Ph.D. Thesis

Disjoint-path Routing in Hypercubes and their Variants

Author:
Antoine Charles Martin Bossard

Supervisor:
Pr. Keiichi Kaneko

Graduate School of Engineering
Tokyo University of Agriculture and Technology

September 2011

Abstract

Modern supercomputers are massively parallel systems which connect hundreds of thousands of processor nodes, soon millions. It is thus critical to link this huge quantity of nodes according to a suitable topology in order to retain high performance. Interconnection networks used for this purpose are thus graphs with a high connectivity. However, due to physical restrictions, additional requirements such as the diameter of the network must be satisfied so as to achieve practicability.

In this thesis, we will be addressing several routing problems inside different interconnection networks. Precisely, we shall focus on hypercubes and some of their variants, such as perfect hierarchical hypercubes. Inside these topologies, we shall describe solutions to disjoint-path routing problems such as the node-to-set disjoint-path routing problem or the set-to-set disjoint-path routing problem.

Because of the very large and continuously increasing number of computing nodes inside massively parallel systems, we understand that data routing is a critical topic to retain high performance. At the same time, given the increasing complexity of the topologies used as interconnection network of modern supercomputers, routing is, likewise, becoming increasingly challenging and requires much attention.

The routing problems addressed in this thesis are the node-to-set disjoint-path routing problem and the set-to-set disjoint-path routing problem. The former one is about finding node-disjoint paths between one source node and a set of destination nodes inside a given interconnection network. The latter one is about finding node-disjoint paths between a set of source nodes and a set of destination nodes.

Disjoint-path routing algorithms have two main advantages. First, generating disjoint paths is one solution to avoid notorious resource allocation issues of parallel systems such as deadlocks, livelocks or starvations. Second, considering the high number of nodes inside modern supercomputers, we understand that faults are likely to occur. In this context, generating disjoint paths maximises the probability of finding a fault-free path: in this case, one faulty node can effectively compromise at most one path.

Chapter 1 presents the background of our research by introducing the current context of supercomputing and massively parallel systems, as well

as by explaining the motivations of our research.

Chapter 2 recalls several general terms and notations of graph theory before reviewing numerous previous related works.

We give in Chapter 3 a formal definition of each of the interconnection networks studied in this thesis.

Chapter 4 presents an algorithm solving the node-to-set disjoint-path routing problem inside hypercubes.

Chapter 5 describes a solution to the node-to-set disjoint-path routing problem inside perfect hierarchical hypercubes. Benefiting from a low degree and a small diameter compared to a hypercube of the same size, perfect hierarchical hypercubes are suitable as interconnection network of massively parallel systems.

Then Chapter 6 states a node-to-set disjoint-path routing algorithm in metacubes. Similarly to perfect hierarchical hypercubes, metacubes enjoy a low degree and a small diameter compared to a hypercube of the same size.

Chapter 7 describes a set-to-set disjoint-path routing algorithm in perfect hierarchical hypercubes.

This thesis is finally concluded by Chapter 8.

Contents

Chapter 1

Introduction

Routing refers to *path selection*. Once a path is selected, data can be subsequently transmitted along that path. Informally, a path linking two nodes of a graph is a sequence of distinct adjacent nodes connecting these two nodes. A routing algorithm is thus in charge of generating such a sequence of adjacent nodes between two specified end-nodes. The diameter (informally the graph's breadth) and the degree (number of edges per node) of a graph (see Chapter 2) are important notions directly impacting the path generation process between two nodes.

Now, to introduce interconnection networks, we first need to talk about supercomputing. Supercomputing refers to the usage of large scale computers; by definition, supercomputers are always front when it comes to computation power. Introduced in the 60's when they were using one or very few processors, modern supercomputers are made of hundreds of thousands of nodes. Today supercomputers distribute their tasks among these nodes to perform them asynchronously as efficiently as possible: they are massively parallel systems. Because modern supercomputers are using multi-core processors, let us distinguish *meta-nodes* from regular nodes. Each meta-node can be seen as an independent computer, having its own memory and CPU unit. Today, the CPU unit of a meta-node is often made of several processors (i.e. several regular nodes). As an example, let us consider the Jaguar, built by the Cray company, ranked no.1 on the TOP500 list of the world's fastest computers as of 2010 [110]. The Jaguar combines XT5 and XT4 meta-nodes. It includes 18,688 XT5 meta-nodes, each XT5 meta-node containing a dual quad-core AMD Opteron 2356 (Barcelona) processor, thus resulting in 149,504 regular nodes. Additionally, it includes 7,832 XT4 meta-nodes, each XT4 meta-node containing one quad-core AMD Opteron 1354 (Budapest) processor, thus resulting in 31,328 regular nodes. Hence the Jaguar supercomputer includes a total of 180,382 regular computing nodes [7]. Even more recently, the K computer built by Fujitsu at the RIKEN Advanced Institute for Computational Science (AICS) in Kobe, Japan, ranked

no.1 on the TOP500 list as of 2011, currently contains 68,544 meta-nodes, each made of an octocore CPU, thus resulting in a total of 548,352 regular nodes [88, 109, 111].

An interconnection network is responsible for connecting all these nodes each other. Theoretically, an interconnection network is an undirected graph, each node of the interconnection network corresponding to one node of the graph. Each node is identified by a unique address, its format depending on the topology chosen to bind all the nodes of the network. As of today, many different interconnection networks have been described in the literature (see Chapter 2). The main concern when designing an interconnection network is the network efficiency and as such, because one want to stick closely to hardware limitations to make the network realistic, we try, among other important properties, to keep both the diameter and the degree of the corresponding graph as low as possible. As explained a few years ago, there is a physical limit of eight links per node [76]; we note that even the most recent supercomputers are sticking to this limitation [7]. Also because of hardware considerations, the regularity of a network [45] is another critical property: it is indeed much more cost-effective to build a network whose nodes are physically identical. There is no absolute best interconnection network; depending on hardware restrictions and other conditions, each manufacturer will choose the most appropriate interconnection network to build its machine. For example, the IBM Blue Gene/L is using a 3-dimensional torus interconnection network to bind its 65,536 nodes [38], whereas the SGI Origin 2000 is using a hypercube interconnection network as described in [76]. Nevertheless, today interconnection networks used for these massively parallel systems have significantly evolved and more complicated but of high performance topologies have been introduced as a replacement for the more conventional simple topologies [39, 63, 87].

Routing in the context of interconnection networks similarly refers, as explained previously, to the selection of a path between two distinct nodes to prepare data transmission between these two entities of the network. Because of the huge and steadily increasing number of compute nodes inside modern supercomputers, we understand that data routing inside such an ocean of nodes is a critical topic to retain performances of supercomputers as high as possible. Also, because of the continuously increasing complexity of the topologies in use inside supercomputers, routing is becoming more and more complicated and thus requires much attention.

Now let us discuss the importance of *disjoint-path* routing. First and foremost, disjoint-path routing is a critical topic to ensure lock-free algorithms. Effectively, as described by Duato et al. in [25], routing inside interconnection networks may have to face different blocking situations. Let us briefly go through these different routing problems. When sending data on a network, a deadlock situation occurs when the message containing the data is unable to continue its progression to the next node due to the un-

availability of that node. A node can become unavailable if it is already busy transmitting data of another message and cannot process at the same time several messages. Another deadlock case would similarly happen if a message is requesting a resource to a node which has already granted this resource to another message. In return, the requesting message may hold a resource and that until being granted access to the requested resource, thus making the current node unavailable as well. In a deadlock situation, messages are blocked forever. A second blocking situation, the livelock, occurs when messages are denied a resource by a node and are thus rerouted by the routing algorithm to another node. Hence we understand that a message can be indefinitely routed along an auxiliary path, waiting for an unavailable resource. Finally, in a network bearing an important amount of messages, a so-called starvation situation can arise when a message requests a resource always granted to other messages.

By using node-disjoint paths, data transmission performed along these paths is guaranteed to avoid any of these blocking situations; indeed a node can be used by only one message, message following one of the generated disjoint paths. Hence resources of a node are necessarily available for a message passing by, if any.

Also, disjoint-path routing can be seen as an improvement regarding efficiency. By limiting the number of messages going through a node to one, each node is thus ensured to process at most one message during the whole transmission operation. It is easy to understand that a node having to handle many messages, queued or simultaneous, is likely to affect global routing performance as this node has become a bottle-neck.

Throughout our research, we designed several routing algorithms for different interconnection networks. Firstly we focused on the node-to-set disjoint-path routing problem. To introduce this routing operation, let us start by looking at one related problem, multicasting, and its concrete applications. Multicast, a one-to-many communication scheme, is about passing data from one sender to several receivers, using, unlike node-to-set disjoint-path routing, transmission paths which are not necessarily disjoint. Multicast has applications in several domains. One of its most well-known use is for IP multicasting, which consists in distributing information from a common sender to a set of different IP addresses of the same subnet or group [98]. Also, multicasting is used by peer-to-peer technologies to efficiently deliver the same information data to several recipients [6]. Now, within interconnection networks, the node-to-set disjoint-path routing problem is generalized to the following statement: “find node-disjoint paths between one common source node and several distinct destination nodes”. In this context, generating disjoint paths also has direct applications: disjointly transmitting data from one node to several others. Because the paths are internally disjoint, it is a guarantee against blocking situations as explained previously. But more generally, as the number of nodes in massive parallel

systems continuously grows, faults are likely to occur. Hence paths disjointness is a critical asset for a routing algorithm so as to establish communication routes under a faulty environment. In this context, multicasting is not as reliable since the generated paths share several nodes at the beginning of the data transmission. Then only one fault occurring on such a shared node suffices to jeopardize data transmission. Also, another motivation for node-to-set disjoint-path routing is that since the transfer of the same message occurs simultaneously across the network, it is much more efficient to send it using a node-to-set operation than several successive node-to-node routings. Globally, destination nodes will receive the data sooner and the network will be in use a shorter period of time. And we can see here another motivation: a shorter use in time of the network leads to a more efficient algorithm regarding *Green IT* [94].

Secondly we focused on the set-to-set disjoint-path routing problem. Similarly to the node-to-set disjoint-path routing problem and for the same reasons, set-to-set disjoint-path routing is a critical issue when designing a new network topology to efficiently transmit data, this time from a set of senders to a set of receivers. Avoiding blocking situations, a set-to-set disjoint-path routing performs at once what would require many successive node-to-node routing operations, which results in a significant gain regarding the overall routing operation efficiency. Again, the network will be in use a shorter period of time, thus reducing power consumption and addressing environmental issues.

In Chapter 2 we first recall general graph theory definitions such as degree, diameter or path. Additionally, several notations frequently used throughout this work are also given. Then, we review numerous previous related works: we shall briefly present several interconnection networks and their solutions to routing problems so as to better appreciate the environment of our research which is focused on hypercube-based topologies. We shall thus extend our audit of topologies and their routing solutions to different kind of interconnection networks, such as permutation-based topologies, to have a global view of the surrounding activity of this research field.

Then, Chapter 3 formally introduces and defines the interconnection networks used in the later parts of this thesis. Additionally, several basic routing algorithms inside these topologies are provided as they will be frequently employed in following chapters.

As extensively used by next chapters, we describe in Chapter 4 a node-to-set disjoint-path routing algorithm in hypercubes. This algorithm is described in Section 4.2 of Chapter 4 and its correctness and complexities are established in Section 4.3.

Chapter 5 describes a node-to-set disjoint-path routing algorithm in perfect hierarchical hypercubes (HHC). As formally introduced in Chapter 2 and 3, the HHC topology has been introduced independently by Malluhi et al. [86] and Wu et al. [115]. This network has for main motivation that it

can connect many nodes while keeping a low degree and a small diameter. For instance, with only three links per node, the HHC network can connect 2^6 nodes. It would require twice as many links per node to a hypercube network to connect that same amount of nodes. As current hardware limits the number of links per node to eight [102], we easily understand the merit of a network keeping its degree low: it is suitable to connect a significant number of nodes where a more simple topology like a hypercube would not be satisfactory due to its high degree as explained in the previous example. Perfect hierarchical hypercubes are thus suitable as interconnection network of massively parallel systems. This algorithm is described in Section 5.2 of Chapter 5 and its correctness and complexities are established in Section 5.3.

Chapter 6 describes a node-to-set disjoint-path routing algorithm in the metacube (MC) interconnection network. As detailed in Chapter 2 and 3, metacubes have been introduced by Li et al. as an interconnection network for massively parallel systems [83]. Like perfect hierarchical hypercubes, metacubes have for merit to connect a huge number of nodes while retaining a low degree as well as a small diameter compared to an hypercube of the same size. For example, an MC network requires only six links per node to connect 2^{27} nodes whereas it would require 27 links per node to a hypercube to be able to connect that same amount of nodes. Metacubes are thus suitable as interconnection network of massively parallel systems. This algorithm is described in Section 6.2 of Chapter 6 and its correctness and complexities are established in Section 6.3.

Subsequently to the presentation of three node-to-set disjoint-path routing algorithms, we describe in Chapter 7 a set-to-set disjoint-path routing algorithm in perfect hierarchical hypercubes. This algorithm is described in Section 7.2 of Chapter 7 and its correctness and complexities are established in Section 7.3.

Finally we conclude this thesis in Chapter 8 where suggestions of future works are presented.

Chapter 2

Previous works

We present in this chapter several interconnection networks and discuss routing problems and their solutions inside these different topologies. Such routing problems include node-to-node (aka. point-to-point or unicast) routing, node-to-node disjoint-path routing, node-to-set disjoint-path routing, set-to-set disjoint-path routing and finally k-pairwise disjoint-path routing.

The two node-to-node routing problems mentioned above differ from the fact that the node-to-node *disjoint-path* routing problem is to find several internally node-disjoint paths between a random pair of nodes, whereas the node-to-node routing problem, or unicast, is to simply find one path between a random pair of nodes.

Regarding the existence, and more precisely the maximum number of disjoint paths that can be found for the aforementioned disjoint-path routing problems, we can state directly from Menger's theorem [92] that a k-connected graph (i.e. at least k nodes need to be removed to lose connectivity) implies that k disjoint paths can be found when solving the node-to-node disjoint-path, node-to-set disjoint-path or set-to-set disjoint-path routing problem. This property shall be extensively used when performing disjoint-path routing to give an upper bound of the number of disjoint paths to be found.

A maximum flow algorithm describes a generic solution to these disjoint-path routing problems [35]. However the time complexity of such an algorithm is high: it is polynomial in the number of nodes and edges of the graph. In a graph $G = (V, E)$, several implementations of an algorithm solving the maximum flow problem have been given. The time complexity of some of them is as follows: $O(|V| \cdot |E|^2)$ [28], $O(|V|^2 \cdot |E|)$ [42, 18, 43], $O(|V|^3)$ [42, 18], $O(|V| \cdot |E| \cdot \log |V|)$ [43].

For example, in an n-dimensional hypercube Q_n, solving the node-to-set disjoint-path routing problem using a maximum flow algorithm implementation like [28] requires $O(2^{3n})$ time complexity, thus exponential in n. We described a hypercube node-to-set disjoint-path routing algorithm requiring

$O(n^2)$ time complexity, this time polynomial in n [8]. This example shows that the maximum flow algorithm is not efficient when solving such routing problems, and that there is much space for improvement regarding time complexity.

As discussed in [114], due to the increasing number of processors in modern supercomputers but also to the error-prone nature of any hardware component, fault-tolerance is a critical topic when designing algorithms, hence fault tolerance as well as cluster fault tolerance topics shall also be considered in this chapter. Also, we should note that the fault tolerance property of routing algorithms is widely used by the algorithms described in Chapter 5, 6 and 7.

A first section shall recall some general definitions. Then we review interconnection networks depending on their structure; some derive from hypercubes, others from meshes, etc. We shall first focus on hypercube-based interconnection networks. Due to the popular nature of hypercubes, there exist in the literature many different topologies based on hypercubes. Then we review some permutation-based interconnection networks, one of the most famous being the star graph. A third section shall deal with mesh-based interconnection networks, and finally some other networks falling in none of the previous categories shall be reviewed in the last section of this chapter.

2.1 Preliminaries

We recall in this section several definitions of graph theory as well as notations often used hereinafter.

Definition 1 [24] *A graph $G = (V, E)$ is defined by a set of vertices (nodes) V and a set of edges $E \subseteq V \times V$.*

Definition 2 [24] *For a graph $G = (V, E)$, any graph $G' = (V', E')$ with $V' \subseteq V$ and $E' \subseteq E$ is a* subgraph *of G.*

Definition 3 [24] *In a graph $G = (V, E)$, two nodes u, v are* adjacent*, or* neighbours*, if $(u, v) \in E$.*

Definition 4 [24] *A node u is* incident *to an edge e if $v \in e$.*

Definition 5 [24] *The* degree *of a node u is the number of edges starting at u, that is the number of neighbours of u; it is denoted by $d(u)$.*

Definition 6 [24] *If all the nodes of a graph G have the same degree k, then G is k-*regular*. We say that G has a* degree *k.*

Definition 7 [24] *In a graph $G = (V, E)$, a* path *$P = (V', E')$ is a subgraph of G (ie. $V' \subseteq V$ and $E' \subseteq E$) of the form*

$$V' = \{u_0, u_1, \ldots, u_k\} \quad E' = \{(u_0, u_1), (u_1, u_2), \ldots, (u_{k-1}u_k)\}$$

The length *of P corresponds to its number of edges, that is k. P shall be referred to as simply $u_0 \to u_1 \to \ldots \to u_k$ or in more concise form $u_0 \rightsquigarrow u_k$. u_0 and u_k are referred to as the* end-nodes *of P.*

Definition 8 [45] *Two paths $P = (V, E)$ and $P' = (V', E')$ are* node-disjoint *(abbr.* disjoint*) if $V \cap V' = \emptyset$.*

Definition 9 [45] *Two paths $P = u_0 \to u_1 \to \ldots \to u_{k-1} \to u_k$ and $P' = u'_0 \to u'_1 \to \ldots \to u'_{k-1} \to u'_k$ are* internally disjoint *if $\forall i, j, 1 \leq i, j \leq k-1, u_i \neq u'_j$.*

Definition 10 *A* subpath *of a path P is a subgraph of P.*

Definition 11 [24] *The* distance *between two nodes u, v, denoted by $d(u, v)$, is the length of a shortest path linking u and v.*

Definition 12 [24] *The* diameter *of a graph G, denoted by $diam(G)$, is the greatest distance between any two nodes of G.*

Definition 13 *The* parity *of a bit sequence corresponds to the parity of its number of bits set to 1.*

Definition 14 [45] *The* center *of a graph $G = (V, E)$ is the node $c \in V$ such that the sum of the distances $\sum_{v \in V} d(c, v)$ is the smallest as possible.*

Notation 1 *The set of all the neighbours of a node u is denoted by $N(u)$.*

Notation 2 *The Hamming distance between two bit sequences a and b is denoted by $H(a, b)$.*

Notation 3 *The length of a path P is denoted by $L(P)$.*

Notation 4 *An arbitrary shortest path between any two nodes a, b is denoted by $a \overset{\text{SPR}}{\rightsquigarrow} b$.*

Notation 5 *An edge between a node u and a node in a set V is denoted by $u \to V$.*

Notation 6 *The binary exclusive OR operation (XOR) is denoted by $\oplus$.*

2.2 Hypercube-based interconnection networks

The hypercube topology has been used in supercomputing as soon as the early 80's, the Caltech Cosmic Cube being the first parallel computer based on this network. At that time, the Caltech Cosmic Cube used a six dimensional hypercube network to bind 64 processors [101].

