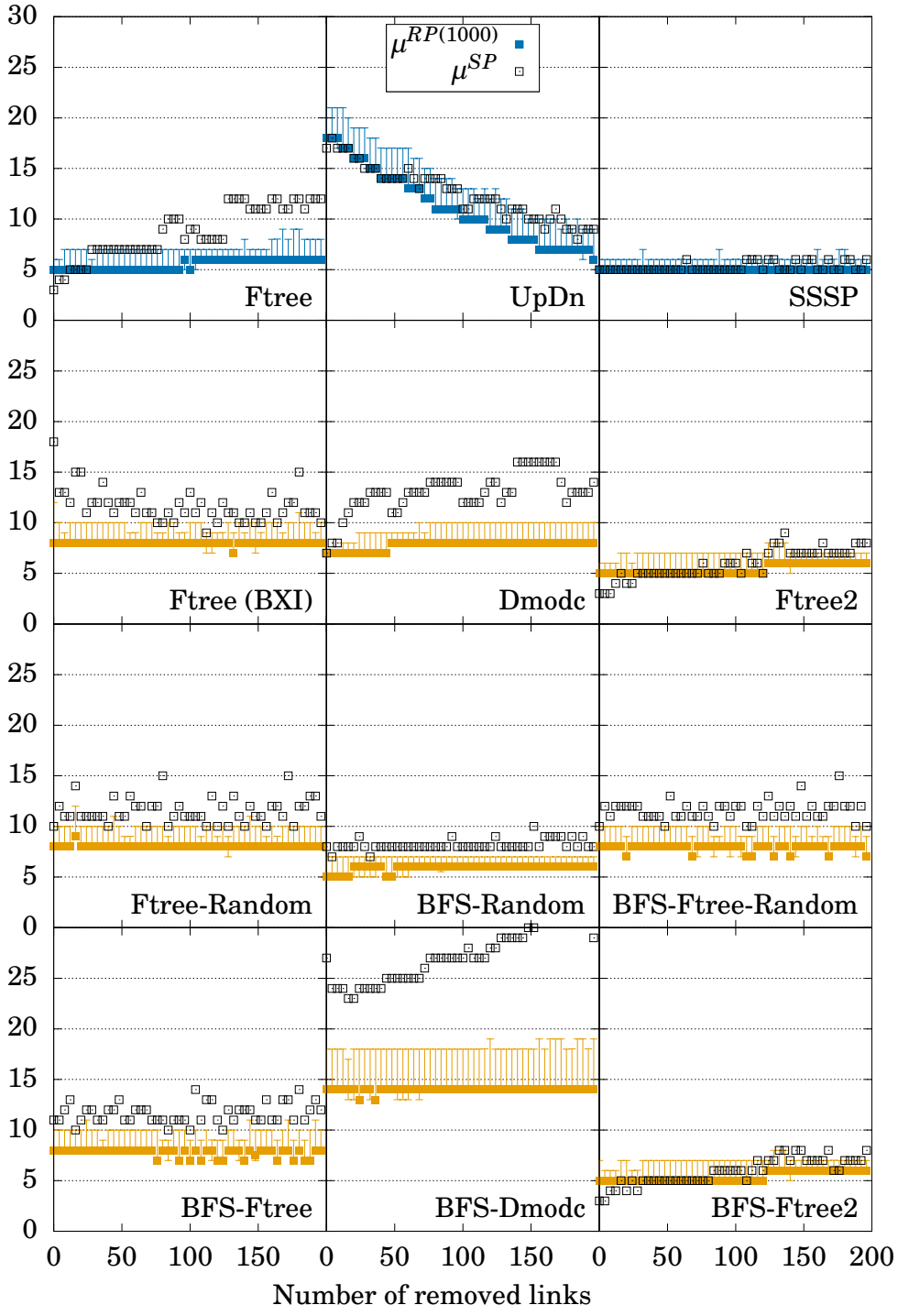


Figure 5.10: Second sequence of progressively randomly link-degraded 1036-node quasi-fat-tree (G_{QFT})

Conclusion

Contributions

The interactions between multiple aspects of the design of supercomputers and their interconnect are key elements to consider when working to improve their real-life performance. Such aspects include network topology, static and adaptive routing, communication patterns, deadlock-freedom, fault resilience, node-type heterogeneity. This multiplicity of potential interactions leads to many potential worthwhile improvements; it also reflects an intricate situation, and the challenge of evaluating the performance of a given system. The progression through each chapter reflects a series of specific problems coming from the industrial context, as well as the improvements we propose to solve them. We begin this progression with a focus on performance evaluation so that subsequent steps can be evaluated quantitatively. Each consideration on interactions to study, and attempt to take explicitly into account in the design of routing algorithms, also brings the cost of added intricacy. The software that was developed alongside this research was designed to be as elegant and fast as possible considering the multiplicity of algorithms and configurable techniques it had to implement. This was possible, first thanks to the fact that the original BXIFM was written from scratch years after such existing software was made available (and extensively studied in recent research). Second, it was rewritten entirely (as BXI AFM) after most research contributions from this thesis had been largely fleshed out. The latter rewrite took into account these results: for example, removing the offline/online distinction was pivotal in focusing on closed-form algorithms like Dmodc.

This work was partly done in an industrial context and research exploration was guided by the technological state of the art, to apply and extend existing theoretical research. Even though this thesis aims to promote and improve on high-level models and algorithm definitions, experience from the accompanying development work highlighted how important low-level details were to effective performance and quality. Throughout the work combining research and industrial implementation, a large part of the effort spent was taken by the iterative design of data structures and algorithms modelling the desired routing techniques. It became quite apparent how closely coupled routing engines are to the rest of a fabric manager and how providing both a high-quality and an efficient routing software requires a good overall understanding of the fabric manager.

Combining existing elements of research regarding quality measurement of static routing functions and a simple congestion model, Chapter 2 presents a framework in which to perform quantitative static estimation of congestion risk for a static routing function in a given network topology for sample com-

munication patterns. Rather than provide direct measurements of realistic network characteristics, this instead provides a tool to compare performance between multiple routing algorithms in a given situation. This framework was then used in the three subsequent chapters to help evaluate efficiently the quality of static routing tables computed by the existing and novel approaches compared fairly for the considered problems.

The first such problem considered (in Chapter 3) was that of heterogeneous supercomputers, affected by avoidable network congestion in realistic worst-case scenarios. After evaluating the problem quantitatively (in terms of congestion risk) for the existing static routing function, an intuitive preprocessing step was presented and evaluated, and ultimately shown to potentially improve the existing routing algorithms, assuming the considered worst-case scenario is actually happening. A traditional dynamic simulation confirmed the results evaluated using the static metrics. This is one glaring case in which real requirements caused the common theoretical model to entirely fail to predict the congestion risk, and where solving this problem at the static routing level can be an efficient approach. However, this approach stresses a common (but potentially undesirable) expectation that communications to different node-types happen at different times.

The second problem considered (in Chapter 4) was the apparent lack of fast routing algorithm for PGFTs, providing high-quality / low-congestion-risk routing even under large-scale equipment failure. A new algorithm was presented to fill this gap, using a combination of a largely sequential network discovery preprocessing phase and a parallel computation phase. This algorithm was also evaluated statically to show how it reacts gracefully to equipment failure while incurring only a fraction of the run time of current routing engines in our implementation. Incidentally, this algorithm is used in production in multiple large supercomputers and was helpful in developing the new fabric manager for Atos in-house interconnects.

Lastly, the scope of the problem was widened to a more generalized network topology in Chapter 5; taking into account both a wider class of realistic networks and fault resilience, while aiming to provide better performance than topology-agnostic techniques. A large selection of both existing and new routing algorithms was presented and fairly compared to show potential improvements applicable to a variety of realistic use-cases even under large-scale equipment failure, while showcasing a varied use of the static evaluation framework.

These proposals are often composable, and care was taken in algorithmic design and programming to implement fast algorithms. These proposals can improve the performance of HPC clusters, but they require some amount of attention as to their applicability and some parameters might affect results according to application. This reflects the trade-offs revealed by these approaches, which must be navigated to take into account the gap between theoretical systems and real supercomputers.

Future research

Up-and-coming supercomputers are made up of nodes with more and more cores. Since each node is connected to the rest of the network through one link, this means that for a given number of cores, both the number of nodes and the network size are going down. Accordingly, switch radix and bandwidth tend to go up with every new generation, to accommodate nonetheless increasing node demands while decreasing network size. With these aspects in mind, it seems that interconnect routing requirements are only decreasing. However, several elements indicate that large-scale interconnects are going to keep expanding in the coming future, and thus require demanding research. First, overall performance demands have been increasing for over half a century, and the largest supercomputers are accordingly increasing in core count every year. This trend seems to be continuing in this same way for the coming future. Second, memory requirements in many distributed applications result in the use of more nodes than computation requirements would justify. Third, the introduction of multi-slot NICs [90] increases the network requirements for a given node count, which might in turn impact routing requirements. For these reasons, continuing research on improving HPC interconnects is still very much warranted.

Focusing on a specific contribution, one of the goals of this thesis was to provide a useful static flow metric from which we deduce probable congestion to quantitatively compare algorithms. Even applied to deterministic routing, this doesn't capture all dynamic behaviour; and this approach goes against the grain of most recent research measuring routing performance. For these reasons, we intend on producing a more thorough analysis of the relationship between this metric and actual congestion depending on fine-grain communication interaction. A corresponding study of the new algorithms (other than Gxmodk) based on simulation rather than only a static congestion metric will also provide more realistic results for some edge cases.

Some of the investigations that have been undertaken in parallel to this thesis focus on optimising adaptive route selection, alongside nonminimal adaptive routing of MegaFly networks. We are working on proposing a configurable multi-step selector with understanding of VCs, per-destination least-recently used output port, and the minimal/non-minimal nature of output ports. They also focused on separation of deterministic, minimal adaptive and nonminimal adaptive traffics into three separate VCs (for each traffic flow where adaptive routing will be available). These investigations will also include detailed simulations as well as attempts at reduction to static analysis, in the vein of this thesis.

Regarding the offline/online distinction which is not required for closed-form algorithms such as Dmodc: this obliviousness helps focus on good quality fast routing, however recovering some knowledge of network history can be useful.

For example, it allows developing strategies to mitigate large scale routing table changes on frequent network updates. This should limit out-of-order delivery of messages for some cases (as a more involved generalisation of the technique described in the last paragraph of Section 4.3). I am participating in the development of an approach merging two parallel routing tables, and this technique will be introduced in a future version of BXI AFM.

Instead of enforcing local distributions of routes (from the bottom switches for sequential algorithms), route selection can be reorganised to enforce distribution of routes at the center of the network (so-called top switches). This can be done explicitly when computing routing functions which have source or destination coalescing points, mapping these onto the set of top switches for example. This global decision-making process can be described as a closed formula, allowing similar parallel designs to those making up Dmodc for performance and repeatability. However, care should be taken to verify whether such a top–down approach might not cause unnecessary congestion to happen in very irregular topologies due to intractable route collisions at the local scale resulting from preference given to global scale distribution. I have developed such a routing engine as a proof of concept during the course of my thesis.

The techniques proposed in this thesis could be combined with existing techniques providing job-placement-aware static routing. For example, introducing job-placement (or some other accurate estimation of communication pattern) as a primary factor in topological NID renumbering should easily make Dmodc adapt well to changing job placements, potentially reducing congestion in cases of job fragmentation or custom communicators.

Translated conclusion

Conclusion in French

Contributions

Lorsqu'on travaille à améliorer les performances réelles des supercalculateurs, une approche clé consiste à étudier les intéractions entre plusieurs aspects de leur design. Parmi ces aspects, on retrouve la topologie du réseau, le routage statique et adaptatif, les schémas de communication, la garantie d'absence de deadlocks, la résilience aux pannes, l'hétérogénéité de types de nœuds. Cette pluralité d'intéractions amène de nombreux axes d'améliorations potentiels ; mais c'est également le reflet d'une situation épineuse, et la source de difficultés à évaluer les performances d'un système donné. Le cheminement des chapitres suit une série de problématiques spécifiques émanant du contexte industriel, ainsi que des améliorations proposées pour y répondre. Ce cheminement débute avec un effort particulier sur l'évaluation de performance, afin de pouvoir évaluer quantitativement les étapes suivantes. À chaque fois que l'on choisit d'étudier une certaine intéraction, et de la prendre en compte dans la construction des algorithmes de routage, s'ajoute le coût du couplage. J'ai cherché à développer les codes qui accompagnent cette thèse de manière aussi élégante et rapide que possible, lorsqu'on considère la diversité des algorithmes et des techniques configurables qu'ils accomplissent. Le premier élément qui a aidé à aller dans cette direction est le fait que le logiciel BXIFM initial a été développé à zéro, alors que des systèmes concurant existaient (et avaient été étudié publiquement) déja depuis des années. Par la suite, il a été entièrement réécrit (sous la forme de BXI AFM) après que la majeure partie des contributions de cette thèse avaient été entreprises. Cette réécriture a impacté la suite de la thèse : par exemple, le retrait de la distinction offline/online a permi de se concentrer sur des algorithmes à forme close tels que Dmodc.

Ces travaux ont partiellement eu lieu dans un contexte industriel, et la recherche a suivi l'état de l'art technologique, afin d'appliquer la recherche théorique existante et de l'approfondir. Même si le but de cette thèse est de promouvoir et améliorer des modèles à haut niveau et la définition des algorithmes, le travail de développement logiciel a révélé combien les détails d'implémentation ont une importance sur les performances effectives et la qualité résultante. Au cours de l'effort combinant recherche et implémentation industrielle, une large partie du travail a été dédiée au design itératif des structures de données avec les algorithmes accomplissant les techniques de routage désirées. Il apparût comme une évidence à quel point les fonctions calculant les tables de routage sont fortement couplées au reste du fabric manager, et que le calcul de routes de bonne qualité de manière efficiente

requiert une connaissance approfondie de ce dernier.

Combinant la recherche existante au sujet de la mesure de qualité des fonctions de routage statiques avec un modèle simple de congestion, le Chapitre 2 présente une méthode permettant d'estimer le risque statique de congestion de manière quantitative pour une fonction de routage statique dans une topologie et pour un schéma de communication donnés. Plutôt que de mesurer des caractéristiques réseau réalistes, cet outil permet de comparer les performances de plusieurs algorithmes de routage dans une situation donnée. Cette méthode est ensuite utilisée dans les trois chapitres suivants pour aider à évaluer efficacement la qualité des tables de routage statiques calculées par les approches existantes et nouvelles, en les comparant équitablement dans le contexte des problématiques étudiées.

La première de ces problématiques étudiée (dans le Chapitre 3) est celle des supercalculateurs hétérogènes, pouvant être impactés par de la congestion réseau évitable dans des scénarios de pire cas, mais réalistes. Après une étude quantitative du problème (en terme de risque de congestion) pour une fonction de routage statique existante, un étape de précalcul intuitive a été présentée, étudiée, et enfin montrée comme pouvant améliorer les algorithmes de routage existant, si tant est que le scénario de pire cas ait vraiment lieu. Une simulation dynamique traditionnelle a confirmé les résultats obtenus à l'aide des métriques statiques. Ce cas met en exergue comment des conditions réelles peuvent mettre en échec complet la capacité au modèle théorique de prédire le risque de congestion, et comment il peut être viable de résoudre ce problème au niveau du routage statique. En revanche, cette approche souligne également comment l'organisation courante des communications en différentes phases temporelles peut être indésirable.

La seconde problématique considérée (dans le Chapitre 4) est celle du manque apparent d'algorithmes de routage rapides pour les PGFT, fournissant un routage de haute qualité (à faible risque de congestion) même en cas de nombreuses pannes. Pour combler ce manque, un nouvel algorithme est présenté, combinant une phase partiellement séquentielle de précalcul pour la découverte du réseau et une phase parallèle de calcul des routes. Cet algorithme a aussi été évalué avec la métrique statique, montrant comme il reste faiblement impacté par les pannes matérielles, tout en ne coûtant (dans notre implémentation) qu'une fraction du temps d'exécution des algorithmes de routage courants. En outre, cet algorithme est utilisée en production dans plusieurs grands supercalculteurs et a aidé au développement du nouveau fabric manager pour l'interconnect produit par Atos.