Shortly came in 1985 the Intel iPSC/1 which at that time was made of 32 to 128 CPU nodes (7-dimensional hypercube) [90], as well as the nCUBE 10, first parallel computer of the NCUBE company. This time, the nCUBE 10 was built upon a 10-dimensional hypercube allowing the connection of 1024 CPU nodes [58].

More recently, SGI introduced in 1996 the Origin 2000 family of computers [76], including from 2 to at most 128 CPU nodes depending on the model. It was followed in 2000 by the Origin 3000 family [10], including from 2 to at most 512 CPU nodes again depending on the model.

Hypercubes remain today a very popular network topology thanks to their simplicity and interesting properties as explained later in this section. By using hypercubes either as low-level entities (we usually use the terms nucleus or cluster) as for hierarchical cubic networks [41, 13] or metacubes [83], or as high-level structure as for cube-connected cycles [97], hypercube-based interconnection networks can rely on these important properties. Due to this popularity, existing hypercube-based interconnection networks clearly outnumber any other category of interconnection networks.

2.2.1 Hypercubes

As presented in the introduction of this section, the hypercube topology is a popular interconnection network thanks to many interesting properties. Routing in parallel computers built upon the hypercube topology has been largely discussed by Rabin in [99]. As explained by Saad et al. in [100], a hypercube has a recursive structure: an n-dimensional hypercube (or simply n-cube) Q_n is made of two distinct $(n-1)$-dimensional hypercubes. This property shall be extensively used when designing disjoint-path routing algorithms. Another critical property when it comes to routing is that two adjacent nodes of a hypercube have addresses differing in one and only one bit position. Thanks to its properties and simplicity, many interconnection networks are using a hypercube either as low level cluster (e.g. metacubes [83]) or as a high level structure connecting different clusters (e.g. cube-connected cycles [97]).

Each node of a Q_n has an n-bit address. Two nodes are adjacent if and only if their addresses differ in one single bit. A Q_n is made of two hypercubes (also called subcubes) Q^0_{n-1} (abbr. Q^0) and Q^1_{n-1} (abbr. Q^1), where for a dimension i, Q^0 is induced by the set containing all the nodes of Q_n whose i-th bit is set to 0, and Q^1 is induced by the set containing all the

nodes of Q_n whose i-th bit is set to 1.

A node-to-node disjoint-path routing algorithm in a hypercube was described in [100]. Because a Q_n is n-connected, Menger's theorem [92] shows that there exist n internally disjoint paths between any two nodes s and d. The main idea of this algorithm to ensure path disjointness is to start each path by flipping a bit at a distinct dimension for each path. We understand that during the first flipping operation, a bit may be flipped although it was already set to its final value, that is s and d have the same bit value on the dimension of the flipped bit. Hence because we need in such case to flip again that bit, this algorithm generates paths of length at most $H(s,d)+2$ in linear time complexity.

A first hypercube fault-tolerant node-to-node routing algorithm was introduced in [47]. In a Q_n containing at most $n-1$ faulty nodes, this algorithm finds a fault-free path between any two nodes s and d. The main idea is to reduce Q_n into Q^0 and Q^1 so that s and d are not inside the same subcube, before recursively applying the algorithm onto the subcube containing the least faulty nodes. This divide-and-conquer approach is often used when routing inside hypercubes and more generally inside recursive topologies. The source or destination node may need to be routed to the opposite subcube with a fault-free path of length at most two before the recursion. Once a fault-free subcube is reached, the two nodes are connected using a shortest-path routing algorithm. Since a Q_n can be reduced at most $\lceil \log_2(n-1) \rceil$ times until one subcube is fault free, there will be therefore at most $\lceil \log_2(n-1) \rceil$ paths of length two used before applying a shortest-path routing algorithm, hence the generated path will be of length at most $2\lceil \log_2(n-1) \rceil + (H(s,d) - \lceil \log_2(n-1) \rceil) = H(s,d) + \lceil \log_2(n-1) \rceil$. Because the number of faulty nodes is at least divided by two at each iteration of the algorithm, finding a fault-free path to route s or d to the opposite subcube is taking at the k-th iteration of the algorithm $O(n/2^k)$ tests to complete. Hence we obtain a total time complexity of $O(n) + O(\sum_{k=0}^{\lceil \log_2(n-1) \rceil} n/2^k) = O(n)$.

This previous result has been improved in [50]. Because a path of length two can be used to route s or d to the opposite subcube, a penalty edge (i.e. not on a shortest-path toward s or d) may be introduced. This new algorithm gets rid of such penalty edge by carefully selecting which dimensions to use when routing s or d into the opposite subcube. If possible, only nodes included in a shortest-path from s to d shall be used. There may be a case where all neighbours of s or d included on such a shortest-path are faulty nodes. In this case, a neighbour node not included on a shortest-path shall be used, thus introducing two penalty edges (that dimension will be flipped twice). This case shall occur only once hence the path generated has a length of at most $H(s,d)+2$. Time complexity remains $O(n)$.

Finally, Gu et al. showed in [49] that the maximal length of the paths generated by this hypercube fault-tolerant node-to-node algorithm is in practice at most $n+1$ inside a Q_n.

A hypercube node-to-set disjoint-path routing algorithm has been proposed by Latifi et al. [75]. In a Q_n it can find k disjoint paths between one common source node s and k destination nodes $D = \{d_1, d_2, \ldots, d_k\}$ ($k \leq n$). This algorithm reduces a Q_n into two subcubes Q^0 and Q^1, but this time the reduction is made along using the rightmost bit, that is the dimension 0. Assume s is set to 0 (hence $s \in Q^0$). One path is constructed inside Q^1 and the algorithm is recursively applied onto Q^0 only, routing unprocessed destination nodes of Q^1 with disjoint-path of length at most two into Q^0. We understand a path will suffer a penalty if its destination node d_j is blocked by other destination nodes from being connected to a node of Q^0 by flipping a bit set to 1. By balancing the number of bits of d_j set to 1 and the number of needed blocking destination nodes, we can deduce that paths will never include more than $n+1$ edges.

A fault-tolerant set-to-set disjoint-path routing algorithm in hypercubes was given in [46]. In a Q_n, given two sets of $k \leq n$ non-faulty nodes S the source nodes and D the destination nodes, and a set of at most $n-k$ faulty nodes, the algorithm finds k fault-free disjoint paths between the nodes of S and the nodes of D. The main idea of the algorithm to achieve path disjointness is, once again, to follow a divide-and-conquer strategy. Q_n is reduced into Q^0 and Q^1 using the dimension i such that $D \cap Q^0 \neq \emptyset$ and $D \cap Q^1 \neq \emptyset$. The main task is then to balance the number of source and destination nodes inside each subcube before recursively solving the problem onto Q^0 and Q^1 , respectively. A fault-tolerant node-to-node routing algorithm shall be used when reaching a subcube containing only one pair of source and destination nodes. Since at most two edges are needed to route a source or destination node to the opposite subcube, the paths have a length of at most $(n-1)+(k-1)+2 = n+k$. The time complexity $T(k,n)$ of the algorithm can be expressed as $T(k,n) = T(k_1, n-1) + T(k_2, n-1) + O(n), k_1 + k_2 = k$ ($O(n)$ is the time complexity to reduce a Q_n, that is especially to balance the number of source and destination nodes inside each subcube). With further calculations we obtain a value of $T(k,n) = O(kn + kO(n))$ finally simplified to $O(n \log k)$.

Let us now focus on cluster fault-tolerant (CFT) routing algorithms in hypercubes. As explained previously, there is an evident motivation for fault tolerance in algorithms. The aim for cluster fault tolerance is to group faulty nodes into subgraphs of small diameter so that it becomes easier for routing algorithm to handle faults inside the network. Concretely, a faulty cluster is a connected subgraph whose nodes are all faulty.

A hypercube CFT node-to-node routing algorithm was described in [49]. In a Q_n, given a set F of at most $2n-3$ faulty nodes grouped into at most $n-1$ clusters of diameter at most one, this algorithm finds a fault-free path between two non-faulty nodes s and d. The main idea of the algorithm is to reduce Q_n into two subcubes Q^0 and Q^1 such that s and d are inside different subcubes, say $s \in Q^0$. Then depending on which subcube contains the more faulty nodes, we recursively route s toward d inside the opposite subcube, that is the less faulty subcube, using a fault-tolerant node-to-node routing algorithm in hypercubes. If there exists a fault-free path of length at most two connecting $s \in Q^0$ to a node $s' \in Q^1$, then we apply the fault-tolerant node-to-node algorithm onto Q^1 to connect s' and d. Otherwise we have three edges to connect s to a node in Q^1, plus at most $n-1$ edges for shortest-path routing in Q^1. In both cases we have a total of at most $3 + (n-1) = n+2$ edges. The time complexity of this algorithm is linear in n: finding a fault-free path to connect s to a node in Q^1 takes $O(n)$ time since there are at most $2n-3$ faulty nodes, and both a shortest-path routing as well as a fault-tolerant node-to-node in hypercubes are in $O(n)$ time.

A cluster fault-tolerant node-to-set disjoint-path routing algorithm in hypercubes was described in [55]. In a Q_n, given a non-faulty nodes s, a set of $k, 2 \leq k \leq n$ non-faulty nodes D and a set of at most $n-k$ faulty clusters of diameter at most one (ie. at most $2(n-k)$ faulty nodes), this algorithm can find k fault-free disjoint paths connecting s to every node of D. The main idea of the algorithm is to follow a divide-and-conquer approach by reducing Q_n into Q^0 and Q^1 such that $D \cap Q^0 \neq \emptyset$ and $D \cap Q^1 \neq \emptyset$ hold. Assume that $s \in Q^0$. The routing problem is recursively solved in Q^1 and Q^0, respectively. A separate routing procedure is used when two destination nodes remain, ensuring at most $n+2$ edges in $O(n)$ time. By induction hypothesis the paths in Q^1 are of length at most $(n-1)+2$. Since at most one additional edge is required to connect paths of Q^1 to s, we have a maximum path length of $(n-1)+2+1 = n+2$. As for the time complexity of the algorithm we have $T(k,n) = T(|D \cap Q^0|, n-1) + T(|D \cap Q^1|, n-1)$ with $|D \cap Q^0| + |D \cap Q^1| = k$ and $T(k,n) = O(n)$ for $k \leq 2$. Therefore $T(k,n) = O(kn)$.

A CFT set-to-set disjoint-path routing algorithm in hypercubes was introduced in [55]. In a Q_n, given two sets of k $(2 \leq k \leq n)$ non-faulty nodes S and D and at most $n-k$ faulty clusters of diameter at most one, this algorithm finds k fault-free disjoint paths between a node of S and a node D. The main idea of this algorithm is to apply a divide-and-conquer approach by reducing Q_n into Q^0 and Q^1 such that $D \cap Q^0 \neq \emptyset$ and $D \cap Q^1 \neq \emptyset$ hold. The key is to balance inside each subcube the number of source and destination nodes before recursively applying this algorithm onto Q^0 and Q^1 recursively. Hence some source nodes are routed

to the opposite subcube with fault-free disjoint paths of length at most two. A separate routing algorithm is used when reaching a subcube containing at most two source and two destination nodes, ensuring at most $n+4$ edges in $O(n)$. Regarding the maximal path length $L(k,n)$, we have $L(k,n) \leq \max\{L(k_1,n-1), L(k_2,n-1)\}+2$ with $k_1+k_2=k$ since we need at most two edges to connect a source node of Q^0 to node in Q^1. Also $L(2,n) \leq n+4$ and $L(1,n) \leq n+2$, hence $L(k,n) \leq n+k+2$. For the time complexity we have $T(k,n) = T(k_1,n-1)+T(k_2,n-1)+O(kn)$ with $k_1+k_2=k$ since for each of the $O(k)$ source nodes it takes $O(n)$ to find a fault-free disjoint path routing inside the opposite subcube. Also $T(1,n)=O(n)$, hence $T(k,n)=O(kn+k\cdot kn)=O(k^2n)$, later reduced to $O(n\log k)$.

A first cluster fault-tolerant k-pairwise disjoint-path routing algorithm in hypercubes was given in [52]. In a Q_n ($n \geq 4$), this algorithm finds k fault-free disjoint paths connecting each pair $(s_1,t_1),\ldots,(s_k,d_k)$ ($2 \leq k \leq \lceil n/2 \rceil$) with at most $n-2k+1$ faulty clusters of diameter one. The main idea of the algorithm is to reduce Q_n into Q^0 and Q^1 such that at least one the k pairs (s_j,d_j) is separated: $s_j \in Q^0$ (resp. $s_j \in Q^1$) and $d_j \in Q^1$ (resp. $d_j \in Q^0$) to then recursively solve the problem onto Q^0 and Q^1, recursively. Again, the key is to balance the number of nodes s_i and d_j inside each subcube. This is achieved by routing with fault-free disjoint paths of length at most three some nodes d_i into the opposite subcube before the recursive call. The induction stops when there remain only one or two pairs to connect (CFT node-to-node routings). A CFT node-to-node routing generates a path of length at most $n+2$, and a CFT algorithm connecting two pairs at most $n+4$. The maximum path length $L(k,n)$ is thus given by $L(k,n)=L(k-1,n-1)+3$ ($k \geq 3$) and $L(1,n)=n+2$, $L(2,n)=n+4$. Hence $L(k,n)=2n$. Regarding the time complexity $T(k,n)$, we have $T(k,n)=T(k_0,n-1)+T(k_0,n-1)+O(n)$ with $k_0+k_1=k$ and $T(1,n)=O(n)$. With further calculations we obtain $T(k,n)=O(kn+kO(n))$ finally simplified to $O(kn\log k)$.

This CFT k-pairwise disjoint-path routing algorithm in hypercubes has been improved in [57] by significantly reducing the maximum length of the generated paths. The main idea for this improvement is to balance the number of pairs (and not only the number of s_i, d_j) to connect inside each subcube. By balancing the number of pairs connected to one subcube so that each subcube contains almost the same number of pairs to connect, the number of hypercube reductions will significantly be reduced, and the penalty edges to rejoin separated pairs will be decreased since each pair will be separated at most $\lceil \log k \rceil$ times. Hence in the worst case, CFT node-to-node routing will be applied in a subcube of dimension $n-\lceil \log k \rceil$, thus requiring at most $n-\lceil \log k \rceil+2$ edges. With at most two edges to rejoin a

separated pair, we obtain in that worst case of $\lceil \log k \rceil$ hypercube reductions a path of length at most $(n - \lceil \log k \rceil + 2) + 2\lceil \log k \rceil = n + \lceil \log k \rceil + 2$.

2.2.2 Cube-connected Cycles

We review in this section the cube-connected cycles (CCC) network first described by Preparata and Vuillemin [97]. The CCC network has a 2-level structure: it connects cycles using a hypercube topology. Each node of any CCC network has a degree of at most three: at most two cycle edges and at most one hypercube edge. This interconnection network has for merit that compared to a hypercube of the same size it has a smaller diameter and lower degree. For example a $CCC(3,3)$ also denoted CCC_3 connects 24 nodes with only three links per node, whereas a hypercube network requires at least 5 links per node to connect that same number of nodes.

A $CCC(n,k)$ is parametrized by n the dimension of the underlying hypercube connecting cycles of length k. Each has an address (c,p) where c identifies the cycle including the node and p represents the position of the node inside that cycle. Hence $0 \leq c \leq 2^n - 1$ and $0 \leq p \leq k - 1$ hold. A node (c,p) is adjacent to the following at most three nodes: $(c,(p+1) \bmod k), (c,(p-1) \bmod k)$ (inducing cycle edges) and $(c \oplus 2^p, p), 0 \leq p \leq n-1$ (inducing hypercube edges). From this definition we understand that k is necessarily greater or equal than n. We also deduce that inside each cycle $k - n$ nodes are not incident with a hypercube edge and are thus of degree at most two. One should note that the special case $CCC(n,n)$, also denoted CCC_n, is frequently used as a simplification of CCC networks as it is regular and vertex symmetric.

A node-to-node routing algorithm in $CCC(n,k)$ finding a shortest-path between $(0,0)$ and any node (c,p) was given in [91]. The main idea of this algorithm is to traverse cycles so that the positions of the bits set to 1 in c are visited using the minimum number of cycle edges.

The problem can be summarized as finding a shortest-path which goes through all positions set to 1 from the position 0 to the position p on the cycle of length n made of 1's when a hypercube edge must be traversed and of 0's otherwise. Traversing optimally the positions set to 1 in that cycle depends on the respective positions of p and the leftmost / rightmost bits set to 1 in c. Eight traversing types can be distinguished, each having at most two inflection points (direction inversion), leading to at most $n + \lfloor n/2 \rfloor - 2$ cycle edges if $k = n$, at most $n + \lfloor k/2 \rfloor - 1$ cycle edges if $n+1 \leq k \leq 2n-2$ and at most k cycle edges if $k \geq 2n-1$. Adding the at most n hypercube edges, we obtain a diameter for $CCC(n,k)$ of $2n + \lfloor n/2 \rfloor - 2$ if $k = n > 3$, $2n + \lfloor k/2 \rfloor - 1$ if $n+1 \leq k \leq 2n-2$ or $n+k$ if $k \geq 2n-1$.

2.2.3 Perfect Hierarchical Hypercubes

Hierarchical hypercubes of dimension n have been introduced in the literature by Malluhi and Bayoumi in [86]. A perfect hierarchical hypercube (HHC) is a $(2^m + m)$-dimensional hierarchical hypercube. The HHC network has a 2-level structure: on the low-level, nodes are connected each other forming several distinct hypercubes, and on the high-level, each node of one low-level hypercube is connected to another node of a distinct low-level hypercube. This idea of connecting hypercubes has motivated Wu et al. to name this interconnection network Cube-connected Cubes when they described it independently in [115]. The main motivation for the perfect hierarchical hypercube interconnection network is that it can connect many nodes in comparison with its low degree as well as its small diameter. For example, an HHC_6 can connect 2^6 nodes with only 3 links per node, half less than a hypercube Q_6 of the same size.