Enfin, la problématique a été élargie à une classe de topologie plus généralisée dans le Chapitre 5 ; ceci prenant en compte une classe plus large de réseaux réalistes et la tolérance aux pannes, tout en conservant une meilleure performance que les techniques complètement génériques à n'importe quelle topologie. Une sélection étendue d'algorithmes de routage aussi bien exis-

tants que nouveaux est présentée et comparée équitablement pour montrer les améliorations potentielles applicables à une variété de cas d'usage réalistes même en cas de nombreuses pannes matérielles, tout en affichant un panel varié d'utilisations de la méthode d'évaluation statique.

Ces propositions sont pour la plupart modulaires, et un effort particulier a été entrepris dans l'écriture algorithmique et la programmation pour l'implémentation de routage rapide. Ces propositions peuvent améliorer les performances de clusters HPC, mais elles nécessitent une certaine attention au sujet de leur applicabilité et paramétrisation pouvant affecter les résultats selon les applications. Ceci reflète les intéractions subtiles motivant ces approches, pour lesquelles il faut chercher un équilibre si on veut prendre en compte l'écart entre la théorie et la réalité des supercalculateurs.

Travaux futurs

Les supercalulateurs à venir sont composés de nœuds contenant de plus en plus de cœurs. Vu que chaque nœud est connecté au reste du réseau via un seul lien, ceci signifie que pour un nombre de cœurs donné, le nombre de nœuds baisse (et avec lui la taille totale du réseau). Ce faisant, le nombre de ports et la bande passante des switches tend à augmenter avec les générations matérielles, pour réduire la taille totale du réseau tout en accomodant une demande nonobstant croissante en nœuds. Avec ces éléments à l'esprit, il peut sembler que les performances requises du routage tenderont à décroître. Pourtant, plusieurs éléments indiquent que les interconnect vont continuer à se développer dans les années à venir, et ainsi continuer à motiver des recherches. Premièrement, les besoins en performance ont continuellement augmentés depuis plus d'un demi-siècle, et les supercalculateurs les plus puissants ont de fait augmenté en nombre de cœurs d'année en année. Cette tendance semble continuer ainsi pour le futur à venir. Deuxièmement, les besoins en mémoire de nombreuses applications distribuées poussent à utiliser encore plus de nœuds que les simples besoins en calcul ne justifieraient. Troisièmement, l'introduction de NIC à plusieurs slots [90] augmente les dimensions du réseau pour un nombre de nœuds donné, impactant alors la charge du routage. Pour ces raisons, il demeure utile de continuer la recherche pour améliorer les interconnects HPC.

Au niveau des contributions spécifiques, un des buts de cette thèse était de fournir une métrique statique de flux de laquelle on puisse déduire les risques de congestion, afin de pouvoir comparer les algorithmes de manière quantitative. Même en se restreignant au routage déterministe, ceci ne permet pas de décrire les comportements dynamiques potentiels ; de plus, cette approche va à contre-courant de la majorité des études faisant de la mesure de performance de routage. Pour ces raisons, nous voulons produire une analyse plus approfondie de cette métrique par rapport à la congestion réellement induite, prenant en compte les intéractions de bas-niveau entre communications. En ce

sens, une étude des nouveaux algorithmes (à part Gxmodk) qui reposerait sur de la simulation plutôt que la métrique de congestion statique apporterait du détail dans les résultats.

Certaines des études réalisées en parallèle de cette thèse ont porté sur l'optimisation de la sélection de routes adaptatives, ainsi que le routage adaptatif non-minimal sur les réseaux MegaFly. Nous travaillons à fournir un arbitre configurable à plusieurs niveaux, qui aurait notion des VC, du port de sortie utilisé le plus récemment (pour chaque destination), et de la nature minimale ou non de chaque port de sortie. Ces études visent à séparer les traffics déterministes, adaptatif minimal, et non-minimal, en trois VC distincts (pour chaque flux qui peut être routé adaptativement).

Le fait de ne pas séparer le routage en composantes offline/online (comme dans Dmodc) permet de se concentrer à fournir dès que possible des routes de bonne qualité, mais il peut être bénéfique de réintroduire un certain degré d'historique de l'évolution du réseau. Par exemple, cela permet de développer des stratégies pour mitiger le volume des modifications des tables de routage lors de mises-à-jour fréquentes de l'état du réseau. Une telle mitigation réduit le volume de messages arrivant dans le désordre dans certains cas (et forme une généralisation plus ardue de la technique décrite dans le dernier paragraphe de la section 4.3). J'ai participé au développement d'une approche qui combine deux tables de routage calculées en parallèle, qui sera livrée dans une version future de BXI AFM.

Au lieu de répartir les routes en considérant des choix à petite échelle (depuis les switches du bas pour les algorithmes séquentiels), la sélection des routes peut être réorganisée afin de garantir la bonne distribution des routes au milieu du réseau (au niveau des switches de plus haut niveau). Ceci peut être réalisé de manière explicite pour les fonctions de routage qui agglutinent par source ou destination, en distribuant les points agglutinants sur les top switches. Ce processus de prise de décision à un niveau global peut être décrit par une formule close, permettant une implémentation parallélisée similaire à celle de Dmodc, performante et déterministe. En revanche, il conviendra de vérifier si une telle approche de haut en bas ne causerait pas de la congestion évitable dans des topologies très irrégulières, suite à des collisions de routes à une échelle locale qu'on ne pourrait séparer suivant la préférence donnée à la distribution à plus haute échelle. J'ai développé le prototype d'un tel algorithme de routage durant ma thèse.

Les techniques proposées dans cette thèse pourraient être combinées avec des techniques existantes pour calculer le routage déterministe en fonction du placement des jobs. Par exemple, prendre en compte le placement des jobs (ou toute autre élément apportant une estimation précise des schémas de communication) en tant que facteur primordial dans la renumérotation des « topological NID » mènerait Dmodc naturellement à s'adapter aux changement de placement des jobs, réduisant ainsi le risque de congestion lors de la

fragmentation des jobs, ou de l'utilisation de communicateurs mal optimisés.

Conclusion in Spanish

Conclusiones

Las interacciones entre los múltiples aspectos del diseño de los supercomputadores y sus redes de interconexión son elementos clave a considerar cuando se trabaja para mejorar sus prestaciones en sistemas reales. Dichos aspectos incluyen la topología de la red, el encaminamiento estático o adaptativo, los patrones de comunicación, las garantías de ausencia de interbloqueo, la tolerancia a fallos, la heterogeneidad de los nodos finales, etc. Esta diversidad de potenciales interacciones posibilita muchas mejoras potenciales que merece la pena explorar; también refleja una situación compleja, y hace que evaluar con precisión el rendimiento de un sistema dado constituya un reto. Cada capítulo de esta tesis refleja una serie de problemas específicos provenientes del contexto industrial, así como las mejoras que proponemos para resolverlos. La tesis comienza centrándose en la evaluación de prestaciones para que las propuestas posteriores puedan evaluarse cuantitativamente. Cada consideración sobre las interacciones a estudiar, y cada intento de tenerlas en cuenta explícitamente en el diseño de algoritmos de encaminamiento, también conlleva el coste de una complejidad añadida. El software de control de la red que se desarrolló en paralelo a esta investigación fue diseñado para ser lo más elegante y rápido posible, considerando la multiplicidad de algoritmos y técnicas configurables implementadas. Esto fue posible, en primer lugar, gracias al hecho de que el software de control original (BXIFM) se creó desde cero años después de que software similar estuviese disponible (y fuese estudiado exhaustivamente). En segundo lugar, el software de control se reescribió por completo (con el nombre de BXI AFM) después de que la mayoría de las contribuciones de esta tesis se hubiesen desarrollado. Efectivamente, esta última reescritura tuvo en cuenta los resultados de esta tesis: por ejemplo, eliminar la distinción entre offline y online fue fundamental para centrarse en algoritmos de forma cerrada como Dmodc.