A $(2^m + m)$-dimensional (i.e. perfect) hierarchical hypercube is denoted by HHC_{2^m+m}. Each node of an HHC_{2^m+m} is a pair of a 2^m-bit sequence, the subcube ID, and a m-bit sequence, the processor ID. Two nodes $a = (\sigma_a, \pi_a)$ and $b = (\sigma_b, \pi_b)$ are adjacent if and only if one of the following conditions holds: $\sigma_a = \sigma_b$ and $H(\pi_a, \pi_b) = 1$, or, $\sigma_a = \sigma_b \oplus 2^{\pi_b}$ and $\pi_a = \pi_b$. Edges induced by the first condition are called internal edges, whereas edges induced by the second condition are called external edges. From this we can deduce the degree of HHC_{2^m+m} is $m + 1$. Also we note that the nodes having the same subcube ID σ induce an m-dimensional hypercube, denoted subcube $Q_m(\sigma)$.

A node-to-node routing algorithm in HHC was given in [86]. In an HHC_{2^m+m}, the algorithm finds a path connecting any two nodes $s = (\sigma_s, \pi_s)$ and $d = (\sigma_d, \pi_d)$. The main idea is to follow a Hamiltonian path in an m-dimensional hypercube starting from π_s to π_d (or a neighbour of π_d if the parity of $\pi_s \oplus \pi_d$ is even) for internal edges. External edges are then simply inserted at the appropriate place inside this Hamiltonian path to obtain the final path. We understand that at most 2^m external edges will be required to set all the bits of σ_s to σ_d, and at most 2^m internal edges for the Hamiltonian path in an m-dimensional hypercube as described. Hence the maximum path length of the generated path is at most 2^{m+1} and we can deduce that the diameter of an HHC_{2^m+m} is 2^{m+1}.

A node-to-node disjoint-path routing algorithm in an HHC was described in [116]. In an HHC_{2^m+m}, the algorithm finds $m+1$ disjoint paths connecting any two nodes $s = (\sigma_s, \pi_s)$ and $d = (\sigma_d, \pi_d)$. The main idea of this algorithm to achieve path disjointness is to route each path inside distinct subcubes (i.e. not used by any other path). Concretely, considering the sequence of

dimensions of differing bits of σ_s and σ_d following an m-bit Gray code, each path shall start with an external edge according to a distinct dimension of the Gray code considered, and then connect d by traversing the m-bit Gray code until reaching σ_d, performing routing inside each subcube. All the paths may not be able to start from a dimension differing in σ_s and σ_d, hence the remaining paths introduce two additional external edges to start by visiting a distinct subcube after the subcube of s. By discussing the number of edges required inside each subcube, we obtain paths of length at most $\max\{2^m+2m+r, 2^m+m+r+4\} \leq \max\{2^{m+1}+2m+1, 2^{m+1}+m+4\}$.

2.2.4 Dual-Cubes

Dual-cubes [77] have a two-level structure: clusters (hypercubes) on the low-level, which are linked each other on the high-level depending on their class. There are two classes in a dual-cube, each cluster is member of one unique class. While retaining interesting properties of hypercubes, dual-cubes can connect many more nodes than a hypercube of the same size, while retaining a small diameter and a low degree.

Each node of a dual-cube F_r has an address of $1 + 2(r-1) = 2r-1$ bits: the leftmost bit represents the class of the node. The remaining bits are divided into two sequences of $r-1$ bits (left and right). Two nodes $u = (u_{2r-1} \dots u_1)$ and $v = (v_{2r-1} \dots v_1)$ are adjacent if the following three conditions hold: 1. u and v differ in one bit, say the i-th bit, 2. $1 \leq i \leq r-1 \Rightarrow u_{2r-1} = v_{2r-1} = 0$ and 3. $r \leq i \leq 2r-2 \Rightarrow u_{2r-1} = v_{2r-1} = 1$ or if 4. $u_i = v_i, 1 \leq i \leq 2r-2$. Condition 4. induces cross-edges. The cluster of one node (clusterID) is identified by the class bit and the left $(r-1)$-bit sequence if the class is 0, and by the right $(r-1)$-bit sequence if the class is 1. Clusters are $(r-1)$-dimensional hypercubes induced by the remaining $(r-1)$-bit sequence, called nodeID.

A fault-tolerant node-to-node routing algorithm in a dual-cube F_r having at most $r-1$ faulty nodes finding a path between any two nodes s and d was given in [78]. The main idea is to consider s, d as nodes of an extended cube so as to apply a fault-tolerant node-to-node routing in hypercubes. The clusters of s and d, both $(r-1)$-cubes, are linked with paths traveling to one other distinct cluster if the classes of s and d are identical, and otherwise there exists one pair of a node of the cluster of s and a node of the cluster of d that are connected with a single cross-edge, and all the $r-2$ other pairs can be connected via two clusters, plus three cross-edges. Hence considering two clusters as an r-cube, we can apply a hypercube node-to-node fault-tolerant routing to this extended cube, considering a node as faulty if the path connecting it to the opposite cluster is not fault-free. By discussing the number of edges inside and between the two clusters we obtain a max-

imum path length of $(r+2)-1+1+(r-1)+1+(r-1)+1 = 3r+2$. Deducing the faulty-nodes from the faulty-edges takes $O(r)$ time and hypercube fault-tolerant node-to-node routing requires $O(r)$ time in a Q_r, hence fault-tolerant node-to-node routing in an F_r is $O(r)$ time complexity.

A node-to-node disjoint-path routing algorithm in dual-cubes was described in [78]. In an F_r this algorithm finds r disjoint paths between any two nodes s and d. As for fault-tolerant node-to-node routing, the main idea of this algorithm is to consider the two clusters of s and d as an extended cube to be able to use a hypercube node-to-node disjoint-path routing algorithm. We obtain a maximum path length of $1+(r-1)+1+(r-1)+1 = 2r+1$ by performing in-cluster routing and replacing some extended cube edges by the corresponding dual-cube paths. Because a hypercube node-to-node disjoint-path routing algorithm can be solved in $O(n^2)$ time in a Q_n, this algorithm in a dual-cube has a time complexity of $O(r^2)$.

A node-to-set disjoint-path routing algorithm in dual-cubes was proposed in [68]. In an F_r this algorithm finds r disjoint paths between one source node s and r destination nodes $d_1, d_2, \ldots, d_r$. The main idea of this algorithm is to route with disjoint paths s and each destination node to clusters so that each cluster does not have more than $r-1$ destination nodes. Finally a hypercube fault-tolerant set-to-set routing algorithm is applied inside each such cluster generating paths of length at most $2(r-1)$. Since hypercube fault-tolerant node-to-node routing can be applied once when finding paths from s, we obtain paths of length at most $r+2+(2r-2)+2 = 3r+2$. The dominant time complexity of this algorithm is that of the hypercube set-to-set disjoint-path routing algorithm, that is $O(r^2 \log r)$.

Lastly a set-to-set disjoint-path routing algorithm in dual-cubes was given in [69]. In an F_r this algorithm finds r disjoint paths between a set of source nodes $S = \{s_1, s_2, \ldots, s_r\}$ and a set of destination nodes $D = \{d_1, d_2, \ldots, d_r\}$. The main idea of the algorithm is to distribute source and destination nodes such that we have inside each cluster the same number of sources (including distributed sources) and destination nodes (including distributed destination nodes). Then inside each cluster containing at least one pair of source and destination nodes, we disjointly connect these pairs by using a hypercube set-to-set disjoint-path routing algorithm inside that cluster. In the worst case, distributing a source node or a destination node requires $r+2$ edges and in-cluster routing $r-1$ edges, hence we obtain a path of length $3r+3$. The dominant time complexity of the algorithm comes from the set-to-set disjoint-path routing algorithm inside an $(r-1)$-cube, connecting $O(r)$ source and destination nodes, that is a total time complexity of $O(r^2 \log r)$.

2.2.5 Metacubes

The metacube interconnection network (MC) [83] is a generalization of dual-cubes. It has a two-level cubic structure: clusters are on the low level, each is member of a class. The high level is responsible for linking nodes of different classes each other. MC can connect a huge quantity of nodes while keeping its degree and diameter small compared to a hypercube of the same size.

Each node address of an $MC(k, m)$ is made of $k + m2^k$ bits, it is of the form $(c, m_{2^k-1}, \ldots, m_1, m_0)$ with c representing the class of the node (classID) where c occupies k bits. $(m_{2^k-1}, \ldots, m_{c+1}, m_{c-1}, \ldots, m_1, m_0)$ represents the $m(2^k - 1)$-bit clusterID, and finally, m_c is the m-bit nodeID. Each cluster is an m-dimensional hypercube induced by the m-bit nodeID (cube-edges). Two nodes of distinct clusters are adjacent if and only if their classIDs differ in one single bit, all the other $m2^k$ bits being identical (cross-edges).

A node-to-node routing algorithm in metacubes was given in [80]. In a $MC(k, m)$ this algorithm finds a path between any two nodes s and d. The main idea is to follow an Hamiltonian path of the classes from the class $c(s)$ of s to the class $c(d)$ of d and to modify inside each cluster the m-bits according to that bits in d. We can deduce from this algorithm the distance between s and d as being 2^k cross-edges to perform the Hamiltonian path on a k-cube, plus the Hamming distance between $\bar{c}(s)$ and $\bar{c}(d)$, formally: $2^k + H(\bar{c}(s), \bar{c}(d))$ where $\bar{c}(u)$ represents all the bits of u excepted the left-most k bits. From this we can deduce the diameter of a metacube $MC(k, m)$ as being $2^k + m2^k = 2^k(m + 1)$.

A node-to-node disjoint-path routing algorithm in metacubes was first introduced in [79] and clarified in [81]. This algorithm aims at finding inside $MC(k, m)$ $k + m$ disjoint paths between any two nodes s and d. The main idea of the algorithm is to connect the neighbours of s inside the cluster of s to the neighbours of d inside the cluster of d, and the neighbours of s outside the cluster s to the neighbours of d outside the cluster of d. To ensure path disjointness, the algorithm introduces the idea of signature: each path shall hold its signature through a bit of the node address. Path disjointness is guaranteed since routing shall occur inside distinct clusters thanks to this signature bit. Regarding the maximum path length, we have 2^k cross-edges to perform a Hamiltonian path of all the classes, plus $H(s, d)$ including the class bits since after finding a signature we leave the class without modifying the current nodeID, thus we need to get back later to that class which requires this amount of cross-edges. Penalty edges must also be added for the signature, four at most, and disjoint class paths (cross-edges only) requires at most $k + 1$ cross-edges for disjoint-path routing inside a k-cube. In total we have paths of length at most $H(s, d) + 2^k + k + 5$.

We conclude this section on hypercube-based interconnection networks by mentioning several important others: Möbius Cubes [19], Folded Hypercubes [30], Hierarchical Folded Hypercubes [27], Hierarchical Cubic Networks [41, 120, 37], Generalized Hypercubes [5], Extended Hypercubes [74], Crossed Cubes [29], Cyclic Cubes [36], Augmented Cubes [14], Folded Petersen Cubes [95], Fibonacci Cubes [59], Fully Connected Cubic Networks [11, 117], Multilevel Hypercubes [1], Recursive Cube of Rings [105].

2.3 Permutation-based interconnection networks

We focus in this section on graphs defined by permutations on groups of symbols. Like hypercubes, these graphs are Cayley graphs, they correspond to the permutation group. The main motivation for such graphs is that both their diameter and degree are lower than that of a hypercube of the same size. Hence these interconnection networks allow to bind more efficiently (less links per node) generally more processors than hypercubes, while retaining important properties of Cayley graphs. Routing is, however, of higher difficulty compared to routing inside hypercube interconnection networks.

2.3.1 Star Graphs

Star graphs have been introduced in the literature by Akers et al. in [2, 3] as a particular Cayley graph. They have a smaller diameter and a lower degree than a hypercube of the same size.

Each node of an n-dimensional star graph G_n corresponds to one permutation of the set $\{1, 2, \ldots, n\}$. Each node is linked to every other permutation obtained by swapping the first symbol with any of the others, that is each node $(p_1, p_2, \ldots, p_n)$ of G_n is adjacent to the $n-1$ nodes $(p_i, p_2, p_3, \ldots, p_{i-1}, p_1, p_{i+1}, \ldots, p_n)$ $(2 \leq i \leq n)$. Also, G_n is made of n subgraphs G_{n-1}: considering one of the n positions of node addresses, a subgraph G_{n-1} is induced by the nodes having the same symbol at that position. Since there are n symbols possible, G_n contains n subgraphs G_{n-1}. Also there are n possible ways to reduce G_n into n substars G_{n-1}, one for each different position considered.

A simple node-to-node routing algorithm finding a path between any two nodes was described in [3]. The main idea is to check if the first symbol of the current node is already at its final position. If so, then we swap this first symbol with a symbol which is not in its final position. Otherwise we swap the first symbol with the symbol standing at the final position of the first symbol. The algorithm is applied recursively until each symbol has reached its final position. Akers et al. [3] showed that the diameter of G_n

is $\lfloor 3(n-1)/2 \rfloor$.

A fault-tolerant node-to-node routing algorithm in star graphs was described in [47]. In a G_n, given two nodes s and d and a set F of at most $n-2$ faulty nodes, this algorithm finds a fault-free path between s and d. The main idea of this algorithm is to connect s and d with a fault-free path of length at most two or three to a fault-free subgraph G_{n-1} if any, and in this case the path is completed by applying a shortest-path routing inside this G_{n-1}, or to the subgraph G_{n-1} containing the least faulty nodes, and in this case the algorithm is then applied recursively onto this G_{n-1}. By discussing the number of edges required to reach the selected G_{n-1} from s, d as well as the distance between s and d, we obtain a maximum path length of $\min\{diam(G_n) + 3, d(s,d) + 6\}$. s and d are connected to the selected G_{n-1} in $O(n)$ time since there are at most $n-2$ faulty nodes. Shortest-path routing in G_{n-1} is also $O(n)$ time, therefore this algorithm requires $O(n)$ time complexity.

A node-to-set disjoint-path routing algorithm in star graphs was introduced in [53]. This algorithm generates disjoint paths between a source node s and $k \leq n-1$ destination nodes $D = \{d_1, d_2, \ldots, d_k\}$. A first version of the algorithm returns paths of length at most $diam(G_n) + 3$. Assuming s is the identity permutation, the main idea is to first distribute destination nodes to nodes whose first symbol is 1, with a disjoint path of length at most two, and then to use a shortest-path routing to reach the identity permutation in $O(n)$ time. Hence the total time complexity of this algorithm is $O(kn)$. A second version of the algorithm reduces the maximal path length to $diam(G_n) + 1$. The main idea is to distribute destination nodes to nodes whose first symbol is 1 by following a shortest-path routing toward the identity permutation. Concretely this is achieved by prioritizing nodes having less candidate subpaths to nodes whose first symbol is 1. The time complexity remains unchanged though.

A fault-tolerant set-to-set disjoint-path routing algorithm in star graphs was given in [51]. In a G_n, given a set of non-faulty source nodes $S = \{s_1, s_2, \ldots, s_k\}$ a set of non-faulty destination nodes $D = \{d_1, d_2, \ldots, d_k\}$ ($k \leq n-1$) and a set F of at most $n-1-k$ faulty nodes, this algorithm finds k fault-free disjoint paths between each node of S and D. The main idea of the algorithm is to find k fault-free subgraphs G_{n-1}, each of them used to connect one node s_i to a node d_j using a shortest-path routing algorithm. By discussing the number of edges needed to reach selected subgraphs G_{n-1} and considering the diameter of a G_{n-1}, we obtain a maximum path length of $diam(G_n) + 5$. For one path, reaching a selected subgraph G_{n-1} requires $O(n)$, hence we obtain a total time complexity of $O(kn)$.

A k-pairwise disjoint-path routing algorithm in star graphs was presented in [54]. In a G_n, given $k = \lceil (n-1)/2 \rceil$ pairs of distinct nodes $(s_1, d_1), (s_2, d_2), \ldots, (s_k, d_k)$, this algorithm finds k disjoint paths $s_i \rightsquigarrow d_i$ $(1 \leq i \leq k)$. The main idea of the algorithm is to reduce the k-pairwise disjoint-path problem in a G_n to k shortest-path routings inside distinct subgraphs G_{n-1}. A subgraph is candidate to hosting a shortest-path routing if it contains one pair (s_i, d_j) or at most one source or destination node. At most three edges are required to connect a node s_i or d_j to its destination subgraph. Shortest-path routing inside a G_{n-1} requires at most $diam(G_{n-1})$ edges, therefore we have paths of length at most $diam(G_{n-1}) + 6 \leq diam(G_n) + 5$ edges. Connecting a node s_i or d_i to its destination subgraph requires $O(n)$ time. Shortest-path routing is also $O(n)$ time, hence constructing all the k paths is $O(kn)$ time complexity.

We now focus on cluster fault tolerant (CFT) routing algorithms inside star graphs. A CFT node-to-node routing algorithm in star graphs was introduced in [48]. In a G_n, given a set of at most $n-2$ faulty clusters of diameter at most two and two distinct non-faulty nodes s and d, this algorithm finds a fault-free path between $s = (p_1, \ldots, p_n)$ and d. The main idea of the algorithm is to route s and d to a subgraph G_{n-1} containing no center of a faulty cluster of diameter at most two (ie. node of degree greater than one, arbitrarily select any other node otherwise) so that we can apply a fault-tolerant node-to-node routing algorithm inside this G_{n-1}. Each path requires at most three edges to connect s to the destination subgraph G_{n-1} (same for d) and at most $diam(G_{n-1}) + 3$ edges to apply a fault-tolerant node-to-node algorithm inside G_{n-1}. Hence the maximum path length is $diam(G_{n-1}) + 9 \leq diam(G_n) + 8$. Finding the center of all faulty clusters, the destination subgraph G_{n-1} and a fault-free path to connect s and d to that subgraph takes $O(n)$ time. Hence the algorithm is $O(n)$ time complexity.

A CFT node-to-set disjoint-path routing algorithm inside a star graph was given in [56]. In a G_n, given one source node s, a set of destination nodes $D = \{d_1, d_2, \ldots, d_k\}$ $(2 \leq k \leq n-1)$ and at most $(n-1)-k$ faulty clusters of diameter at most two, this algorithm finds k fault-free disjoint paths between s and each node of D. The main idea of the algorithm is to connect s and each destination node to a distinct subgraph G_{n-1} not containing s nor the center of a faulty cluster, and then apply a CFT node-to-node routing algorithm inside this G_{n-1}. By discussing the number of edges required to connect s and each destination node to its destination subgraph as well as the routing inside each destination subgraph, we obtain a maximum path length of $diam(G_{n-1}) + 10 \leq diam(G_n) + 9$. It takes $O(n)$ time to find faulty cluster centers and $O(kn)$ to find k destination subgraphs. Routing a destination node to its destination subgraph takes $O(n)$ time and node-to-

node routing inside G_{n-1} takes $O(n)$ time. Hence the total time complexity of this algorithm is $O(kn)$.

A k-pairwise cluster fault-tolerant disjoint-path routing algorithm in star graphs was presented in [56]. In a G_n, given $2 \leq k \leq \lceil (n-1)/2 \rceil$ pairs $(s_1, d_1), \ldots, (s_k, d_k)$ of non-faulty nodes and at most $n - 2k$ faulty clusters of diameter at most two, this algorithm finds k disjoint fault-free paths connecting s_i to d_i $(1 \leq i \leq k)$. Again, the main idea of the algorithm is to find k destination subgraphs G_{n-1} not containing the center of a faulty cluster nor two nodes of different pairs to connect each pair using a node-to-node routing algorithm. s_i and d_j nodes are connected to their destination subgraph in at most three edges. Fault-tolerant node-to-node routing inside each destination subgraph requires at most $diam(G_{n-1})+3 \leq diam(G_n)+2$ edges. Hence the maximum path length is $diam(G_n) + 8$.

2.3.2 Macro-Star Networks

The macro-star (MS) interconnection network [119] is based on star graphs. Compared to a star graph of the same size, the degree of each node of a MS network is much lower while the diameter remains of the same order.

Each node of an MS(l, n) is a permutation of the set $\{1, 2, \ldots, k\}$ with $k = nl + 1$. Each node $(p_1, p_2, \ldots, p_n)$ of an MS(l, n) is adjacent to the n nodes $(p_i, p_2, p_3, \ldots, p_{i-1}, p_1, p_{i+1}, \ldots, p_k)$ $(2 \leq i \leq n+1)$ (this operation corresponds to the swap of the first symbol with one symbol of the first n-symbol portion) and to the $l - 1$ nodes $(p_1, p_{(i-1)n+2}, \ldots, p_{in+1}, p_{n+2}, \ldots, p_{(i-1)n+1}, p_2, \ldots, p_{n+1}, p_{in+2}, \ldots, p_k)$ $(2 \leq i \leq l)$ (this operation corresponds to the swap of the first n-symbol portion with another). Also, MS networks have a recursive structure: each node is included in a star graph G_{n+1} induced by the first adjacency condition above, and the second adjacency condition induces a subgraph $MS(l, n, p_j \ldots, p_k)$ $(j = (i-1)n + 2)$.

A node-to-node routing algorithm in an MS(l, n) was described in [119]. The main idea of the algorithm is to consider for each symbol of the node address into which n-symbol portion of the destination node it is located. The algorithm first groups each symbol into a portion so that each portion can be found at any place in the destination node. Then it reorders the portions each other. By discussing the number of edges required to place symbols into the correct portion and to reorder the portions, we obtain a path of length at most $\lfloor 5(nl + l - 1)/2 \rfloor$. As an improvement, we can try to eliminate the second step (portions reordering) by placing a portion to its final position as soon as it is complete. We would obtain in this case a path of length at most $2nl + \lfloor nl/2 \rfloor + 2(l - 1)$.

2.3.3 Pancake Graphs

Pancake graphs have been introduced by Akers et al. [3] as a particular Cayley graph. Similarly to star graphs, pancake graphs can connect many nodes while keeping a low degree and a small diameter.

Each node of a pancake graph P_n corresponds to a permutation of the set $\{1, 2, \ldots, n\}$. Each node $(p_1, p_2, \ldots, p_n)$ of a P_n is adjacent to the $n-1$ nodes $(p_i, p_{i-1}, \ldots, p_1, p_{i+1}, p_{i+2}, \ldots, p_n)$ $(2 \leq i \leq n)$, that is by reversing the i first (leftmost) symbols of the node. Also P_n is made of n subgraphs P_{n-1} induced by the nodes sharing the same last (rightmost) symbol.

A node-to-node routing algorithm in a P_n finding a path between any two nodes $s = (s_1, \ldots, s_n)$ and $d = (d_1, \ldots, d_n)$ was given in [72]. The main idea of the algorithm is to reverse the leftmost symbols of the current node (beginning with s) so that the right part of the node address is matching the corresponding right part of d. We start by routing s to the node whose rightmost symbol is equal to d_n. Formally, for each i from n to 1, if $s_i \neq d_i$ we first connect s to the node $s^{(k)} = (s_k = d_i, \ldots, s_1, s_{k+1}, \ldots, s_n)$ and then connect $s^{(k)}$ to the node $s^{(k,i)} = (s_i, s_{i-1}, \ldots, s_{k+1}, s_1, \ldots, s_k = d_i, s_{i+1}, \ldots, s_n)$ which becomes the new node s. If $s_i = d_i$ there is nothing to do, the symbol s_i is already correctly positioned. Finding k such that $s_k = d_i$ requires $O(n)$ time complexity. Hence by traversing all the n symbols, the total time complexity of this algorithm is $O(n^2)$. It generates a path of length $O(n)$. Gates and Papadimitriou gave an upper bound for the diameter of P_n: $diam(P_n) = 5(n+1)/3$ [40].

A node-to-node disjoint-path routing algorithm in a P_n was given in [106]. This algorithm finds $n-1$ internally disjoint paths between any two nodes s and d. The main idea of the algorithm is to recursively apply this algorithm onto the subgraphs P_{n-1} containing s or d, reached by traversing distinct subgraphs to ensure path disjointness. By discussing the number of edges required to reach the subgraph used to recursively apply this algorithm, we obtain path lengths of $O(n)$. Since at each step of the recursion a shortest-path routing is applied in the intermediary subcube for each path, we obtain a total time complexity $T(n)$ of $T(n-1) + O(n^3) = O(n^4)$.

Kaneko and Peng later described in [67] a node-to-node disjoint-path routing algorithm in a P_n being $O(n^2)$ time complexity. The main idea of this algorithm is to first route s to $n-1$ distinct subgraphs using disjoint paths of length at most two, and similarly to route d to $n-1$ distinct subgraphs. Then we extend these paths so that we can connect with a shortest-path routing each pair of nodes routed from s and d inside a distinct subgraph. We need at most four edges to route s or d, but not both, to their destination subgraph. Other extension paths are of length at most

two. Hence we have paths of length at most $diam(P_{n-1})+4+2 = 5n/3+6$. Extension paths are found in $O(n)$ time. Shortest-path routing inside a P_{n-1} is also $O(n)$ time. Therefore the total time complexity of this algorithm is $O(n^2)$.

A node-to-set disjoint-path routing algorithm in a pancake graph was described in [72]. In a P_n, given a source node $s = (1, \ldots, n)$ and a set of $n-1$ destination nodes $D = \{d_1, d_2, \ldots, d_{n-1}\}$, the algorithm finds $n-1$ disjoint paths $s \rightsquigarrow d_i$ $(1 \leq i \leq n-1)$. The main idea of the algorithm is to recursively solve the problem inside the subgraph P_{n-1} containing s and to achieve path disjointness by using class paths (i.e. paths whose nodes are inside the same class; each node is contained in one class only). A preprocessing task connects each destination node to a distinct class and then with a class path reach the subgraph of s before applying the algorithm recursively onto that subgraph. Because classes are rings of length $2n$, the sum of the path lengths is $l(n) = l(n-1) + O(n^2) = O(n^3)$. Connecting destination nodes to distinct classes requires $O(n^4)$ time. Hence the total time complexity is $T(n) = T(n-1) + O(n^4) = O(n^5)$.

Another node-to-set disjoint-path routing algorithm in a pancake graph was given in [67], reducing the time complexity to $O(n^2)$. This time the main idea of this algorithm is to distribute destination nodes into distinct subgraphs which will be later used to perform a shortest-path routing. The algorithm first connects s to $n-1$ distinct subgraphs with disjoint paths of length at most two, then connects each destination node to these $n-1$ subgraphs with a disjoint path of length at most three and finally apply a shortest-path routing to complete the paths. Because a path may need to be rerouted to another subgraph, we obtain a maximum path length of $diam(P_{n-1}) + 2 + 4 \leq 5n/3 + 6$. Distributing destination nodes to distinct subgraphs requires $O(n^2)$ time complexity. Shortest-path routing inside each subgraph requires $O(n)$ time, hence the total time complexity of this algorithm is $O(n^2)$.

A set-to-set disjoint-path routing algorithm in a pancake graph was given in [96]. In a P_n, given a set of $n-1$ source nodes S and a set of $n-1$ destination nodes D, this algorithm finds $n-1$ disjoint paths between all elements of S and D. The main idea of this algorithm is to first distribute source and destination nodes to $n-1$ distinct subgraphs such that each subgraph has exactly one pair of distributed nodes in S and D. A shortest-path routing is then performed to connect each pair. Since each pair is connected inside a distinct subgraph, path disjointness is guaranteed. By discussing the number of edges required to distribute source and destination nodes, we obtain paths of length at most $5n/3+O(1)$. Node distribution and shortest-path routing are performed in $O(n^2)$ time, which is the dominant

time complexity of the algorithm.

2.3.4 Burnt Pancake Graphs

In this section we shall describe another permutation-based interconnection network: burnt pancake graphs. Burnt pancake graphs have been originally described in [40]. They can connect still more nodes than a pancake graph or a star graph of the same size, while keeping their degree and diameter of the same order.

Each node of a burnt pancake graph BP_n corresponds to a signed permutation of the set $\{1, 2, \ldots, n\}$. Each node $(p_1, p_2, \ldots, p_n)$ of a BP_n is adjacent to the n nodes $(\bar{p}_i, \bar{p}_{i-1}, \ldots, \bar{p}_1, p_{i+1}, p_{i+2}, \ldots, p_n)$ $(1 \leq i \leq n)$, that is by reversing as well as negating the i first (leftmost) symbols of the node. Also a BP_n is made of $2n$ disjoint subgraphs BP_{n-1} induced by the nodes sharing the same last (rightmost) symbol.

A simple node-to-node routing algorithm in a BP_n finding a path between any two nodes $s = (s_1, \ldots, s_n)$ and $d = (d_1, \ldots, d_n)$ was given in [70]. The main idea of the algorithm is to reverse the leftmost symbols of the current node (beginning with s) so that the right part of the node address is matching the corresponding right part of d. We start by routing s to the node whose rightmost symbol is equal to d_n. Let $c = (c_1, \ldots, c_n)$ be s. Formally, for each i from n to 1, if $c_i \neq d_i$ we find k such that $|c_k| = |d_i|$. If $k \neq 1$ then we append the node $c^{(k)} = (\bar{c}_k, \ldots, \bar{c}_1, c_{k+1}, \ldots, c_n)$ to the path; this node becomes the new node c. If $i \neq 1$ and $c_1 = d_i$ then we reverse the sign of c_1 $(= d_i)$ by appending the node $c^{(1)}$ to the path; this node becomes the new node c. Finally, we connect c to the node $c^{(i)}$ to correctly place the symbol c_1 in the right part of the node address; this node becomes the new node c. This way, the rightmost $n - i + 1$ symbols of c are matching the rightmost $n-i+1$ symbols of d. If $c_i = d_i$ there is nothing to do, the symbol c_i is already correctly positioned. Finding k such that $|c_k| = |d_i|$ requires $O(n)$ time complexity. Hence by traversing all the n symbols, the total time complexity of this algorithm is $O(n^2)$. Also, this algorithm generates a path of length at most $3n - 2$ since at most three prefix reversal operations are performed for each iteration.

Another node-to-node routing algorithm in a BP_n was described in [16]. Because of the symmetric structure of burnt pancake graphs, this node-to-node routing problem is equivalent to sorting the permutation, that is to route from any node $s = (s_1, \ldots, s_n)$ to the identity permutation $(1, 2, \ldots, n)$. The main idea of the algorithm is to always keep already sorted consecutive symbols together. Once a sequence of two symbols is obtained, we merge these two symbols into one, thus reducing the number of symbols by one. The algorithm is finished when no symbol remains. This time the

maximum path length is reduced to $2n$ while the time complexity remains $O(n^2)$.

A node-to-node disjoint-path routing algorithm in a burnt pancake graph was described in [70]. In a BP_n, given a source node s and a destination node d, this algorithm finds n internally disjoint paths between s and d. The main idea idea of this algorithm is for each path to perform a shortest-path routing inside a distinct subgraph which will ensure path disjointness: s is connected to $n-1$ distinct subgraphs with at most two edges and d is disjointly connected to these subgraphs in at most three edges before applying a shortest-path routing onto each of these subgraphs. By discussing special cases, we obtain a maximum path length of $3n+4$ and a time complexity of $O(n^3)$.

A node-to-set disjoint-path routing algorithm in a burnt pancake graph was given in [63]. In a BP_n, given a source node s and a set of n destination nodes D, this algorithm finds n disjoint paths between s and all nodes in D. The main idea idea is to ensure path disjointness by using class paths, that is paths including exclusively nodes of the same class (each node is in one class only). Finally we solve the routing problem inside the subgraph containing s. A preprocessing task connects each destination node to a distinct class in $O(n^4)$ time, and then routes to the subgraph of s by traversing that class. Since a class is a ring of length $4n$, the sum of the paths lengths is $l(n) = l(n-1)+O(n^2) = O(n^3)$. The total time complexity is $T(n) = T(n-1) + O(n^4) = O(n^5)$.

A set-to-set disjoint-path routing algorithm in a burnt pancake graph was proposed in [61]. In a BP_n, given a set of n source nodes S and a set of n destination nodes D, this algorithm finds n disjoint paths between the source and destination nodes. The main idea of this algorithm is to first distribute source nodes to distinct subgraphs with disjoint paths of length at most two (source nodes whose subgraph does not contain any other source node act as distributed source nodes themselves). A similar distribution process is then applied to the destination nodes. Pairs of distributed source and destination nodes inside the same subgraphs are connected. Lastly, remaining unconnected distributed source nodes are connected to subgraphs containing remaining unconnected distributed destination nodes where these last pairs are finally connected. This ultimate step may obviously generate a longest path, precisely of length at most $2n+12$ and in $O(n^4)$ time complexity.

A fault-tolerant node-to-node routing algorithm inside a burnt pancake graph was given in [60]. In a BP_n, given any two non-faulty nodes s and d and a set of at most $n-1$ faulty nodes, this algorithm finds a fault-free

path connecting s to d. The main idea of the algorithm is to first connect s and d to distinct subgraphs such that each pair of end-nodes of a path for s and d is located inside a distinct subgraph. s is connected to n distinct subgraphs with disjoint paths of length at most two, and d is connected to these n subgraphs with disjoint paths of length at most three. Finally we apply onto a fault-free subgraph a shortest-path routing to obtain a fault-free path connecting s to d. By discussing in special cases the number of edges required before applying a shortest-path routing, we obtain a maximum path length of $2n + 4$. Finding a fault-free subgraph takes $O(n^2)$ time complexity which is the dominant time complexity of the algorithm.

A cluster fault-tolerant node-to-node routing algorithm was given in [62]. In a BP_n, given any two non-faulty nodes s and d and a set of $n - 1$ faulty clusters of diameter at most three, this algorithm finds a fault-free path connecting s to d. The main idea is to first connect s and d to a subgraph not containing any center of a faulty cluster and containing at most $n - 2$ of their neighbours, and second to apply a fault-tolerant node-to-node routing algorithm inside this subgraph to complete the path. Since the paths connecting s and d to the destination subgraph are of length at most four, we obtain a maximum path length of $2(n - 1) + 4 + 4 + 4 = 2n + 10$. Computing the center of one faulty cluster of diameter at most three requires $O(n)$ time complexity. Hence computing the centers of all faulty clusters requires $O(n^2)$ time complexity, which is the dominant time complexity of this algorithm.

2.3.5 Rotator Graphs

We focus in this section on another interconnection network based on permutations of symbols, rotator graphs, introduced by Faber and Moore in [31] and rediscovered by Corbett [17]. Similarly to star graphs and pancake graphs, rotator graphs have the useful ability to connect many nodes while retaining a low degree and small diameter. However rotator graphs have a smaller diameter compared to star graphs and pancake graphs of the same size.

Each node of a rotator graph R_n is a permutation of the n symbols $1, 2, \ldots, n$. Each node $(p_1, p_2, \ldots, p_n)$ of an R_n is adjacent to the $n - 1$ nodes $(p_2, p_3, \ldots, p_i, p_1, p_{i+1}, \ldots, p_n)$ $(2 \leq i \leq n)$. Also, an R_n has a recursive structure, it is composed of n distinct rotator subgraphs R_{n-1}, each of them induced by the nodes of R_n sharing the same last symbol p_n.

A node-to-set disjoint-path routing algorithm in an n-rotator graph R_n finding $n - 1$ disjoint paths was described in [71]. The main idea of the algorithm is to recursively apply the algorithm inside a subgraph R_{n-1}, each time creating one path to decrease the number of destination nodes

before inducting on that subgraph R_{n-1}. Path disjointness is ensured by traversing one distinct class for each path (each node is member of an unique class). A preprocessing task is required to connect each destination node to a distinct class. Hence a path traversing exclusively nodes of a same class is guaranteed to be disjoint from paths traversing different classes. Regarding the maximal path length, since a class is a ring of length n, class traversal using shortest-path routing requires $O(n)$ edges. Because class traversal can be performed for each path at each step of the reduction, we obtain a sum of the path lengths of $O(n^3)$. Assuming the comparison of two node addresses takes $O(n)$ time complexity, each step of the recursion takes $O(n^4)$ total time complexity, dominated by destination node connection to distinct classes. Hence applying the algorithm recursively $O(n)$ times gives a total time complexity of $O(n^5)$.

2.3.6 Bi-rotator Graphs

We focus in this section on the bi-rotator interconnection network. Originally introduced by Lin et al. in [84], bi-rotator graphs are the undirected variation of rotator graphs. Sharing the same interesting properties with rotator graphs (eg. connecting many nodes with a low degree), bi-rotator graphs, thanks to the bi-directional edges, benefit from a smaller average routing distance [84].

Each node of a bi-rotator graph BR_n is a permutation of the n symbols $1, 2, \ldots, n$. Each node $(p_1, p_2, \ldots, p_n)$ of a BR_n is adjacent to the $n-1$ nodes $(p_2, p_3, \ldots, p_i, p_1, p_{i+1}, \ldots, p_n)$ $(2 \leq i \leq n)$ and to the $n-1$ nodes $(p_i, p_1, p_2, \ldots, p_{i-1}, p_{i+1}, \ldots, p_n)$ $(2 \leq i \leq n)$. Also, a BR_n has a recursive structure, it is composed of n distinct bi-rotator subgraphs BR_{n-1}, each of them induced by the nodes of BR_n sharing the same last symbol p_n (rightmost position).