Los trabajos de esta tesis se desarrollaron parcialmente en un entorno industrial, y en general la investigación estuvo orientada hacia la aplicación a tecnologías punta de la investigación teórica existente, extendiendo la misma. A pesar de que esta tesis intenta impulsar y mejorar los modelos de alto nivel y las definiciones de algoritmos, durante su desarrollo se hizo evidente cuán importantes son en la práctica los detalles de bajo nivel para el rendimiento y la calidad. Al combinar a lo largo del trabajo la investigación y la implementación industrial, una gran parte del esfuerzo invertido se dedicó al diseño iterativo de estructuras de datos y algoritmos que modelan las técnicas de encaminamiento deseadas. Se hizo bastante evidente cuán estrechamente unidos están los encaminamientos con el resto de aspectos configurables del sistema, y cómo

el proporcionar un software de encaminamiento eficiente y de alta calidad requiere una buena comprensión general de todos estos aspectos.

En el Capítulo 2 de esta tesis se combinan elementos de investigación existentes sobre la medición de la calidad de las funciones de encaminamiento estático con un modelo simple de congestión, para presentar un marco en el que realizar una estimación estática cuantitativa del riesgo de congestión, para una función de encaminamiento estático en una topología de red dada, y para muestras de patrones de comunicación. En lugar de proporcionar mediciones directas de características de red realistas, este enfoque proporciona una herramienta para comparar el rendimiento de varios algoritmos de encaminamiento en una situación determinada. Este marco se usó luego en los tres capítulos siguientes para evaluar la calidad de las tablas de encaminamiento estático calculadas tanto por las técnicas existentes como por las nuevas propuestas, comparando unas y otras de forma justa en los escenarios considerados.

El primero de tales escenarios (en el Capítulo 3) fue el del problema de los supercomputadores heterogéneos afectados por congestión evitable, en casos extremos pero realistas. Una vez evaluado este problema cuantitativamente (en términos de riesgo de congestión) para la función de encaminamiento estático existente, se presentó y evaluó una fase de preprocesamiento intuitiva que, finalmente, se demostró que mejora potencialmente los algoritmos de encaminamiento existentes, asumiendo que el peor de los casos considerado está sucediendo realmente. Además, una simulación dinámica tradicional confirmó los resultados obtenidos utilizando las métricas estáticas. Este es un caso evidente en el que los requisitos reales hicieron que el modelo teórico común fallara por completo en la predicción del riesgo de congestión, y para el que resolver este problema a nivel de encaminamiento estático puede ser un enfoque eficiente. Sin embargo, en este enfoque subyace una expectativa común (pero potencialmente indeseable) de que las comunicaciones con tipos diferentes de nodos ocurren en diferentes momentos.

El segundo problema considerado (en el Capítulo 4) fue el de la aparente falta de un algoritmo de encaminamiento rápido para las topologías de tipo Parallel Generalized Fat-Tree (PGFT), que proporcionase un encaminamiento de alta calidad y bajo riesgo de congestión, incluso en caso de fallos a gran escala en la red. Para llenar este vacío se presentó un nuevo algoritmo, que utiliza una combinación de una fase de preprocesamiento para descubrir la red (en gran parte secuencial) y una fase de cálculo paralelo. Este algoritmo también se evaluó estáticamente, demostrándose que reacciona eficientemente ante fallos de los componentes de la red, a costa de aumentar sólo una fracción el tiempo de ejecución de las actuales implementaciones software del encaminamiento. Por cierto, este algoritmo se usa realmente en varios grandes supercomputadores, y fue útil en el desarrollo del nuevo software de control de las interconexiones internas de Atos.

Por último, en el Capítulo 5 se extendió el escenario a una topología de red más generalizada, teniendo en cuenta tanto una clase más amplia de redes realistas como la tolerancia a fallos, e intentando siempre ofrecer mejores prestaciones que las técnicas agnósticas respecto a la topología. Se presentó una gran selección de algoritmos de encaminamiento nuevos y existentes, y se aplicó el marco de evaluación estático para compararlos de manera justa y finalmente detectar mejoras potenciales aplicables a una variedad de casos realistas, incluyendo casos de fallos a gran escala en la red.

Varias de estas propuestas son combinables, y en su implementación software se puso especial énfasis en el diseño y la programación algorítmica, de cara a obtener algoritmos rápidos. Estas propuestas pueden mejorar el rendimiento de los clústeres de computación paralela de altas prestaciones (High-Performance Computing, HPC), pero requieren cierto cuidado respecto a su aplicabilidad, y el valor de algunos parámetros puede afectar a los resultados dependiendo de la aplicación. Esto refleja las ventajas y desventajas de estos enfoques, que deben para tener en cuenta la brecha entre los sistemas teóricos y los supercomputadores reales.

Trabajo futuro

Los supercomputadores modernos están formados por nodos finales que incluyen cada vez más y más núcleos. Puesto que cada nodo está conectado al resto de la red a través de un enlace, esto significa que, para una cantidad total concreta de núcleos, tanto la cantidad de nodos finales como el tamaño de la red están disminuyendo. Al mismo tiempo, el número de puertos (radix) y el ancho de banda del conmutador tienden a aumentar con cada nueva generación, para adaptarse a las crecientes demandas de los nodos, mientras se reduce el tamaño de la red. Teniendo estos aspectos en cuenta, parecería que los requisitos que debe satisfacer el encaminamiento de la red de interconexión estarían disminuyendo. Sin embargo, varias circunstancias indican que las redes de interconexión a gran escala seguirán expandiéndose en el futuro próximo y, por lo tanto, requerirán una investigación profunda. En primer lugar, la demanda general de prestaciones ha aumentando constantemente durante más de medio siglo y, en consecuencia, los supercomputadores más grandes están aumentando en número de núcleos cada año. Esta tendencia tiene visos de continuar en el futuro próximo. En segundo lugar, los requisitos de memoria en muchas aplicaciones distribuidas dan como resultado el uso de más nodos de los que cubrirían las necesidades de computación. En tercer lugar, la introducción de interfaces de red (network interface cards, NICs) de múltiples ranuras [90] aumenta los requisitos que debe satisfacer la red para un número total de nodos determinado, lo que a su vez podría afectar a los requisitos que debe satisfacer el encaminamiento. Por estas razones, continuar investigando para mejorar las redes de altas prestaciones par sistemas HPC todavía está plenamente justificado.

Centrándonos en una contribución específica, uno de los objetivos de esta tesis es obtener una métrica de flujo estática que sea útil para deducir la probabilidad de congestión, de cara a comparar algoritmos cuantitativamente. Sin embargo, incluso aplicado al encaminamiento determinista, este enfoque no captura todo el comportamiento dinámico; además, no va en consonancia con la granularidad de investigaciones recientes que miden el rendimiento del encaminamiento. Por estas razones, nuestra intención es realizar un análisis más completo de la relación entre esta métrica y la congestión real, en función de las interacciones de comunicación de grano fino. En este sentido, un estudio de los nuevos algoritmos (aparte de Gxmodk) basado en simulación en lugar de sólo una métrica de congestión estática, proporcionaría resultados más realistas para algunos casos extremos.

Algunas de las investigaciones que se han llevado a cabo en paralelo a esta tesis se centran en optimizar la selección de rutas adaptativas, junto con el encaminamiento adaptativo no mínimo de las topologías MegaFly. Estamos trabajando actualmente para proponer un selector de rutas configurable de varios pasos que considere los canales virtuales (Virtual Channels, VCs), el puerto de salida menos utilizado recientemente para alcanzar cada destino, y la naturaleza mínima/no mínima de la ruta a la que conducen los puertos de salida. Las investigaciones también se han centrado en la separación de los tráficos determinista, adaptativo mínimo y adaptativo no mínimo, en tres VCs separados (para cada flujo de tráfico donde el encaminamiento adaptativo esté disponible). Estas investigaciones también incluirán simulaciones detalladas, así como intentos de reducción al análisis estático, en la línea de esta tesis.

Por otra parte, no tener en cuenta la distinción offline/online en los algoritmos de forma cerrada (como Dmodc) ayuda a centrarse en el encaminamiento rápido de buena calidad, pero recuperar algo de conocimiento del historial de la red también puede ser útil. Por ejemplo, esto permite desarrollar estrategias para mitigar cambios a gran escala en la tabla de encaminamiento debido a actualizaciones frecuentes de la red. Esto debería limitar la entrega fuera de orden de mensajes en algunos casos (como una generalización de la técnica descrita en el último párrafo de la Sección 4.3). Actualmente estamos participando en el desarrollo de un enfoque para fusionar dos tablas de encaminamiento paralelas, lo que se incluirá en una versión futura de BXI AFM.

Además, en lugar de usar distribuciones locales de rutas (desde los conmutadores inferiores para los algoritmos secuenciales), la selección de rutas se puede reorganizar para imponer la distribución de rutas en el centro de la red (o sea, desde los llamados conmutadores superiores). Esto se puede hacer explícitamente cuando se calculan funciones de encaminamiento que tienen puntos convergentes de origen o destino, mapeándolos en el conjunto de conmutadores superiores, por ejemplo. Este proceso global de toma de decisiones se puede describir como una fórmula cerrada, que permite diseños paralelos simi-

lares a los que componen Dmodc en cuanto a rendimiento y repetibilidad. Sin embargo, se debe tener cuidado para verificar si tal enfoque desde arriba hacia abajo podría causar una congestión innecesaria en topologías muy irregulares, debido a colisiones de rutas difíciles o imposibles de evitar a escala local, que resultan de la preferencia otorgada a la distribución a escala global. Durante el curso de esta tesis, se ha desarrollado una implementación de este tipo de encaminamiento, como prueba de concepto.

Las técnicas propuestas en esta tesis podrían combinarse con técnicas existentes que proporcionen encaminamiento estático en función del mapeo de trabajos a nodos. Por ejemplo, la introducción del mapeo de trabajos a nodos (o alguna otra estimación precisa del patrón de comunicación) como un factor principal en la renumeración de los identificadores de los nodos (NIDs) de la topología debería hacer que Dmodc se adapte bien a los cambios del mapeo de trabajos, lo que podría reducir la congestión en casos de fragmentación del trabajo o de nodos comunicándose fuera del patrón estimado.

Annex

1 . Routing Leiserson fat-trees

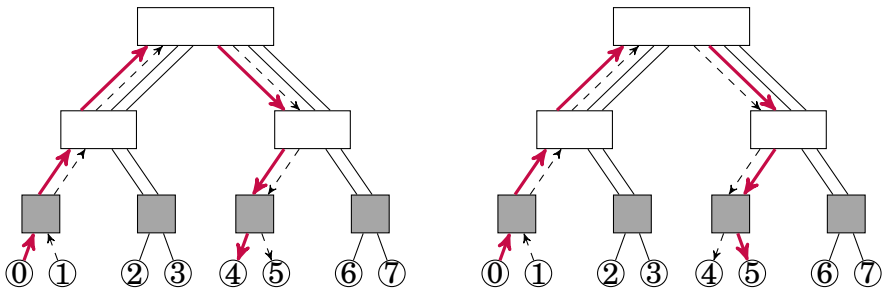
Shortest switch-path selection is simply going up until reaching the nearest common ancestor (NCA) of the source and destination, and then going down towards the destination. Leiserson implements this using cheap binary operations on source and destination node indices. The routing algorithms provided to schedule message sets into delivery frames through the available channels cannot be translated to modern cluster interconnects. However, if considering the switches as crossbar switches, it is relatively straightforward to design a "perfect" static routing algorithm, with each source being assigned a unique uppath to the top and conversely each destination a unique downpath from the top. This is easily implemented using input port index as shown in Algorithm 13.

Algorithm 13: Non-blocking static routing for Leiserson fat-trees.

Data: switch s, input port i (in s) to destination d
Result: output port p (in s)

if *d is down-accessible from s* **then**
    D is the ordered set of down ports of s that lead towards b
    $p \leftarrow D[d \bmod \#D]$
else
    D is the ordered set of down ports of s
    i' is the index of i in D
    U is the ordered set of up ports of s
    $p \leftarrow U[i']$

The resulting static routes illustrate how under exclusive point-to-point communications, new paths are always available regardless of existing traffic (as shown in Figure 11). This is equivalent to the non-blocking nature of Clos networks with $m \geq 2n - 1$, even though it does not reflect other communication patterns that are commonly found in HPC applications.



(a) $0 \to 4$ and $1 \to 5$ (b) $0 \to 5$ and $1 \to 4$

Figure 11: Distinct paths under non-blocking static routing in two different point-to-point communication patterns.

2 . Resilient statistical ranking method

As an alternative statistical method to determine top switches (to that presented in Section 1.4.5), I propose a similar criterion, based on the general hierarchical nature of fat-tree-like topologies, defined in Algorithm 14.

Algorithm 14: Resilient statistical top switch criterion

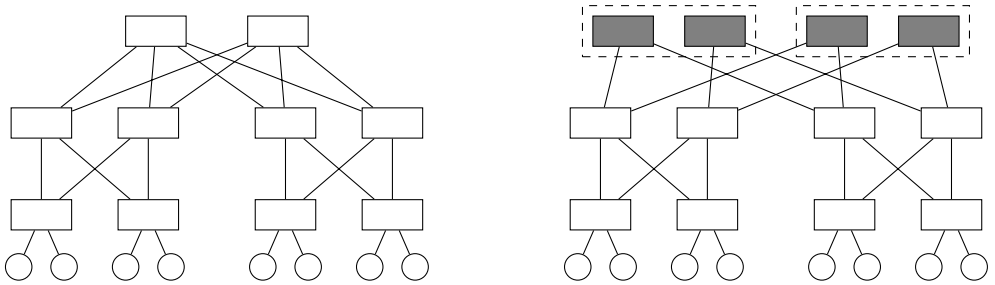
1 Compute each switch's distance from every compute node
2 Find each switch's smallest most frequent distance from nodes ($\bar{D}_s$)
3 Select all switches with $\bar{D}_s = \min_{\forall s'}(\bar{D}_{s'})$ as top switches

The intuition behind this criterion is that non-top switches are at a greater distance from most nodes than top switches are. Note that this is based on real shortest path distance, and not up–down shortest path distance, since this step must come before determining levels or direction. As a result, the method (like OpenSM's) has a complexity of $\mathcal{O}(\#E^3)$ if implemented using repeated Dijkstra.

One advantage of the method is that it guarantees that at least one switch will be selected as top switch.

3 . Routing vPGFTs

Some IFTs in which there are multiple shortest paths from top switches to leaf switches can be converted into PGFTs by splitting up each switch (of one or several levels) into two or more switches. Such topologies can be referred to as vPGFTs, and an example is provided in Figure 12. A valid method of routing vPGFTs is to compute routing tables for the corresponding PGFT, using either Dmodk or Ftree, and joining the tables of each part of each switch into the physical tables. This can be done without conflict for source-based or input port-based deterministic routing because each set of routes stays within its subset of ports. (In such a case, this approach has the potential



(a) vPGFT

(b) Corresponding PGFT with virtual switches in grey.

Figure 12: Example vPGFT, with corresponding PGFT. Note that a $PGFT(3;2,2,2;1,2,1;1,1,2)$, an RLFT, could have been used instead of the vPGFT, using the same equipment.

drawback that available resources are discarded since no route is allowed to go across virtual switches.) Otherwise, there might be deterministic routes to the same destination computed in several virtual switches; these conflicts must be resolved to determine which route must be kept in the vPGFT. If any secondary deterministic route is guaranteed correct, any choice of conflict resolution is correct, albeit potentially nonoptimal. Since the vPGFT is a BMIN, and merged routes are up–down, the resulting routing function is deadlock-free.

Bibliography

[1] IEEE 1003.1c-1995. *Threads extensions*. Standard. Institute of Electrical and Electronics Engineers, June 1995.

[2] ISO 9899:1999. *Programming languages — C*. Standard. International Organization for Standardization, Dec. 1999.

[3] Ahmad Chadi Aljundi, Jean-Luc Dekeyser, M Tahar Kechadi, and Isaac D Scherson. "A study of an evaluation methodology for unbuffered multistage interconnection networks". In: *Proceedings International Parallel and Distributed Processing Symposium*. IEEE. 2003, 8–pp.

[4] Brian W. Barrett et al. *The Portals 4.3 Network Programming Interface*. Tech. rep. SAND2022-8810. Albuquerque, New Mexico: Sandia National Laboratories, 2022. URL: https://www.sandia.gov/portals/.

[5] Maciej Besta, Marcel Schneider, Marek Konieczny, Karolina Cynk, Erik Henriksson, Salvatore Girolamo, Ankit Singla, and Torsten Hoefler. "FatPaths: Routing in Supercomputers and Data Centers when Shortest Paths Fall Short". In: *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. IEEE. 2020, pp. 365–382.