A node-to-node disjoint-path routing algorithm finding $2n-3$ disjoint paths inside a bi-rotator graph BR_n was described in [64]. The main idea of this algorithm to ensure path disjointness is for each two paths to reach distinct subgraphs BR_{n-1} from the source and destination node, that is each such subcube containing two pairs to be linked. There may exist two subcubes containing one pair and a half each, hence additional routing is required to obtain two full pairs in one of them. To connect one pair, shortest-path routing in a BR_{n-1} uses at most $n-1$ edges. Connecting two pairs pairs requires at most $2n-4$ edges, and in the case two subcubes contain one pair and a half each, finding the three disjoint paths requires at most $4n-11$ edges. The latter case generates longest paths, with at most three edges to reach the distinct subcube from the source node and as many from the destination node, we have paths of length at most $4n-11+6 = 4n-5$. Regarding time complexity, pair connection algorithms are all $O(n^2)$. Hence

the total time complexity of this algorithm is $O(n^3)$ to generate all the $2n-3$ paths.

A node-to-set disjoint-path routing algorithm in BR_n finding $2n-3$ disjoint paths was given in [65]. The main idea of the algorithm is to recursively apply the algorithm onto subgraphs BR_{n-1}, each time creating at least one path. Also, path disjointness is achieved by exclusively traversing one distinct class for each path (each node is member of an unique class). A preprocessing task connects each destination node to a node inside a distinct class. Regarding the sum of the path lengths $l(n)$, since shortest-path routing in a BR_n generates a path of $O(n^2)$ length, we have $l(n) = l(n-1) + O(n^2) = O(n^3)$. Routing each destination node to a distinct class requires $O(n^4)$ time complexity. Hence, because of the recursion, the total time complexity $T(n)$ for this algorithm is $T(n) = T(n-1)+O(n^4) = O(n^5)$.

2.3.7 Bubble-sort Graphs

We focus in this section on bubble-sort graphs, originally introduced by Akers et al. [3]. Similarly to star graphs, bubble-sort graphs are Cayley graphs with a recursive structure. They can connect many nodes while retaining a low degree.

Each node of a bubble-sort graph B_n is a permutation of the n symbols $1, 2, \ldots, n$. Each node $(p_1, p_2, \ldots, p_n)$ of a B_n is adjacent to the $n-1$ nodes $(p_1, p_2, \ldots, p_{i-1}, p_{i+1}, p_i, p_{i+2}, \ldots, p_n)$ $(1 \leq i \leq n-1)$. Also, a B_n has a recursive structure, it is composed of n distinct subgraphs B_{n-1}, each of them induced by the nodes of B_n sharing the same last symbol p_n (rightmost position).

A node-to-node disjoint-path routing algorithm in a bubble-sort graph was described in [73]. In a B_n, the main idea of this algorithm to achieve paths disjointness is to first route the source and destination node to distinct subgraphs B_{n-1} ($(n-1)$-dimensional bubble-sort graphs, subgraphs of B_n) to then apply a shortest-path routing algorithm inside these distinct subgraphs. Since reaching a distinct subgraph requires $O(n)$ edges and since shortest-path routing inside a B_{n-1} requires $O(n^2)$ edges, the maximum path length of this algorithm is $O(n^2)$. For the same reason, since shortest-path routing in a B_{n-1} is $O(n^3)$ time complexity, generating all the n paths requires $O(n^4)$ time complexity.

A node-to-set disjoint-path routing algorithm in bubble-sort graph was described in [107]. In a B_n, the main idea of the algorithm is to recursively solve the node-to-set disjoint-path routing problem into subgraphs B_{n-1} and to disjointly connect each destination node by traversing one dis-

tinct class per path (each node is member of an unique class). A preprocessing task is thus required to connect each destination node to a distinct class. The sums of the path lengths when traversing the classes are at most $(n-1)^2$ since a class is a ring of length $n-1$. Because a shortest-path routing algorithm in a B_n outputs a path of length at most $n(n-1)/2$, we can deduce the sum of the path lengths $l(n)$ is equal to $l(n-1) + (n-1)^2 + (n(n-1))/2 = l(n-1) + (3n-2)(n-1)/2$. Also, as shortest-path routing in a B_n is $O(n^3)$ time complexity, generating $O(n)$ class paths requires $O(n^4)$ time complexity. Since the algorithm may be recursively applied onto a B_{n-1}, we can express the total time complexity $T(n)$ as $T(n-1) + O(n^4) = O(n^5)$.

To conclude this section on permutation-based interconnection network, we refer two additional graphs: well-known de Bruijn networks [23, 4, 85] and Incomplete Pancake Graphs [66].

2.4 Meshes, Tori-based interconnection networks

We can distinguish another category of networks including meshes and tori-based interconnection networks. Because of their orthogonality [25], these networks are popular topologies as routing and hardware implementation are much facilitated compared for example to permutation-based interconnection networks.

An n-dimensional mesh interconnection network is made of $k_0 \times k_1 \times \ldots \times k_{n-1}$ nodes, that is k_i nodes along each dimension i with $k_i \geq 2$ and $0 \leq i \leq n-1$. A node u of an n-dimensional mesh is identified by a tuple of n coordinates $(u_0, u_1, \ldots, u_{n-1})$ with $0 \leq u_i < k_i, 0 \leq i \leq n-1$. Regarding adjacency, two nodes $u = (u_0, u_1, \ldots, u_{n-1})$ and $v = (v_0, v_1, \ldots, v_{n-1})$ are neighbours if and only if there exists a j $(0 \leq j \leq n-1)$ such that $\forall i, 0 \leq i \leq n-1, i \neq j, u_i = v_i$ and $u_j = v_j + 1$ or $u_j = v_j - 1$ with respect to $0 \leq u_j < k_j$. Hence we understand that for each node u of an n-dimensional mesh, $n \leq d(u) \leq 2n$ holds.

Closely related to meshes, tori [20] are built so that each node has the same degree, making it a regular and symmetrical network. Concretely, each dimension will contain the same number of nodes k (instead of k_i for meshes) and the adjacency condition of two nodes becomes there exists a j $(0 \leq j \leq n-1)$ such that $\forall i, 0 \leq i \leq n-1, i \neq j, u_i = v_i$ and $u_j = (v_j + 1) \bmod k$ or $u_j = (v_j - 1) \bmod k$. Such torus is formally called a k-ary n-cube.

Because each node of a mesh or torus has a fixed degree independent from the arity k (or k_i) of the network itself (e.g. a k-ary n-cube is of degree $2n$), these graphs (and their siblings) are rarely considered when dealing with disjoint-path routing. Effectively, the maximum number of disjoint

paths that can be found in such graphs is always limited by its number of dimensions, whatever the size of the network itself. For example, in any k-ary 2-cube, we can only find at most four internally disjoint paths between any two nodes, whatever the value of k, that is the size of the torus.

Several other networks are based on meshes or tori: Multi-Mesh [22], Manhattan Street Network [89, 112], Multidimensional Manhattan Street Network [15], Hierarchical Manhattan Street Network [93], Recursive Diagonal Torus [118], Express Cubes [21].

2.5 Other networks

There exist several other interconnection networks not falling under one of the previously mentioned categories. These graphs are often hierarchical: they are connecting subgraphs according to a specific topology. Effectively, a hierarchical structure allows a network to bind many nodes while retaining the degree and the diameter of the graph low. For example, the WK-recursive Network [113, 12, 26] and its variant the Incomplete WK-recursive Network [103, 104] are recursively connecting fully connected graphs (complete graphs).

Also, interestingly some of these hierarchical graphs are generic: they are using any seed network as base, then recursively used to produce a more complex structure. Such networks include the Recursive Dual Net [82] and the Generalized Hierarchical Completely-Connected Network [108].

Chapter 3

Definitions

In this chapter we formally define the interconnection networks used hereinafter, as well as introducing useful lemmas and algorithms. We first focus on the hypercube interconnection network [100]. Then we consider the perfect hierarchical hypercube interconnection network [86]. We conclude this chapter with the metacube interconnection network [83].

3.1 Hypercube

The hypercube (HC) network is extensively used throughout our work since we are focusing on hypercube-based networks. Hence we first formally define the hypercube interconnection network and give related notations omnipresent inside this thesis.

Each node of an n-dimensional hypercube Q_n has an address made of n bits. Thus a Q_n contains a total of 2^n nodes. Two nodes are adjacent if and only if their addresses differ in one single bit, that is their Hamming distance is equal to one. Hence we can deduce the diameter, degree and connectivity of a Q_n as being simply n. Also it is easy to see that a Q_n is symmetric. A four dimensional hypercube is represented in Figure 3.1.

As explained briefly in Chapter 2, a Q_n is a recursive topology: a Q_n is made of two hypercubes of lower dimension Q^0_{n-1} and Q^1_{n-1} (also called subcubes), where for a dimension i ($0 \leq i \leq n-1$), Q^0_{n-1} is induced by all the nodes of Q_n whose i-th bit is set to 0, and Q^1_{n-1} is induced by all the nodes of Q_n whose i-th bit is set to 1. We say that Q_n is *reduced* to two subcubes Q^0_{n-1} and Q^1_{n-1}. See Figure 3.2.

For convenience reasons, we formally name the n neighbours of a node u in a Q_n as $u^{(i)}$ ($0 \leq i \leq n-1$). The node $u^{(i)}$ is the unique neighbour of u differing with u on the i-th bit position. Also, let $u^{(i,j)} = (u^{(i)})^{(j)}$. As an example, in a Q_3, the node $u = 000$ is adjacent to the three nodes $u^{(0)} = 001$, $u^{(1)} = 010$ and $u^{(2)} = 100$.

Inside a Q_n, we assume that a node address can be stored in a fixed

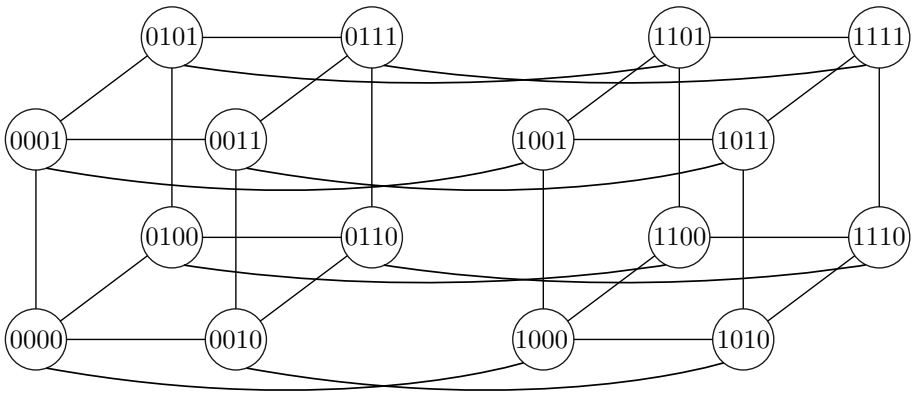

Figure 3.1: A four dimensional hypercube Q_4.

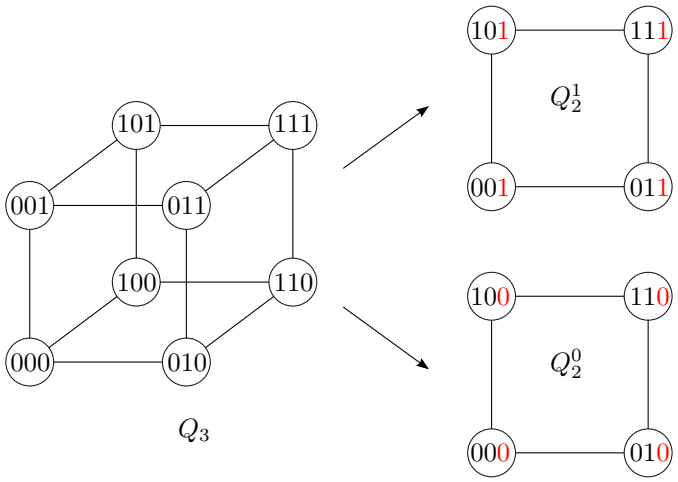

Figure 3.2: Reduction of a hypercube Q_3 along the 0-th dimension.

number of machine words. Hence, for two nodes u and v in a Q_n, the comparison of u and v, the computation of the Hamming distance $H(u, v)$, and the calculation of the most significant bit position can be performed in constant time complexity $O(1)$.

A shortest-path routing algorithm in a hypercube is denoted by SPR. Many different SPR algorithms exist. For example, one of them is called dimension-order shortest-path routing algorithm. This particular SPR finds a shortest path between any two nodes u and v in a Q_n by successively flipping the bits at the positions $\delta_0, \delta_1, \ldots, \delta_{h-1}$ where $h = H(u, v)$ and $u \oplus v = \sum_{i=0}^{h-1} 2^{\delta_i}$.

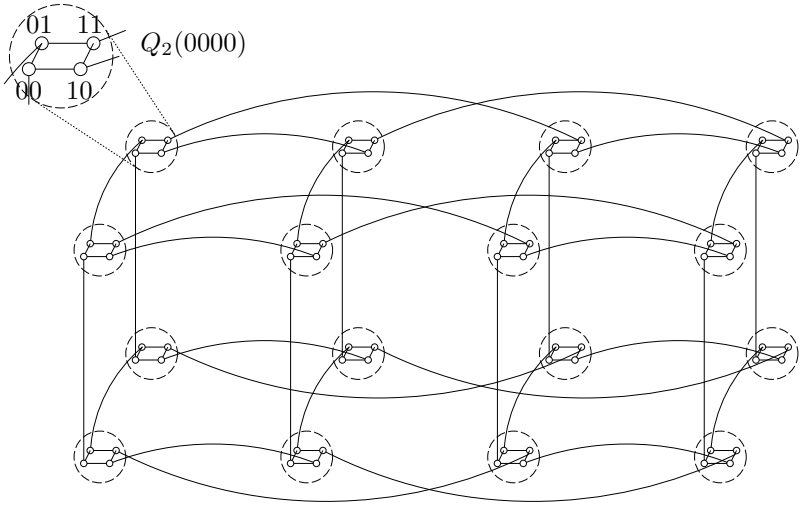

Figure 3.3: HHC_6 ($m = 2$).

3.2 Perfect hierarchical hypercube

We now formally define the perfect hierarchical hypercube (HHC) interconnection network. A $(2^m + m)$-dimensional (i.e. perfect) hierarchical hypercube is denoted by HHC_{2^m+m}. Each node of an HHC_{2^m+m} has an address represented by a pair of a 2^m-bit sequence, called the subcube ID, and an m-bit sequence, called the processor ID. Hence an HHC_{2^m+m} contains 2^{2^m+m} nodes. Two nodes $\boldsymbol{u} = (\sigma_u, \pi_u)$ and $\boldsymbol{v} = (\sigma_v, \pi_v)$ are adjacent if and only if one of the following conditions holds:

- $\sigma_u = \sigma_v$ and $H(\pi_u, \pi_v) = 1$
- $\sigma_u = \sigma_v \oplus 2^{\pi_v}$ and $\pi_u = \pi_v$

Edges induced by the first condition are called internal edges, whereas the edges induced by the second condition are called external edges. From this we can deduce the degree and the connectivity of an HHC_{2^m+m} are both equal to $m + 1$. Again, an HHC_{2^m+m} is symmetric and of diameter 2^{m+1} [116]. Also we note that the nodes having the same subcube ID σ induce an m-dimensional hypercube, denoted subcube $Q_m(\sigma)$.

For example, in an HHC_6 ($m = 2$), the node $\boldsymbol{u} = (0000, 00)$ is adjacent to the three nodes $(0000, 01)$, $(0000, 10)$ (both linked to $\boldsymbol{u}$ with an internal edge) and $(0001, 00)$ (linked to $\boldsymbol{u}$ with an external edge). An HHC_6 is illustrated by Figure 3.3.

An HHC_{2^m+m} has a two-level structure: on the lower level are subcubes, induced by internal edges, and on the higher level is a 2^m-dimensional hypercube Q_{2^m}, induced by external edges. Each node of Q_{2^m} corresponds to a distinct subcube of HHC_{2^m+m}. Practically, Q_{2^m} is obtained by mapping

each subcube $Q_m(\sigma)$ of HHC_{2^m+m} to the single node σ. Hence we consider inside an HHC_{2^m+m} two types of nodes:

- HHC-level nodes are nodes of the HHC_{2^m+m}.
- Cube-level nodes are nodes of the Q_{2^m}; they are subcube IDs.

From there, an HHC-level path denotes a path made of HHC-level nodes, and a cube-level path denotes a path made of cube-level nodes.

We now describe an algorithm CONV which converts a cube-level path in Q_{2^m} to an HHC-level path in HHC_{2^m+m}.

Given a cube-level path $P : \sigma_0 \rightsquigarrow \sigma_n$ and two processor IDs π_{beg}, π_{end}, CONV generates an HHC-level path $(\sigma_0, \pi_{beg}) \rightsquigarrow (\sigma_n, \pi_{end})$ according to P. For a path $P : \sigma_0 \rightarrow \sigma_1 \rightarrow \cdots \rightarrow \sigma_n$, let π_j $(1 \leq j \leq n)$ be an m-bit sequence satisfying the condition $\sigma_{j-1} \oplus 2^{\pi_j} = \sigma_j$. The corresponding HHC-level path $(\sigma_0, \pi_{beg}) \rightsquigarrow (\sigma_0, \pi_1) \rightarrow (\sigma_1, \pi_1) = (\sigma_0 \oplus 2^{\pi_1}, \pi_1) \rightsquigarrow (\sigma_1, \pi_2) \rightarrow \cdots \rightarrow (\sigma_{n-1}, \pi_{n-1}) \rightsquigarrow (\sigma_{n-1}, \pi_n) \rightarrow (\sigma_n, \pi_n) \rightsquigarrow (\sigma_n, \pi_{end})$ is constructed by using a shortest-path routing algorithm inside every subcube to connect each node (σ_j, π_j) to the node (σ_j, π_{j+1}). The pseudocode of CONV is given in Algorithm 1.

Regarding time complexity, in an HHC_{2^m+m}, for any cube-level path P in Q_{2^m} of length l, CONV applies a shortest-path routing algorithm inside each visited subcube, thus requiring $O(lm)$ time complexity.