[6] Bartosz Bogdanski, Frank Olaf Sem-Jacobsen, Sven-Arne Reinemo, Tor Skeie, Line Holen, and Lars Paul Huse. "Achieving predictable high performance in imbalanced fat trees". In: *16th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE. 2010, pp. 381–388.

[7] Bartosz Bogdanski, Bjørn Dag Johnsen, Sven-Arne Reinemo, and Frank Olaf Sem-Jacobsen. "Discovery and routing of degraded fat-trees". In: *2012 13th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*. IEEE. 2012, pp. 697–702.

[8] Bartosz Bogdanski, 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". In: *ACM Transactions on Architecture and Code Optimization (TACO)* 8.4 (2012), p. 55.

[9] Julian Borrill, Leonid Oliker, John Shalf, and Hongzhang Shan. "Investigation of leading HPC I/O performance using a scientific-application derived benchmark". In: *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*. 2007, pp. 1–12.

[10] Dong Chen, Philip Heidelberger, Craig Stunkel, Yutaka Sugawara, Cyriel Minkenberg, Bogdan Prisacari, and German Rodriguez. “An evaluation of network architectures for next generation supercomputers”. In: *2016 7th International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*. IEEE. 2016, pp. 11–21.

[11] Charles Clos. “A study of non-blocking switching networks”. In: *Bell System Technical Journal* 32.2 (1953), pp. 406–424.

[12] Tiffany Connors, Taylor Groves, Tony Quan, and Scott Hemmert. “Simulation Framework for Studying Optical Cable Failures in Dragonfly Topologies”. In: *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE. 2019, pp. 859–864.

[13] Cray. *Slingshot: the Interconnect for the Exascale Era (White Paper)*. 2019.

[14] William J Dally and Charles L Seitz. *Deadlock-Free Message Routing in Multiprocessor Interconnection Networks, Dept. of Computer Science, California Institute of Technology*. Tech. rep. Technical Report 5206: TR: 86, 1986.

[15] William J. Dally and Hiromichi Aoki. “Deadlock-free adaptive routing in multicomputer networks using virtual channels”. In: *IEEE transactions on Parallel and Distributed Systems* 4.4 (1993), pp. 466–475.

[16] William James Dally. “Performance Analysis of k-ary n-cube Interconnection Networks”. In: *IEEE transactions on Computers* 39.6 (1990), pp. 775–785.

[17] William James Dally and Brian Patrick Towles. *Principles and practices of interconnection networks*. Elsevier, 2004.

[18] Daniele De Sensi, Salvatore Di Girolamo, Kim H McMahon, Duncan Roweth, and Torsten Hoefler. “An In-Depth Analysis of the Slingshot Interconnect”. In: *arXiv preprint arXiv:2008.08886* (2020).

[19] Jens Domke, Torsten Hoefler, and Wolfgang E Nagel. “Deadlock-free oblivious routing for arbitrary topologies”. In: *International Parallel & Distributed Processing Symposium*. IEEE. 2011, pp. 616–627.

[20] Jens Domke, Torsten Hoefler, and Satoshi Matsuoka. “Fail-in-place network design: interaction between topology, routing algorithm and failures”. In: *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. IEEE. 2014, pp. 597–608.

[21] Jens Domke, Torsten Hoefler, and Satoshi Matsuoka. “Routing on the dependency graph: A new approach to deadlock-free high-performance routing”. In: *25th International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*. ACM. 2016, pp. 3–14.

[22] Jens Domke and Torsten Hoefler. “Scheduling-aware routing for supercomputers”. In: *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. IEEE. 2016, p. 13.

[23] Jack Dongarra and Piotr Luszczek. *HPCG list*. www.top500.org/lists/hpcg. 2017–2020.

[24] José Duato, Sudhakar Yalamanchili, and Lionel Ni. *Interconnection networks*. Morgan Kaufmann, 1997.

[25] José Duato, Olav Lysne, Ruoming Pang, and Timothy Mark Pinkston. “A theory for deadlock-free dynamic network reconfiguration”. In: *IEEE Transactions on Parallel and Distributed Systems* 16.5 (2005), pp. 412–427.

[26] José Duato, Antonio J Pena, Federico Silla, Rafael Mayo, and Enrique S Quintana-Ortí. “rCUDA: Reducing the number of GPU-based accelerators in high performance clusters”. In: *2010 International Conference on High Performance Computing & Simulation*. IEEE. 2010, pp. 224–231.

[27] Peyman Faizian, Md Atiqul Mollah, Md Shafayat Rahman, Xin Yuan, Scott Pakin, and Mike Lang. “Throughput models of interconnection networks: the good, the bad, and the ugly”. In: *25th Annual Symposium on High-Performance Interconnects (HOTI)*. IEEE. 2017, pp. 33–40.

[28] Mario Flajslik, Eric Borch, and Mike A Parker. “Megafly: A topology for exascale systems”. In: *International Conference on High Performance Computing*. Springer. 2018, pp. 289–310.

[29] Jose Flich, Manuel P Malumbres, Pedro Lopez, and Jose Duato. “Improving routing performance in Myrinet networks”. In: *14th International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE. 2000, pp. 27–32.

[30] Sally Floyd. “TCP and explicit congestion notification”. In: *ACM SIGCOMM Computer Communication Review* 24.5 (1994), pp. 8–23.

[31] Yixiao Gao, Yuchen Yang, Tian Chen, Jiaqi Zheng, Bing Mao, and Guihai Chen. “DCQCN+: Taming large-scale incast congestion in RDMA over ethernet networks”. In: *2018 IEEE 26th International Conference on Network Protocols (ICNP)*. IEEE. 2018, pp. 110–120.

[32] Marina García, Enrique Vallejo, Ramón Beivide, Miguel Odriozola, and Mateo Valero. “Efficient routing mechanisms for dragonfly networks”. In: *2013 42nd International Conference on Parallel Processing*. IEEE. 2013, pp. 582–592.

[33] Pedro Javier García, Francisco J Quiles, Jose Flich, Jose Duato, Ian Johnson, and Finbar Naven. “Efficient, scalable congestion management for interconnection networks”. In: *IEEE Micro* 26.5 (2006), pp. 52–66.

[34] Patrick Geoffray and Torsten Hoefler. “Adaptive routing strategies for modern high performance networks”. In: *16th Annual Symposium on High Performance Interconnects (HOTI)*. IEEE. 2008, pp. 165–172.

[35] Christopher J Glass and Lionel M Ni. “The turn model for adaptive routing”. In: *ACM SIGARCH Computer Architecture News* 20.2 (1992), pp. 278–287.

[36] Crispín Gómez, Francisco Gilabert, María Engracia Gómez, Pedro López, and José Duato. “Deterministic versus adaptive routing in fat-trees”. In: *International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE. 2007, pp. 1–8.

[37] Ronald I Greenberg and Charles E Leiserson. “Randomized routing on fat-tress”. In: *26th annual symposium on Foundations of Computer Science (SFCS 1985)*. IEEE. 1985, pp. 241–249.

[38] Wei Lin Guay, Bartosz Bogdanski, Sven-Arne Reinemo, Olav Lysne, and Tor Skeie. “vFtree-A fat-tree routing algorithm using virtual lanes to alleviate congestion”. In: *International Parallel & Distributed Processing Symposium (IPDPS)*. IEEE. 2011, pp. 197–208.

[39] Marie-Claude Heydemann, Jean Claude Meyer, and Dominique Sotteau. “On forwarding indices of networks”. In: *Discrete Applied Mathematics* 23.2 (1989), pp. 103–123.

[40] Torsten Hoefler, Timo Schneider, and Andrew Lumsdaine. “Multistage Switches are not Crossbars: Effects of Static Routing in High-Performance Networks”. In: *International Conference on Cluster Computing (CLUSTER)*. IEEE. 2008, pp. 116–125.

[41] Torsten Hoefler, Timo Schneider, and Andrew Lumsdaine. “Optimized routing for large-scale InfiniBand networks”. In: *17th Annual Symposium on High Performance Interconnects (HOTI)*. IEEE. 2009, pp. 103–111.