Algorithm 1 CONV($P = \sigma_0 \rightarrow \sigma_1 \rightarrow \ldots \rightarrow \sigma_n$, π_{beg}, π_{end})

Input: A cube-level path $P = \sigma_0 \rightsquigarrow \sigma_n$ and two processors IDs specifying the HHC end-nodes (σ_0, π_{beg}) and (σ_n, π_{end}).
Output: The HHC-level path $(\sigma_0, \pi_{beg}) \rightsquigarrow (\sigma_n, \pi_{end})$ corresponding to P.

```
if L(P) = 0 then
    π_0(= π_beg) → π_1 → ... → π_λ(= π_end) := SPR(π_beg, π_end);
    return (σ_0, π_0) → (σ_0, π_1) → ... → (σ_0, π_λ)
else
    π_next := log_2(σ_0 ⊕ σ_1);
    π_0(= π_beg) → π_1 → ... → π_λ'(= π_next) := SPR(π_beg, π_next);
    P' := CONV(σ_1 ⇝ σ_n, π_next, π_end);
    return (σ_0, π_0) → (σ_0, π_1) → ... → (σ_0, π_λ') → P'
end if
```

3.3 Metacube

As briefly explained in Chapter 2, the metacube (MC) interconnection network has a two-level cubic structure: on the low level, the metacube is made of *clusters* linking nodes each other using a hypercube structure. Each cluster is member of a *class*. On the high level, nodes of different classes are connected each other, again using a hypercube structure.

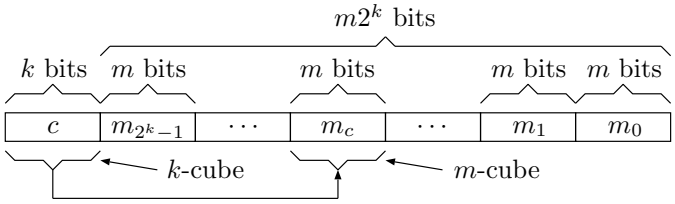

Figure 3.4: $MC(k, m)$ node address format.

Formally, each node address of a metacube $MC(k, m)$ is made of $k + m2^k$ bits; an $MC(k, m)$ has thus a total of 2^{k+m2^k} nodes. A node $\boldsymbol{u}$ in an $MC(k, m)$ is represented by a tuple $(c, m_{2^k-1}, \ldots, m_1, m_0)$ where c represents the class of $\boldsymbol{u}$ (classID). The class c occupies k bits and induces a high-level k-dimensional hypercube. Hence an $MC(k, m)$ has 2^k classes, represented by the leftmost k bits of node addresses.

Each m_i $(0 \leq i \leq 2^k - 1)$ is an m-bit sequence. The tuple $(m_{2^k-1}, \ldots, m_{c+1}, m_{c-1}, \ldots, m_1, m_0)$ represents the $m(2^k - 1)$-bit clusterID of a node. Finally, m_c is the m-bit nodeID inducing an m-dimensional hypercube (cluster). Hence each class contains $2^{m(2^k-1)}$ clusters, and each cluster contains 2^m nodes. See Figure 3.4 for an illustration of a node address in an $MC(k, m)$.

For a node $\boldsymbol{u}$ of an $MC(k, m)$, let $m_i(\boldsymbol{u})$ $(0 \leq i < 2^k)$ be the bit sequence m_i of $\boldsymbol{u}$ and let $c(\boldsymbol{u})$ be the class of $\boldsymbol{u}$, $\bar{c}(\boldsymbol{u})$ representing the $m_{2^k-1}, \ldots, m_1, m_0$ part of the address of $\boldsymbol{u}$, called k-cube of $\boldsymbol{u}$. The address of $\boldsymbol{u}$ can thus also be written $(c(\boldsymbol{u}), \bar{c}(\boldsymbol{u}))$. Two nodes $\boldsymbol{u}$ and $\boldsymbol{v}$ of an $MC(k, m)$ are adjacent if $H(\boldsymbol{u}, \boldsymbol{v}) = 1$ and one of the two following conditions is satisfied:

- $H(m_c(\boldsymbol{u}), m_c(\boldsymbol{v})) = 1$ where $c = c(\boldsymbol{u}) = c(\boldsymbol{v})$
- $H(c(\boldsymbol{u}), c(\boldsymbol{v})) = 1$

In other words, two nodes are adjacent either if they are inside the same cluster and their nodeID differ in one bit, or if they are inside distinct clusters and their addresses differ in only one bit, the differing bit necessarily located on the classID. The first condition induces edges inside a cluster; such edges are called cubes-edges. The second condition induces edges between two clusters; such edges are called cross-edges.

Hence we understand that the degree of each node in a $MC(k, m)$ is equal to $m + k$. We also note that a metacube of the form $MC(0, m)$ is an m-dimensional hypercube, and a metacube of the form $MC(1, m)$ corresponds to a dual-cube F_m as seen in Chapter 2. An $MC(1, 2)$ is given in Figure 3.5. In this figure the clusterID is displayed in the middle of each cluster.

For convenience reasons we formally name the neighbours of a node $\boldsymbol{u}$ of a $MC(k, m)$. Let $\boldsymbol{u}^{(i)}$ $(0 \leq i \leq m - 1)$ be the i-th dimensional neighbour of

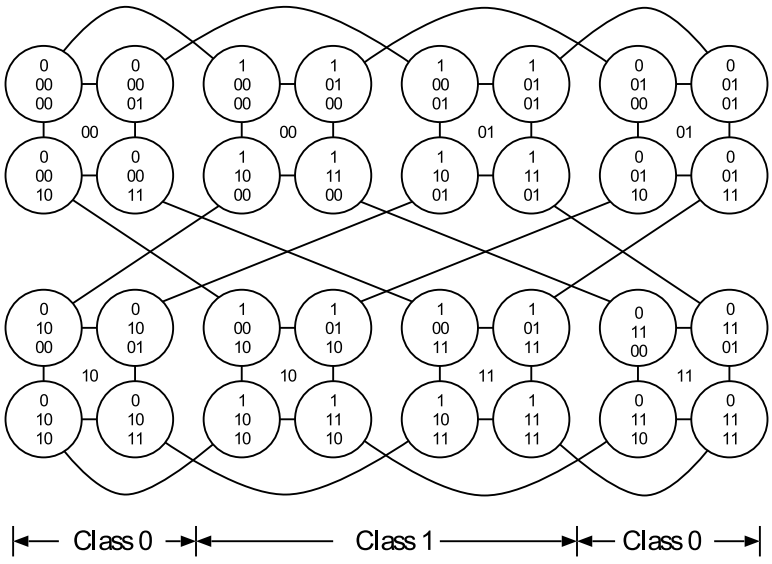

Figure 3.5: A metacube $MC(1,2)$.

$\boldsymbol{u}$ inside its cluster such that $\boldsymbol{u}$ and $\boldsymbol{u}^{(i)}$ differ on the i-th bit of $m_{c(\boldsymbol{u})}$. We assume the LSB (rightmost) of $m_{c(\boldsymbol{u})}$ corresponds to the 0-th dimension. Let $\boldsymbol{u}^{(i)}$ $(m \leq i \leq k+m-1)$ be the i-th dimension neighbour of $\boldsymbol{u}$ such that $\boldsymbol{u}$ and $\boldsymbol{u}^{(i)}$ differ in the $(i-m)$-th bit of the classID. Finally let $\boldsymbol{u}^{(i,j)} = (\boldsymbol{u}^{(i)})^{(j)}$ $(0 \leq i, j \leq k+m-1)$.

As an example, in a $MC(2,2)$, the node $\boldsymbol{u} = (01, 00, 00, 00, 00)$ is adjacent to the four nodes $\boldsymbol{u}^{(0)} = (00, 00, 00, 01, 00)$, $\boldsymbol{u}^{(1)} = (00, 00, 00, 10, 00)$ (both inside the same cluster as $\boldsymbol{u}$), $\boldsymbol{u}^{(2)} = (00, 00, 00, 00, 00)$ and $\boldsymbol{u}^{(3)} = (11, 00, 00, 00, 00)$. An $MC(2,2)$ is shown in Figure 3.6, regrouping clusters of the same class. Note that only four high-level cubes are fully represented in this figure.

A simple node-to-node routing algorithm in a metacube was described in [80]. Inside a metacube $MC(k,m)$, given any two nodes $\boldsymbol{s}$ and $\boldsymbol{d}$, this algorithm finds a path between $\boldsymbol{s}$ and $\boldsymbol{d}$. The main idea of the algorithm is to follow an Hamiltonian path of the classes from the class $c(\boldsymbol{s})$ of $\boldsymbol{s}$ to the class $c(\boldsymbol{d})$ of $\boldsymbol{d}$. Then we modify inside each cluster visited the m-bits according to that bits in $\boldsymbol{d}$ by applying a shortest-path routing in an m-dimensional hypercube. The pseudocode of this algorithm is given in Algorithm 2

From this approach, we understand that the distance between $\boldsymbol{s}$ and $\boldsymbol{d}$ can be expressed as first, 2^k cross-edges to perform the Hamiltonian path on a k-cube to browse all the classes, plus second, the number of edges corresponding to the Hamming distance between $\bar{c}(\boldsymbol{s})$ and $\bar{c}(\boldsymbol{d})$. Formally the distance between $\boldsymbol{s}$ and $\boldsymbol{d}$ is bound by $2^k + H(\bar{c}(\boldsymbol{s}), \bar{c}(\boldsymbol{d}))$. We can subsequently deduce the diameter of a metacube $MC(k,m)$ as being $2^k +$

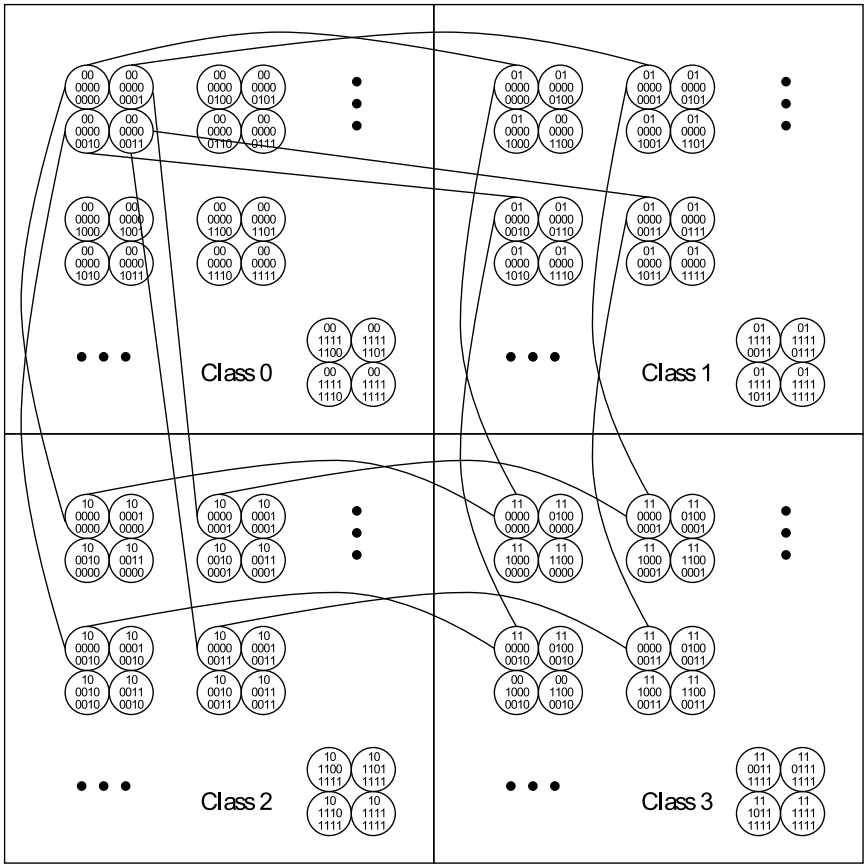

Figure 3.6: A metacube $MC(2, 2)$.

$m2^k = 2^k(m+1)$.

Algorithm 2 MC-N2N($MC(k,m)$, $\boldsymbol{s}$, $\boldsymbol{d}$)

Input: A metacube $MC(k,m)$, a source node $\boldsymbol{s}$ and a destination node $\boldsymbol{d}$.
Output: A path inside $MC(k,m)$ between $\boldsymbol{s}$ and $\boldsymbol{d}$.

```
/* next(c) represents the next class after c in the Hamiltonian path of the classes. */
/* Let h = 2^k. */
c := c(s);
v := s;
P := v;
while 1 do
    w = (c, m_{h-1}(v), ..., m_{c+1}(v), m_c(d), m_{c-1}(v), ..., m_0(v));
    if w ≠ v then
        P := P ~>^SPR w
    end if
    if w = d then
        break
    end if
    v := w;
    w := (next(c), m_{h-1}(v), ..., m_{c+1}(v), m_c(v), m_{c-1}(v), ..., m_0(v));
    P := P → w;
    c := next(c)
end while
```

Chapter 4

Optimal node-to-set disjoint-path routing in hypercubes

The routing algorithms we describe in the next chapters strongly rely on hypercube node-to-set disjoint-path routing. Subsequently we propose in this chapter an optimal node-to-set disjoint-path routing algorithm in hypercubes. Also, while retaining the optimal length of the generated paths as well as the $O(kn)$ optimal time complexity, our algorithm attains some kind of fault-tolerance. Effectively, without introducing any penalty to the maximum path length or to the time complexity, our algorithm optionally takes as input a set of faulty neighbours of the source node and generates fault-free disjoint paths. A cluster fault-tolerant node-to-set disjoint-path routing algorithm in hypercubes was proposed by Gu and Peng [55]. Addressing a different problem, this algorithm by Gu and Peng generates in a Q_n paths of lengths at most $n + 2$ and is thus not satisfying for our needs.

The rest of this chapter is organized as follows. Section 4.1 first describes as preliminaries a node-to-node routing algorithm in hypercubes. Then Section 4.2 describes the hypercube node-to-set disjoint-path routing algorithm Cube-N2S. Section 4.3 establishes the correctness and complexity of Cube-N2S. Finally Section 4.4 summarizes this chapter.

4.1 Preliminaries

We describe in Lemma 1 a hypercube node-to-node routing algorithm Cube-N2N, which finds a path between a source node s and a destination node d, avoiding some neighbours of s. This algorithm Cube-N2N will be used by the node-to-set disjoint-path routing algorithm Cube-N2S described in Section 4.2.

Lemma 1 *Inside a Q_n, given two nodes s, d and a set M of at most $n-1$ neighbours of s, we can find a path $P : s \rightsquigarrow d$ including neither an edge $s \to M$ nor a node in $M \setminus \{d\}$, and with $L(P) \leq n+1$ in $O(n)$ time complexity.*

Proof: First we select a node s' in the set $N(s) \setminus M$. Let δ be the unique dimension on which s and s' differ. We reduce Q_n into two subcubes Q^0_{n-1} and Q^1_{n-1} along the dimension δ. We can assume without loss of generality that $s \in Q^0$. If $d \in Q^1$, we connect s' to d by successively flipping the dimensions differing between s' and d in any order. See Figure 4.1. If $d \in Q^0$,

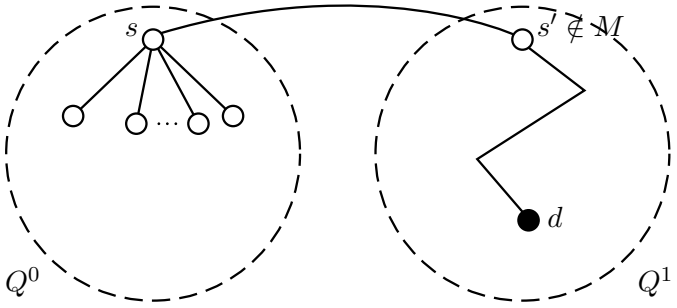

Figure 4.1: $d \in Q^1$.

we first connect s' to $d' = d \oplus 2^\delta$ by successively flipping the dimensions differing between s' and d' in any order, and we finally flip the dimension δ to reach d. See Figure 4.2. Because $M \subset Q^0$ and $P \cap Q^0 \subset \{s, d\}$, the

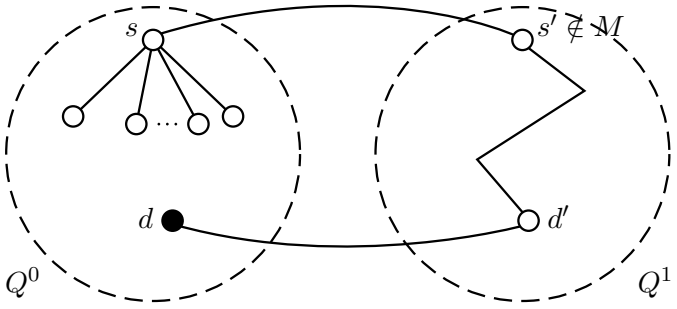

Figure 4.2: $d \in Q^0$.

statement $P \cap (M \setminus \{d\}) = \emptyset$ holds.

Regarding the maximum length of the path $s \to s' \rightsquigarrow d$, we distinguish two cases. First assume $H(s, d) \leq n-1$. If s' is on a shortest path s to d, that is $H(s', d) = H(s, d) - 1$, then the dimension δ is included in the differing bits between s and d. Therefore there are at most $n-2$ differing bits between s' and d. Hence s' is connected to d in at most $n-2$ edges. Taking the edge connecting s to s' into consideration, we obtain a path

of length at most $n-1$. If s' is not on a shortest path s to d, that is $H(s',d) = H(s,d)+1$, then there are at most n differing bits between s' and d. Hence s' is connected to d using at most n edges. Considering the edge connecting s to s', we obtain a path of length at most $n+1$. Now assume $H(s,d) = n$. Then the dimension δ must be included in the differing bits between s and d. Therefore the statement $H(s',d) = H(s,d)-1$ holds. Thus s' is connected to d with $n-1$ edges. Considering the edge connecting s to s', we obtain a path of length n.

This algorithm flips at most $n-1$ dimensions once, and at most one dimension twice, hence it performs at most $n+1$ dimension flips. Finding the dimension δ requires $O(n)$ time complexity. Each of the dimension flips is performed in constant time. Therefore Cube-N2N has a time complexity of $O(n)$. □

4.2 Node-to-set disjoint-path routing algorithm

We describe in this section a hypercube node-to-set disjoint-path routing algorithm Cube-N2S. In a Q_n, given a source node s, a set D of k ($k \leq n$) destination nodes and a set M of at most $n-k$ neighbours of s, Cube-N2S generates k disjoint paths between the source node and the destination nodes, not including an edge between s and M, and not including a node in $M \setminus D$. Cube-N2S recursively reduces the node-to-set disjoint-path routing problem to subproblems in subcubes of the original hypercube. The pseudocode of Cube-N2S is given in Algorithm 3.