[42] IEEE. *802.1Qau – Congestion Notification*. 1.ieee802.org/dcb/802-1qau/. retrieved 2020.

[43] Nikhil Jain, Abhinav Bhatele, Xiang Ni, Nicholas J Wright, and Laxmikant V Kale. “Maximizing throughput on a dragonfly network”. In: *SC’14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE. 2014, pp. 336–347.

[44] Nan Jiang, John Kim, and William J Dally. “Indirect adaptive routing on large scale interconnection networks”. In: *ACM SIGARCH Computer Architecture News* 37.3 (2009), pp. 220–231.

[45] Gregory Johnson, Darren J Kerbyson, and Michael Lang. “Application specific optimization of infiniband networks”. In: *PAL Roadrunner, LAUR 06* 7234 (2006).

[46] Michael Jurczyk and Thomas Schwederski. “Phenomenon of higher order head-of-line blocking in multistage interconnection networks under nonuniform traffic patterns”. In: *IEICE Transactions on Information and Systems* 79.8 (1996), pp. 1124–1129.

[47] Georgios Kathareios, Cyriel Minkenberg, Bogdan Prisacari, German Rodriguez, and Torsten Hoefler. “Cost-effective diameter-two topologies: Analysis and evaluation”. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM. 2015, p. 36.

[48] John Kim, Wiliam J Dally, Steve Scott, and Dennis Abts. “Technology-driven, highly-scalable dragonfly topology”. In: *2008 International Symposium on Computer Architecture*. IEEE. 2008, pp. 77–88.

[49] Michel A Kinsy, Myong Hyon Cho, Tina Wen, Edward Suh, Marten Van Dijk, and Srinivas Devadas. “Application-aware deadlock-free oblivious routing”. In: *Proceedings of the 36th annual international symposium on Computer architecture*. 2009, pp. 208–219.

[50] Smaragda Konstantinidou and Lawrence Snyder. “The chaos router”. In: *IEEE Transactions on Computers* 43.12 (1994), pp. 1386–1397.

[51] Charles E Leiserson. “Fat-trees: universal networks for hardware-efficient supercomputing”. In: *IEEE transactions on Computers* 100.10 (1985), pp. 892–901.

[52] Xuan-Yi Lin, Yeh-Ching Chung, and Tai-Yi Huang. “A multiple LID routing scheme for fat-tree-based InfiniBand networks”. In: *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings.* IEEE. 2004, p. 11.

[53] Olav Lysne, Timothy Mark Pinkston, and José Duato. “A methodology for developing deadlock-free dynamic network reconfiguration processes”. In: *IEEE Transactions on Parallel and Distributed Systems* 16.5 (2005), pp. 428–443.

[54] Daniel Maag. “Congestion-aware Simulation of Large-scale HPC Networks”. B.S. thesis. ETH Zurich, 2016.

[55] German Maglione-Mathey, Pedro Yebenes, Jesús Escudero-Sahuquillo, Pedro Javier García, and Francisco J Quiles Flor. “Combining openfabrics software and simulation tools for modeling InfiniBand-based interconnection networks”. In: *2016 2nd IEEE International Workshop on High-Performance Interconnection Networks in the Exascale and Big-Data Era (HiPINEB)*. IEEE. 2016, pp. 55–58.

[56] German Maglione-Mathey, Pedro Yebenes, Jesús Escudero-Sahuquillo, Pedro Javier García, Francisco José Quiles Flor, and Eitan Zahavi. “Scalable deadlock-free deterministic minimal-path routing engine for infiniband-based dragonfly networks”. In: *IEEE Transactions on Parallel and Distributed Systems* 29.1 (2017), pp. 183–197.

[57] German Maglione Mathey, Pedro Yebenes, Pedro Javier García, Francisco José Quiles Flor, and Jesús Escudero-Sahuquillo. “Analyzing available routing engines for infiniband-based clusters with dragonfly topology”. In: *2015 International Conference on High Performance Computing & Simulation (HPCS)*. IEEE. 2015, pp. 168–171.

[58] Mellanox. *The SHIELD: Self-Healing Interconnect (White Paper)*. 2019.

[59] Cyriel Minkenberg and Germán Rodriguez. “Trace-driven co-simulation of high-performance computing systems using OMNeT++”. In: *Proceedings of the 2nd International Conference on Simulation Tools and Techniques*. 2009, pp. 1–8.

[60] Lionel M. Ni and Philip K. McKinley. “A survey of wormhole routing techniques in direct networks”. In: *Computer* 26.2 (1993), pp. 62–76.

[61] Sabine R Ohring, Maximilian Ibel, Sajal K Das, and Mohan J Kumar. “On generalized fat trees”. In: *Proceedings of the 9th International Parallel Processing Symposium*. IEEE. 1995, pp. 37–44.

[62] OpenSM. *Current OpenSM Routing*. retrieved 2007.

[63] OpenSM. *Current OpenSM Routing § UPDN Routing Algorithm*. retrieved 2007.

[64] 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*. IEEE. 1997, pp. 87–93.

[65] Michalis Potamias, Francesco Bonchi, Carlos Castillo, and Aristides Gionis. “Fast shortest path distance estimation in large networks”. In: *Proceedings of the 18th ACM conference on Information and knowledge management*. 2009, pp. 867–876.

[66] Bogdan Prisacari, German Rodriguez, Cyriel Minkenberg, and Torsten Hoefler. “Fast pattern-specific routing for fat tree networks”. In: *ACM Transactions on Architecture and Code Optimization (TACO)* 10.4 (2013), p. 36.

[67] Bogdan Prisacari, German Rodriguez, Marina Garcia, Enrique Vallejo, Ramon Beivide, and Cyriel Minkenberg. “Performance implications of remote-only load balancing under adversarial traffic in dragonflies”. In: *Proceedings of the 8th International Workshop on Interconnection Network Architecture: On-Chip, Multi-Chip*. 2014, pp. 1–4.

[68] Jean-Noël Quintin and Pierre Vignéras. “Transitively Deadlock-Free Routing Algorithms”. In: *2nd International Workshop on High-Performance Interconnection Networks in the Exascale and Big-Data Era (HiPINEB)*. IEEE. 2016, pp. 16–24.

[69] Sven-Arne Reinemo, Bartosz Bogdanski, and Bjørn Dag Johnsen. “Multi-homed fat-tree routing with InfiniBand”. In: *2014 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. IEEE. 2014, pp. 122–129.

[70] José Rocher-Gonzalez, Jesús Escudero-Sahuquillo, Pedro Javier García, and Francisco José Quiles Flor. “On the Impact of Routing Algorithms in the Effectiveness of Queuing Schemes in High-Performance Interconnection Networks”. In: *25th Annual Symposium on High-Performance Interconnects (HOTI)*. IEEE. 2017, pp. 65–72.

[71] José Rocher-Gonzalez, Jesús Escudero-Sahuquillo, Pedro Javier García, Francisco José Quiles Flor, and Gaspar Mora. “Towards an efficient combination of adaptive routing and queuing schemes in Fat-Tree topologies”. In: *Journal of Parallel and Distributed Computing* 147 (2021), pp. 46–63.

[72] German Rodriguez, Ramon Beivide, Cyriel Minkenberg, Jesús Labarta, and Mateo Valero. “Exploring pattern-aware routing in generalized fat tree networks”. In: *23rd International Conference on Supercomputing*. ACM. 2009, pp. 276–285.

[73] German Rodriguez, Cyriel Minkenberg, Ramon Beivide, Ronald P Luijten, Jesús Labarta, and Mateo Valero. “Oblivious routing schemes in extended generalized fat tree networks”. In: *International Conference on Cluster Computing and Workshops*. IEEE. 2009, pp. 1–8.

[74] José Carlos Sancho and Antonio Robles. “Improving the up*/down* routing scheme for networks of workstations”. In: *European Conference on Parallel Processing*. Springer. 2000, pp. 882–889.

[75] Timo Schneider, Torsten Hoefler, and Andrew Lumsdaine. “ORCS: An oblivious routing congestion simulator”. In: *Indiana University, Computer Science Department, Tech. Rep* (2009).

[76] Bianca Schroeder and Garth A Gibson. “A large-scale study of failures in high-performance computing systems”. In: *Transactions on Dependable and Secure Computing* 7.4 (2009), pp. 337–350.