Step 1 First if $s \in D$, we create the path of length zero s and remove s from D. Then we create the trivial paths of length one between the source node s and the nodes of $D \cap N(s) \setminus M$. We accordingly remove these connected destination nodes from D and add them into M. If there remains only one destination node inside D, we connect it to s using Cube-N2N and terminates the algorithm.

Step 2 We reduce Q_n into two subcubes Q^0_{n-1} and Q^1_{n-1} along a dimension δ ($0 \leq \delta \leq n-1$) such that $D \cap Q^0_{n-1} \neq \emptyset$ and $D \cap Q^1_{n-1} \neq \emptyset$. Assume $s \in Q^0_{n-1}$ and let s' be the unique neighbour of s into Q^1_{n-1} ($s' = s \oplus 2^\delta$).

Step 3 We collect in a new set M_1 all the neighbours $s'_i \in Q^1_{n-1}$ of s' whose neighbour s_i in Q^0_{n-1} is in M; formally $M_1 = \{s'_i \mid s'_i \in Q^1_{n-1} \cap N(s'), s_i (= s'_i \oplus 2^\delta) \in M\}$. See Figure 4.3.

Step 4 We apply Cube-N2S recursively onto Q^1_{n-1} to obtain a set $\mathcal{C}_1$ of disjoint paths from s' to the nodes of D', including neither an edge $s' \to M_1$ nor a node in $M_1 \setminus D'$, where D' is defined depending on the two cases below.

• $s' \in D \cap M$. If $s' \in D \cap M$ then we cannot use the edge $s \to s'$. Hence

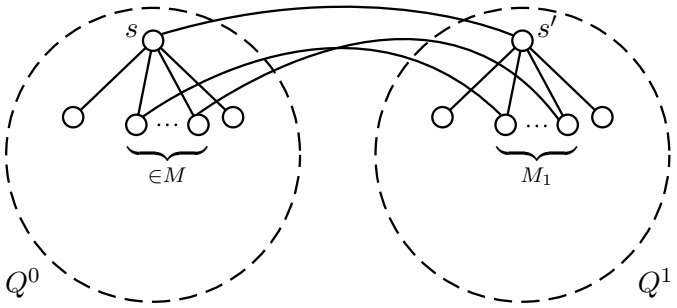

Figure 4.3: The set M_1.

we have to find a neighbour node s'_w of s' in Q^1_{n-1} for connection, and we consider s'_w as a destination node instead of s'. Then D' is defined as $D \cap Q^1_{n-1} \setminus \{s'\} \cup \{s'_w\}$.
• $s' \notin D \cap M$. In this case D' is simply defined as $D \cap Q^1_{n-1}$.

Step 5 If $s' \in D \cap M$ we replace the path $s' \to s'_w$ in $\mathcal{C}_1$ by the path $s' \to s'_w \to s'$ whose edge $s' \to s'_w$ will be later discarded.

Step 6 We connect the paths of $\mathcal{C}_1$ to s depending on the two cases below.
• $s' \notin M$. For each path $(s' \rightsquigarrow d_i) \in \mathcal{C}_1$ except one arbitrary path $(s' \rightsquigarrow d_j) \in \mathcal{C}_1, j \neq i$ we replace the edge $s' \to s'_i$ by the subpath $s \to s_i \to s'_i$ where $s_i = s'_i \oplus 2^\delta$. For the remaining path $s' \rightsquigarrow d_j$, we add the edge $s \to s'$.
• $s' \in M$. For each path $(s' \rightsquigarrow d_i) \in \mathcal{C}_1$ we replace the edge $s' \to s'_i$ by the subpath $s \to s_i \to s'_i$ where $s_i = s'_i \oplus 2^\delta$.

Step 7 Let $M_0 = (M \cap Q^0) \cup \{s_i \mid (s \to s_i \to s'_i \rightsquigarrow d_i) \in \mathcal{C}_1\}$. We apply Cube-N2S recursively onto Q^0_{n-1} to obtain disjoint paths from s to the nodes of $D \cap Q^0_{n-1}$, not including an edge $s \to M_0$ nor a node in $M_0 \setminus (D \cap Q^0_{n-1})$.

4.3 Correctness and complexities

We show the correctness of Cube-N2S by proving the following lemma.

Lemma 2 *In a Q_n, given a node s, a set of k ($k \leq n$) nodes $D = \{d_1, d_2, \ldots, d_k\}$ and a set M of at most $n-k$ neighbours of s, we can find k disjoint paths $s \rightsquigarrow d_i$ ($1 \leq i \leq k$) of lengths at most $n+1$, including neither an edge $s \to M$ nor a node in $M \setminus D$ in $O(kn)$ time complexity.*

Proof: In Step 1 it takes $O(k)$ time complexity to check if $s \in D$ or not. It also takes $O(k)$ time to generate the paths $s \to d_i \in D \cap N(s) \setminus M$. Their lengths are only one and they are trivially disjoint.

Algorithm 3 Cube-N2S(Q_n, s, D, M)

Input: A Q_n, a node s, a set of k ($k \leq n$) nodes $D = \{d_1, \ldots, d_k\}$ and a set M of at most $n-k$ neighbours of s.

Output: k disjoint paths $s \rightsquigarrow d_i$ ($1 \leq i \leq k$) including neither an edge $s \to M$ nor a node in $M \setminus D$.

if $s \in D$ **then** $\mathcal{C} := \{s\}$; /* *s: path of length 0* */
  $D := D \setminus \{s\}$ **else** $\mathcal{C} := \emptyset$ **end if**;
$\mathcal{C} := \mathcal{C} \cup \{s \to s_i \mid s_i \in D \cap N(s) \setminus M\}$;
$M := M \cup (D \cap N(s))$;
$D := D \setminus (N(s) \setminus M)$;
if $D = \emptyset$ **then return** $\mathcal{C}$ **end if**
if $|D| = 1$ **then**
  $\mathcal{C} := \mathcal{C} \cup \{\text{Cube-N2N}(Q_n, s, d_1, M)\}$
else
  Reduce Q_n along δ s.t. $D \cap Q^0_{n-1} \neq \emptyset$ and $D \cap Q^1_{n-1} \neq \emptyset$; /* *Assume* $s \in Q^0_{n-1}$ */
  $s' := s \oplus 2^\delta$; /* *Let* $s'_i \in Q^1_{n-1}$ *be the neighbour of* $s_i \in Q^0_{n-1}$ */
  $M_1 := \{s_i \oplus 2^\delta \mid s_i \in M \cap Q^0_{n-1}\}$;
  if $s' \in D \cap M$ **then**
    Find s'_w such that $s'_w \in N(s') \setminus (D \cup M_1 \cup \{s\})$;
    $D' := D \cap Q^1_{n-1} \setminus \{s'\} \cup \{s'_w\}$
  else
    $D' := D \cap Q^1_{n-1}$
  end if
  $\mathcal{C}_1 := \text{Cube-N2S}(Q^1_{n-1}, s', D', M_1)$;
  if $s' \in D \cap M$ **then**
    $\mathcal{C}_1 := (\mathcal{C}_1 \setminus \{s' \to s'_w\}) \cup \{s' \to s'_w \to s'\}$
  end if
  if $s' \notin M$ **then**
    Select $P \in \mathcal{C}_1$ arbitrarily;
    $\mathcal{C}_1 := \{s \to P\} \cup \{s \to s_{i_j} \to s'_{i_j} \rightsquigarrow d_j \mid s' \to s'_{i_j} \rightsquigarrow d_j \in \mathcal{C}_1 \setminus \{P\}\}$
  else
    $\mathcal{C}_1 := \{s \to s_{i_j} \to s'_{i_j} \rightsquigarrow d_j \mid s' \to s'_{i_j} \rightsquigarrow d_j \in \mathcal{C}_1\}$
  end if
  $M_0 := (M \cap Q^0_{n-1}) \cup \{s_{i_j} \mid (s \to s_{i_j} \to s'_{i_j} \rightsquigarrow d_j) \in \mathcal{C}_1\}$;
  $\mathcal{C}_0 := \text{Cube-N2S}(Q^0_{n-1}, s, D \cap Q^0_{n-1}, M_0)$;
  $\mathcal{C} := \mathcal{C} \cup \mathcal{C}_0 \cup \mathcal{C}_1$
end if
return $\mathcal{C}$

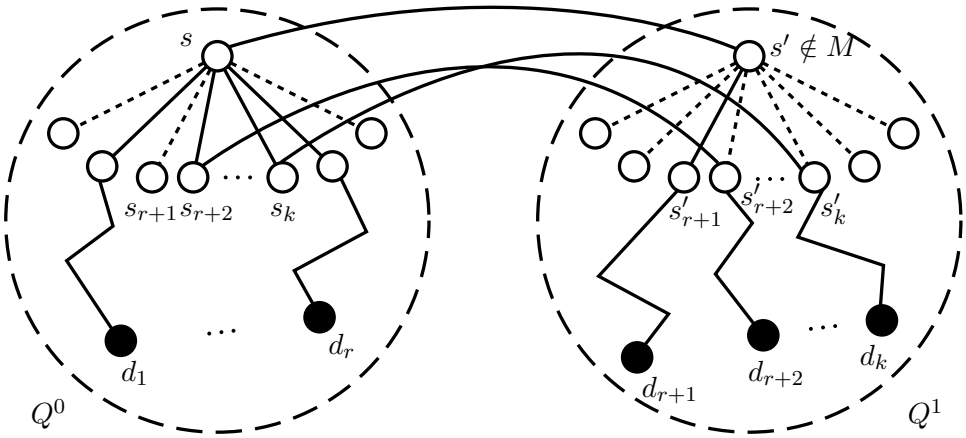

Figure 4.4: Node-to-set disjoint-path routing in a Q_n ($s' \notin M$).

We show by induction on n and k ($n \geq k$) that Cube-N2S generates disjoint paths of length at most $n+1$ not including an edge $s \to M$ and not including a node in $M \setminus D$ in $O(kn)$ time complexity. If $k = 1$, in a Q_n Cube-N2N generates a path $s \rightsquigarrow d \in D$ of length at most $n+1$ not including an edge $s \to M$ and not including a node in $M \setminus \{d\}$ in $O(n)$ time complexity. Hence Lemma 2 holds. Assume $k \geq 2$ and thus $n \geq 2$. We assume Lemma 2 holds for any hypercube of dimension smaller than n with at most $k-1$ destination nodes (induction hypothesis). We prove Lemma 2 holds for a hypercube of dimension n with k destination nodes. Let $T(k,n)$ represent the time complexity of Cube-N2S in a Q_n with $|D| = k$.

In Step 2 it takes $O(1)$ to find δ. We assume $D \cap Q^0_{n-1} = \{d_1, \ldots, d_r\}$ and $D \cap Q^1_{n-1} = \{d_{r+1}, \ldots, d_k\}$.

In Step 3, for each of the at most $n-k$ neighbours s_i of s in $M \cap Q^0$, we put in a new set M_1 the corresponding node s'_i ($= s_i \oplus 2^\delta$). Hence M_1 is created in $O(n)$ time complexity and we have $|M_1| \leq n-k$. Since there are at most $k-1$ destination nodes in Q^1_{n-1} we have $|D \cap Q^1_{n-1}| + |M_1| \leq (k-1) + (n-k) = n-1$. Hence, if $s' \in D \cap M$ then $|((D \cap Q^1_{n-1}) \cup M_1) \cap N(s')| \leq n-2$ since $s' \in D \cap Q^1_{n-1}$. Therefore, in Step 4, if $s' \in D \cap M$ then we can always find $s'_w \in N(s') \setminus (D \cup M_1 \cup \{s\})$, and $|D'| + |M_1| \leq n-1$ holds with $D' = D \cap Q^1_{n-1} \setminus \{s'\} \cup \{s'_w\}, |D'| = k - r \leq k-1$. Finding s'_w and creating D' takes $O(n)$ time complexity. Otherwise if $s' \notin D \cap M$ we have $D' = D \cap Q^1_{n-1}, |D'| = k - r \leq k-1$, also created in $O(n)$ time complexity. Because $|D'| + |M_1| \leq n-1$, we can find by induction onto Q^1_{n-1} $k-r$ disjoint paths of length at most $(n-1)+1 = n$ from s' to the nodes of D', not including an edge $s' \to M_1$ and not including a node in $M_1 \setminus D'$ in $T(k-r, n-1)$ time complexity.

In Step 5, if $s' \in D \cap M$, say $s' = d_j$ ($r+1 \leq j \leq k$), we extend the path $s' \to s'_w$ to $P_j : s' \to s'_w \to s' = d_j$ and we have $L(P_j) = 2 \leq n$ since

$n \geq 2$. Let P_i be the paths $s' \rightsquigarrow d_i$ $(r+1 \leq i \leq k)$ in Q^1_{n-1}. We have thus $L(P_i) \leq n$ $(r+1 \leq i \leq k)$.

In Step 6, if $s' \notin M$, then $|M \cap Q^0_{n-1}| \leq n-k$. We connect one path, say $P_{r+1} : s' \rightsquigarrow d_{r+1}$ to s by simply connecting s to s' in one edge, and we connect each of the remaining $k-r-1$ paths P_i $(r+2 \leq i \leq k)$ to s by replacing the edge $s' \rightarrow s'_i$ by the subpath $s \rightarrow s_i \rightarrow s'_i$. Because each path P_i $(r+2 \leq i \leq k)$ is using a distinct neighbour s'_i of s', the paths P_i $(r+2 \leq i \leq k)$ stay disjoint when linked to s via s_i the neighbour of s'_i in Q^0_{n-1}. Also, because s' is distinct from all the neighbours $s_i \in Q^0_{n-1}$ of s, P_{r+1} is disjoint with the other paths P_i $(r+2 \leq i \leq k)$. Because $P_{r+1} \cap Q^0_{n-1} = \{s\}$, $P_i \cap Q^0_{n-1} = \{s, s_i\}$ $(r+2 \leq i \leq k)$, $s \notin M$ and $s_i \notin M$ $(r+2 \leq i \leq k)$, we have $P_i \cap M = \emptyset$ $(r+1 \leq i \leq k)$. Now, let M_0 be a union of the neighbours $s_i \in Q^0_{n-1}$ of s used in the paths P_i $(r+2 \leq i \leq k)$, and $M \cap Q^0_{n-1}$. We have $|M_0| \leq (k-r-1) + (n-k) = n-r-1$. Because for each path P_i $(r+2 \leq i \leq k)$ we replace the edge $s' \rightarrow s'_i$ by the subpath $s \rightarrow s_i \rightarrow s'_i$, the new paths have a maximum length of $n-1+2 = n+1$. The path $P_{r+1} : s' \rightsquigarrow d_{r+1}$ of length at most n is linked to s with the edge $s \rightarrow s'$, hence the new path is of length at most $n+1$. See Figure 4.4.

Otherwise, if $s' \in M$, then $|M \cap Q^0_{n-1}| \leq (n-k)-1$. We connect each of the $k-r$ paths P_i $(r+1 \leq i \leq k)$ to s by replacing the edge $s' \rightarrow s'_i$ by the subpath $s \rightarrow s_i \rightarrow s'_i$. Because each path P_i $(r+1 \leq i \leq k)$ is using a distinct neighbour s'_i of s', the paths P_i $(r+1 \leq i \leq k)$ stay disjoint when linked to s via s_i the neighbour of s'_i in Q^0_{n-1}. For the same reason as in the case $s' \notin M$, we have $P_i \cap M = \emptyset$ $(r+1 \leq i \leq k)$. Now, let M_0 be a union of the neighbours $s_i \in Q^0_{n-1}$ of s used in the paths P_i $(r+1 \leq i \leq k)$, and $M \cap Q^0_{n-1}$. We have $|M_0| \leq (k-r) + (n-k-1) = n-r-1$. Because for each path P_i $(r+1 \leq i \leq k)$ we replace the edge $s' \rightarrow s'_i$ by the subpath $s \rightarrow s_i \rightarrow s'_i$, the new paths have a maximum length of $n-1+2 = n+1$.

Step 5 and 6 both require $O(k)$ time complexity. We have obtained $|M_0| \leq n-r-1$. In Step 7, because we have $|D \cap Q^0_{n-1}| + |M_0| \leq r + (n-r-1) = n-1$, we can find by induction onto Q^0_{n-1} r disjoint paths $P_i : s \rightsquigarrow d_i$ $(1 \leq i \leq r)$ of length at most $(n-1)+1 = n$ not including an edge $s \rightarrow M_0$ and not including a node in $M_0 \setminus (D \cap Q^0_{n-1})$ in $T(r, n-1)$ time complexity. In addition, because all the neighbour nodes $s_i \in Q^0_{n-1}$ of s used in the paths P_i $(r+1 \leq i \leq k)$ are included in M_0, and because excepted for these nodes $s_i \in Q^0_{n-1}$ the paths P_i $(r+1 \leq i \leq k)$ are located inside Q^1_{n-1}, the paths P_i $(1 \leq i \leq r)$ are disjoint with the paths P_i $(r+1 \leq i \leq k)$.

From this discussion, the k paths generated by Cube-N2S are disjoint, have lengths of at most $n+1$ and include neither an edge $s \rightarrow M$ nor a node in $M \setminus D$. Also we can express the time complexity of Cube-N2S by induction on n and k as follows.

$$T(1, n) = O(n) \tag{4.1}$$

$$T(k, n) = T(r, n-1) + T(k-r, n-1) + O(n) \tag{4.2}$$

$$= O(kn)$$

Equation 4.1 corresponds to the time complexity of Lemma 1. Cube-N2S has thus a total time complexity of $O(kn)$ which is time optimal. □

Now we can state the following theorem.

Theorem 1 *In a Q_n, given a node s, a set of k ($k \leq n$) nodes $D = \{d_1, d_2, \ldots, d_k\}$ and a set M of at most $n-k$ faulty neighbours of s with $D \cap M = \emptyset$, we can find k fault-free disjoint paths $s \rightsquigarrow d_i$ ($1 \leq i \leq k$) of lengths at most $n+1$ in $O(kn)$ time complexity.*

Proof: Since $D \cap M = \emptyset$, by Lemma 2 the disjoint paths returned by Cube-N2S(Q_n, s, D, M) are fault-free. □

4.4 Summary

We have described in this chapter a node-to-set disjoint-path routing algorithm Cube-N2S in a hypercube. Inside a Q_n, given a source node s, a set D of k ($k \leq n$) destination nodes and a set of at most $n-k$ faulty neighbours of s, Cube-N2S finds k fault-free node-disjoint paths between s and each node of D of lengths at most $n+1$ in $O(kn)$ time complexity.