[77] Michael D. Schroeder, Andrew D Birrell, Michael Burrows, Hal Murray, Roger M. Needham, Thomas L. Rodeheffer, Edwin H. Satterthwaite, and Charles P. Thacker. “Autonet: A high-speed, self-configuring local area network using point-to-point links”. In: *IEEE Journal on Selected Areas in Communications* 9.8 (1991), pp. 1318–1335.

[78] Loren Schwiebert. “Deadlock-free oblivious wormhole routing with cyclic dependencies”. In: *Transactions on Computers* 50.9 (2001), pp. 865–876.

[79] Alexander Shpiner, Zachy Haramaty, Saar Eliad, Vladimir Zdornov, Barak Gafni, and Eitan Zahavi. “Dragonfly+: Low cost topology for scaling datacenters”. In: *2017 IEEE 3rd International Workshop on High-Performance Interconnection Networks in the Exascale and Big-Data Era (HiPINEB)*. IEEE. 2017, pp. 1–8.

[80] Arjun Singh. “Load-balanced routing in interconnection networks”. PhD thesis. Stanford University, 2005.

[81] Erich Strohmaier, Jack Dongarra, Hort Simon, Martin Meuer, and Hans Meuer. *TOP500 list*. www.top500.org. 1993–2020.

[82] Leslie G Valiant and Gordon J Brebner. “Universal schemes for parallel communication”. In: *Proceedings of the thirteenth annual ACM symposium on Theory of computing*. ACM. 1981, pp. 263–277.

[83] András Varga. “The OMNeT++ Discrete event simulation system”. In: *Proc. of the European Simulation Multiconference (ESM'2001)*. 2001, pp. 1–7.

[84] András Varga and Rudolf Hornig. “An overview of the OMNeT++ simulation environment”. In: *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). 2008, p. 60.

[85] Pierre Vignéras and Jean-Noël Quintin. “Fault-Tolerant Routing for Exascale Supercomputer: The BXI Routing Architecture”. In: *International Conference on Cluster Computing (CLUSTER)*. IEEE. 2015, pp. 793–800.

[86] Jesús Camacho Villanueva, Tor Skeie, and Sven-Arne Reinemo. *Routing and Fault-Tolerance Capabilities of the Fabriscale FM compared to OpenSM*. Tech. rep. Tech. rep. July, 2015.

[87] ANSI X3.159-1989. *Programming Language C*. Standard. American National Standards Institute, 1989.

[88] Jiaqing Xu, Dongjing Cai, Jie He, and Fuqiao Tang. “A Fault-Tolerant Routing Strategy with Graceful Performance Degradation for Fat-Tree Topology Supercomputer”. In: *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. IEEE. 2019, pp. 405–412.

[89] Ryota Yasudo, Michihiro Koibuchi, Koji Nakano, Hiroki Matsutani, and Hideharu Amano. “Designing high-performance interconnection networks with host-switch graphs”. In: *IEEE Transactions on Parallel and Distributed Systems* 30.2 (2018), pp. 315–330.

[90] Ryota Yasudo, Koji Nakano, Michihiro Koibuchi, Hiroki Matsutani, and Hideharu Amano. “Designing low-diameter interconnection networks with multi-ported host-switch graphs”. In: *Concurrency and Computation: Practice and Experience* (2020), e6115.

[91] Pedro Yébenes, Jesús Escudero-Sahuquillo, Crispín Gómez Requena, Pedro Javier García, Francisco José Quiles, and José Duato. “BBQ: a straightforward queuing scheme to reduce HoL-blocking in high-performance hybrid networks”. In: *European Conference on Parallel Processing*. Springer. 2013, pp. 699–712.

[92] Pedro Yébenes, Jesús Escudero-Sahuquillo, Pedro Javier García, and Francisco José Quiles Flor. “Towards modeling interconnection networks of exascale systems with OMNeT++”. In: *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. IEEE. 2013, pp. 203–207.

[93] Pedro Yébenes, Jesús Escudero-Sahuquillo, Pedro Javier García, and Francisco José Quiles Flor. “Straightforward solutions to reduce HoL blocking in different Dragonfly fully-connected interconnection patterns”. In: *The Journal of Supercomputing* 72.12 (2016), pp. 4497–4519.

[94] Pedro Yébenes, Jesús Escudero-Sahuquillo, Pedro Javier García, Francisco José Quiles Flor, and Torsten Hoefler. “Improving non-minimal and adaptive routing algorithms in slim fly networks”. In: *2017 IEEE 25th Annual Symposium on High-Performance Interconnects (HOTI)*. IEEE. 2017, pp. 1–8.

[95] Eitan Zahavi. “D-Mod-K routing providing non-blocking traffic for shift permutations on real life fat trees”. In: *CCIT Report* 776 (2010).

[96] Eitan Zahavi. “Fat-tree routing and node ordering providing contention free traffic for MPI global collectives”. In: *Journal of Parallel and Distributed Computing* 72.11 (2012), pp. 1423–1432.

[97] Eitan Zahavi, Gregory Johnson, Darren J Kerbyson, and Michael Lang. “Optimized InfiniBand™ fat-tree routing for shift all-to-all communication patterns”. In: *Concurrency and Computation: Practice and Experience* 22.2 (2010), pp. 217–231.

[98] Eitan Zahavi, Isaac Keslassy, and Avinoam Kolodny. “Quasi fat trees for HPC clouds and their fault-resilient closed-form routing”. In: *22nd Annual Symposium on High-Performance Interconnects (HOTI)*. IEEE. 2014, pp. 41–48.

[99] Feroz Zahid, Ernst Gunnar Gran, Bartosz Bogdanski, Bjørn Dag Johnsen, and Tor Skeie. “A weighted fat-tree routing algorithm for efficient load-balancing in infini band enterprise clusters”. In: *2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. IEEE. 2015, pp. 35–42.

[100] Feroz Zahid, Ernst Gunnar Gran, Bartosz Bogdanski, Bjørn Dag Johnsen, and Tor Skeie. “SlimUpdate: Minimal Routing Update for Performance-Based Reconfigurations in Fat-Trees”. In: *International Conference on Cluster Computing (CLUSTER)*. IEEE. 2015, pp. 849–856.

[101] Felix Zahn, Pedro Yebenes, Jesús Escudero-Sahuquillo, Pedro Javier García, and Holger Fröning. “Effects of Congestion Management on Energy Saving Techniques in Interconnection Networks”. In: *2019 International Workshop of High-Perfomance Interconnection Networks in the Exascale and Big-Data Era (HiPNEB)*. IEEE. 2019, pp. 9–16.

[102] Yibo Zhu, Monia Ghobadi, Vishal Misra, and Jitendra Padhye. “ECN or Delay: Lessons Learnt from Analysis of DCQCN and TIMELY”. In: *Proceedings of the 12th International Conference on emerging Networking EXperiments and Technologies*. 2016, pp. 313–327.

Contributions

[103] John Gliksberg, Jean-Noël Quintin, and Pedro Javier García. “Node-type-based load-balancing routing for Parallel Generalized Fat-Trees”. In: *2018 IEEE 4th International Workshop on High-Performance Interconnection Networks in the Exascale and Big-Data Era (HiPINEB)*. IEEE. 2018, pp. 9–15.

[104] John Gliksberg, Antoine Capra, Alexandre Louvet, Pedro Javier García, and Devan Sohier. “High-Quality Fault-Resiliency in Fat-Tree Networks (Extended Abstract)”. In: *2019 IEEE Symposium on High-Performance Interconnects (HOTI)*. 2019, pp. 9–12. DOI: 10.1109/HOTI.2019.00015.

[105] Jean-noël Quintin and John Gliksberg. *Method for establishing communication routes between nodes of a computer cluster, corresponding computer program and computer cluster*. FR 3 078 220; US Patent App. 16/280,678. 2019.

[106] John Gliksberg, Antoine Capra, Alexandre Louvet, Pedro Javier García, and Devan Sohier. “High-Quality Fault Resiliency in Fat Trees”. In: *IEEE Micro* 40.1 (2020), pp. 44–49.

[107] John Gliksberg, Alexandre Louvet, and Antoine Capra. *Rapid method for establishing communication routes between computers of a supercomputer*. FR 3 100 069; US Patent App. 16/999,262. 2021.