Chapter 5

Node-to-set disjoint-path routing in perfect hierarchical hypercubes

We describe in this chapter a node-to-set disjoint-path routing algorithm HHC-N2S in perfect hierarchical hypercubes. The presentation of HHC-N2S is organized as follows. Section 5.1 recalls one lemma. Then Section 5.2 formally describes the algorithm HHC-N2S and gives its pseudocode. The proof of the correctness of HHC-N2S as well as its complexities are addressed in Section 5.3. An example of the execution trace of HHC-N2S is also given. An empirical evaluation of HHC-N2S is performed in Section 5.4. An improvement idea aimed at reducing the paths lengths is proposed in Section 5.5. Finally Section 5.6 summarizes this chapter.

5.1 Preliminaries

In this section we state Lemma 3 which shows that inside an HHC_{2^m+m} we can distribute all the $m+1$ destination nodes to distinct subcubes using disjoint paths of length at most two.

Lemma 3 *Inside an HHC_{2^m+m}, given a set of $m+1$ nodes $D = \{\boldsymbol{d}_1, \boldsymbol{d}_2, \ldots, \boldsymbol{d}_{m+1}\}$, we can find $m+1$ disjoint paths $\boldsymbol{d}_i \rightsquigarrow \boldsymbol{d}'_i$ $(1 \leq i \leq m+1)$ of length at most two in $O(m^3)$ time complexity such that the subcube of $\boldsymbol{d}'_i$ does not include any node in $D \cup (D' \setminus \{\boldsymbol{d}'_i\})$ where $D' = \{\boldsymbol{d}'_1, \boldsymbol{d}'_2, \ldots, \boldsymbol{d}'_{m+1}\}$.*

Proof: For an arbitrary node $\boldsymbol{d}_i = (\sigma, p_i) \in D$, there exist $m+1$ disjoint paths $P_1^{(i)}, \ldots, P_{m+1}^{(i)}$ of length at most two connecting $\boldsymbol{d}_i$ to $m+1$

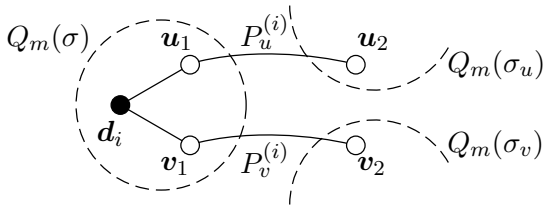

Figure 5.1: Two candidate paths for distribution of $\boldsymbol{d}_i$.

distinct subcubes:

$$\begin{cases} \boldsymbol{d}_i = (\sigma, p_i) \rightarrow (\sigma \oplus 2^{p_i}, p_i) \in Q_m(\sigma \oplus 2^{p_i}) \\ \boldsymbol{d}_i = (\sigma, p_i) \rightarrow (\sigma, p_i \oplus 2^h) \rightarrow (\sigma \oplus 2^{p_i \oplus 2^h}, p_i \oplus 2^h) \in Q_m(\sigma \oplus 2^{p_i \oplus 2^h}) \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (0 \leq h \leq m-1) \end{cases}$$

Now we show that for any $\boldsymbol{d}_j$ $(1 \leq j \leq m+1, i \neq j)$, the path $P_{w_j}^{(j)}$: $\boldsymbol{d}_j \rightarrow \boldsymbol{d}_j'' \rightarrow \boldsymbol{d}_j'$ $(1 \leq w_j \leq m+1)$ can block at most one of the $m+1$ paths $P_1^{(i)}, \ldots, P_{m+1}^{(i)}$.

Let us consider two paths $P_u^{(i)} : \boldsymbol{d}_i \rightarrow \boldsymbol{u}_1 \rightarrow \boldsymbol{u}_2 \in Q_m(\sigma_u)$ and $P_v^{(i)} : \boldsymbol{d}_i \rightarrow \boldsymbol{v}_1 \rightarrow \boldsymbol{v}_2 \in Q_m(\sigma_v)$ with $1 \leq u, v \leq m+1, u \neq v$ (see Figure 5.1). Because $\boldsymbol{u}_1$ and $\boldsymbol{v}_1$ are two distinct neighbours of the same node $\boldsymbol{d}_i$, we have $H(\boldsymbol{u}_1, \boldsymbol{v}_1) = 2$.

First we assume that $\boldsymbol{d}_j \in Q_m(\sigma)$. Then $\boldsymbol{d}_j'' \in Q_m(\sigma)$ and $\boldsymbol{d}_j' \notin Q_m(\sigma)$ hold. Hence $\boldsymbol{u}_1$ and $\boldsymbol{v}_1$ cannot be both on $P_{w_j}^{(j)}$ because $H(\boldsymbol{u}_1, \boldsymbol{v}_1) = 2$. In addition, if $\boldsymbol{d}_j = \boldsymbol{u}_1$ then $\boldsymbol{d}_j' \notin Q_m(\sigma_v)$ holds because there exists only one external edge $\boldsymbol{v}_1 \rightarrow \boldsymbol{v}_2$ between $Q_m(\sigma)$ and $Q_m(\sigma_v)$ and because $\boldsymbol{d}_j'' \neq \boldsymbol{v}_1$. Therefore $P_{w_j}^{(j)}$ cannot block both $P_u^{(i)}$ and $P_v^{(i)}$ if $\boldsymbol{d}_j \in Q_m(\sigma)$.

Next we assume that $\boldsymbol{d}_j \notin Q_m(\sigma)$ and $\boldsymbol{d}_j \in Q_m(\sigma_u)$ hold. We recall that $H(\sigma, \sigma_u) = H(\sigma, \sigma_v) = 1$, hence $H(\sigma_u, \sigma_v) = 2$ since $\boldsymbol{u}_1 \neq \boldsymbol{v}_1$. Therefore there is no external edge between $Q_m(\sigma_u)$ and $Q_m(\sigma_v)$, hence $\boldsymbol{d}_j' \notin Q_m(\sigma_v)$ holds since $P_{w_j}^{(j)}$ has only one external edge. Consequently, if $\boldsymbol{d}_j \in Q_m(\sigma_u)$ holds then $P_{w_j}^{(j)}$ cannot block $P_u^{(i)}$ and $P_v^{(i)}$ at the same time.

We can deduce from this discussion that each path $P_{w_j}^{(j)}$ $(1 \leq w_j \leq m+1)$ can block at most one of the $m+1$ paths $P_1^{(i)}, \ldots, P_{m+1}^{(i)}$. Hence at least $(m+1) - m = 1$ path $P_{w_i}^{(i)}$ $(1 \leq w_i \leq m+1)$ remains to connect $\boldsymbol{d}_i$ to a node $\boldsymbol{d}_i'$. $P_{w_i}^{(i)}$ can be found in $O(m^2)$ time complexity by checking all the $m+1$ paths $P_1^{(i)}, \ldots, P_{m+1}^{(i)}$ of length at most two for $\boldsymbol{d}_i$. Therefore we can connect all nodes $\boldsymbol{d}_i \in D$ to nodes $\boldsymbol{d}_i'$ with disjoint paths of lengths at most two in $O(m^3)$ time complexity. □

Given a subcube ID s_0, a set D of k $(k \leq m+1)$ nodes and a set Z of subcube IDs, we describe in Algorithm 4 an algorithm DISTRIB based on Lemma 3. When DISTRIB is used in the next section for distribution

of destination nodes, some of them are not necessarily distributed. For example, for destination nodes in $Q_m(s_0)$, disjoint paths from the source node are constructed inside $Q_m(s_0)$. Hence it is not necessary to distribute these destination nodes. Instead, the subcubes including the end nodes of opposite sides of external edges incident to the destination nodes in $Q_m(s_0)$ cannot be used for distribution. That is, any destination nodes in these subcubes must be distributed, and any destination nodes outside cannot be distributed to these subcubes. The parameters s_0 and Z are passed to DISTRIB to specify the subcube including the source node and the destination nodes to be distributed, respectively. Then, the algorithm distributes each $\boldsymbol{d}_i = (s_i, p_i) \in D$ with $s_i \in Z$ to distinct subcubes using disjoint paths of lengths at most two.

Algorithm 4 DISTRIB(s_0, $D = \{\boldsymbol{d}_1, \boldsymbol{d}_2, \ldots, \boldsymbol{d}_k\}$, Z)

Input: A subcube ID s_0, a set D of k ($k \leq m+1$) nodes and a set Z of subcube IDs.
Output: A set of HHC-level distributing disjoint paths $\boldsymbol{d}_i = (s_i, p_i) \rightsquigarrow \boldsymbol{d}'_i, s_i \in Z$.

```
H := ∅; D' := ∅;
D̃' := {(s_0 ⊕ 2^{p_i}, p_i) | d_i = (s_i, p_i) ∈ Q_m(s_0)};
for all d_i = (s_i, p_i) with s_i ∈ Z do
    Generate m + 1 paths P_1^(i), ..., P_{m+1}^(i) according to Lemma 3;
    Find a path P_{w_i}^(i) such that
        • P_{w_i}^(i) ∩ (D \ {d_i}) = ∅
        • Q_m(s'_i) ∩ ((D ∪ D' ∪ D̃') \ {d'_i}) = ∅
        • P_{w_i}^(i) ∩ P_{w_j}^(j) = ∅   (∀P_{w_j}^(j) ∈ H)
    hold;
    H := H ∪ {P_{w_i}^(i)};
    D' := D' ∪ {d'_i | P_{w_i}^(i) = d_i ⇝ d'_i}
end for
return H
```

5.2 Node-to-set disjoint-path routing algorithm

In this section we describe an algorithm HHC-N2S finding k ($k \leq m+1$) disjoint paths from a source node $\boldsymbol{s} = (s_0, p_0)$ to k destination nodes $\boldsymbol{d}_i = (s_i, p_i), 1 \leq i \leq k$ in an HHC_{2^m+m}. The main idea of this algorithm is to reduce the node-to-set disjoint-path routing problem in an HHC to the node-to-set disjoint-path routing problem in a hypercube via a 2^m-to-1 mapping of an HHC_{2^m+m} onto a Q_{2^m}. Concretely, for each node $(\sigma, \pi) \in HHC_{2^m+m}$, we map its subcube $Q_m(\sigma)$ to the single node σ of a Q_{2^m}. From there, we distinguish two types of nodes: HHC-level nodes are nodes of the HHC_{2^m+m}, and cube-level nodes are nodes of Q_{2^m}. An HHC-level path is made of HHC-level nodes and a cube-level path is made of cube-level nodes.

First we give an algorithm CONV which generates in an HHC_{2^m+m}, for a cube-level path $P : \sigma_0 \rightsquigarrow \sigma_n$ in Q_{2^m} of length l and two processor

IDs π_{beg}, π_{end}, an HHC-level path $(\sigma_0, \pi_{beg}) \rightsquigarrow (\sigma_n, \pi_{end})$ according to P by applying an SPR inside each visited subcube Q_m. Therefore CONV is $O(lm)$ time complexity. The pseudocode of CONV is given in Algorithm 5.

Algorithm 5 CONV($P = \sigma_0 \rightarrow \sigma_1 \rightarrow \ldots \rightarrow \sigma_n$, π_{beg}, π_{end})

Input: A cube-level path $P = \sigma_0 \rightsquigarrow \sigma_n$ and two processors IDs to specify the HHC nodes (σ_0, π_{beg}) and (σ_n, π_{end}).
Output: The HHC-level path corresponding to P.

```
if L(P) = 0 then
    π_0(= π_beg) → π_1 → ... → π_λ(= π_end) := π_beg ⇝(SPR) π_end;
    return (σ_0, π_0) → (σ_0, π_1) → ... → (σ_0, π_λ)
else
    π_next := log_2(σ_0 ⊕ σ_1);
    π_0(= π_beg) → π_1 → ... → π_λ'(= π_next) := π_beg ⇝(SPR) π_next;
    P' := CONV(σ_1 ⇝ σ_n, π_next, π_end);
    return (σ_0, π_0) → (σ_0, π_1) → ... → (σ_0, π_λ') → P'
end if
```

HHC-N2S is divided into two cases depending on the number of destination nodes inside $Q_m(s_0)$. If it is at least $k-1$, the problem is solved by applying Cube-N2S inside $Q_m(s_0)$ and generating at most one path going outside $Q_m(s_0)$. Otherwise we performs the following steps. In Step 1, we first distribute destination nodes into distinct subcubes. These subcubes are considered as destination nodes when applying Cube-N2S onto Q_{2^m} in Step 2. Step 3 replaces one path generated in Step 2 if a certain condition holds. In Step 4 we discard unnecessary cube-level paths generated in Step 2. Finally, in Step 5, the cube-level paths not discarded will be converted back to HHC-level paths using the CONV algorithm. It is important to note that if $k = m + 1$ then one path must include the edge $\boldsymbol{s} \rightarrow (s_0 \oplus 2^{p_0}, p_0)$ so that we can disjointly connect inside $Q_m(s_0)$ the other m paths to $\boldsymbol{s}$. To satisfy this requirement, a set $Z_4 \subset N(s_0)$ is introduced in Step 2 before applying Cube-N2S in Q_{2^m}. The pseudocode of HHC-N2S is given in Algorithm 6.

Case I: $|D \cap Q_m(s_0)| \geq k-1$
Assume without loss of generality that $\{\boldsymbol{d}_1, \ldots, \boldsymbol{d}_{k-1}\} \subset Q_m(s_0)$.

Case I-a: $\boldsymbol{d}_k \in Q_m(s_0)$ and $k < m+1$
The k disjoint paths $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}_i, 1 \leq i \leq k$ are found by applying Cube-N2S inside $Q_m(s_0)$.

Case I-b: $\boldsymbol{d}_k \in Q_m(s_0)$ and $k = m+1$
Apply Cube-N2S inside $Q_m(s_0)$ to find $k-1$ disjoint paths $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}_i, 1 \leq i \leq k-1$. If $\boldsymbol{d}_k$ is included on one of these m paths, say $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}_j$, discard the subpath $\boldsymbol{d}_k \rightsquigarrow \boldsymbol{d}_j$ and exchange the indices of the nodes $\boldsymbol{d}_j$ and $\boldsymbol{d}_k$. So there is still one destination node $\boldsymbol{d}_k$ to which a disjoint path from $\boldsymbol{s}$ is not obtained. Hence the path $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}_k$ is given by

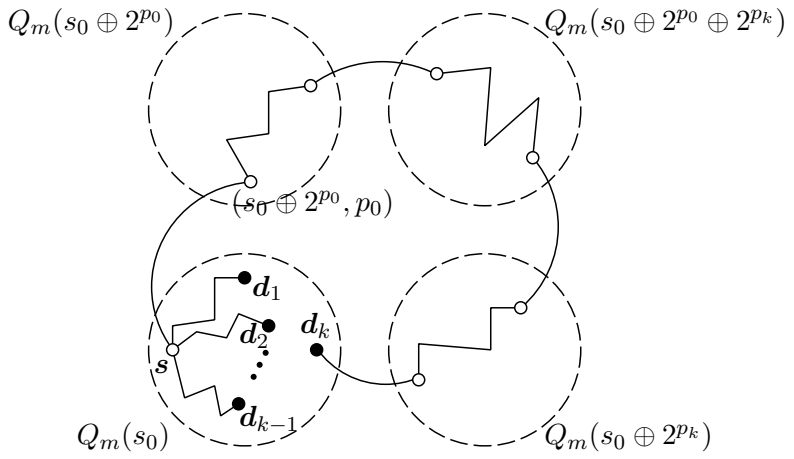

Figure 5.2: Disjoint paths generated in Case I-b.

$\boldsymbol{s} \rightarrow (s_0 \oplus 2^{p_0}, p_0) \overset{\text{SPR}}{\rightsquigarrow} (s_0 \oplus 2^{p_0}, p_k) \rightarrow (s_0 \oplus 2^{p_0} \oplus 2^{p_k}, p_k) \overset{\text{SPR}}{\rightsquigarrow} (s_0 \oplus 2^{p_0} \oplus 2^{p_k}, p_0) \rightarrow (s_0 \oplus 2^{p_k}, p_0) \overset{\text{SPR}}{\rightsquigarrow} (s_0 \oplus 2^{p_k}, p_k) \rightarrow (s_0, p_k) = \boldsymbol{d}_k$. See Figure 5.2.

Case I-c: $\boldsymbol{d}_k \notin Q_m(s_0)$

Apply Cube-N2S inside $Q_m(s_0)$ to find $k-1$ disjoint paths $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}_i, 1 \leq i \leq k-1$. The remaining path $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}_k$ is obtained by first finding a cube-level shortest path $P : s_0 \oplus 2^{p_0} \rightsquigarrow s_k \oplus 2^{p_k}$ so that P does not start with the edge $s_0 \oplus 2^{p_0} \rightarrow s_0$. Next, if $s_k \in P$ discard the subpath $s_k \rightsquigarrow s_k \oplus 2^{p_k}$ of P and apply CONV(P, p_0, p_k). Otherwise if $s_k \notin P$ apply CONV(P, p_0, p_k) and extend the path obtained by one external edge $(s_k \oplus 2^{p_k}, p_k) \rightarrow \boldsymbol{d}_k$.

Case II: $|D \cap Q_m(s_0)| \leq k-2$

Step 1 Assume without loss of generality that $Q_m(s_0) \cap D = \{\boldsymbol{d}_1, \ldots, \boldsymbol{d}_r\}$ ($r \leq k-2$). Let Z_1 be the set of subcube IDs whose corresponding subcubes contain at least two destination nodes (s_0 is excluded from Z_1), formally $Z_1 = \{\sigma \mid |Q_m(\sigma) \cap D| \geq 2\} \setminus \{s_0\}$. Let Z_2 be the set of subcube IDs not in Z_1 such that each of corresponding subcubes is linked to $Q_m(s_0)$ with an external edge whose end node in $Q_m(s_0)$ is a destination node, formally $Z_2 = \{s_0 \oplus 2^{p_1}, \ldots, s_0 \oplus 2^{p_r}\} \setminus Z_1$.

Find a path of length at most two from each destination node $\boldsymbol{d}_i = (s_i, p_i)$ with $s_i \in Z_1 \cup Z_2$ to a node $\boldsymbol{d}'_i = (s'_i, p'_i)$ (called distributed destination node for $\boldsymbol{d}_i$) by applying DISTRIB(s_0, D, $Z_1 \cup Z_2$). Formally, $\forall \boldsymbol{d}_i, 1 \leq i \leq k$ with $s_i \in Z_1 \cup Z_2$, three statements hold with respect to its distribution path $\boldsymbol{d}_i \rightarrow \boldsymbol{d}''_i \rightarrow \boldsymbol{d}'_i$: $s'_i \notin Z_2$, $Q_m(s'_i) \cap (D \cup D' \setminus \{\boldsymbol{d}'_i\}) = \emptyset$ and $\boldsymbol{d}''_i \notin D$, where $D' = \{\boldsymbol{d}'_j \mid s_j \in Z_1 \cup Z_2\}$. Note that if

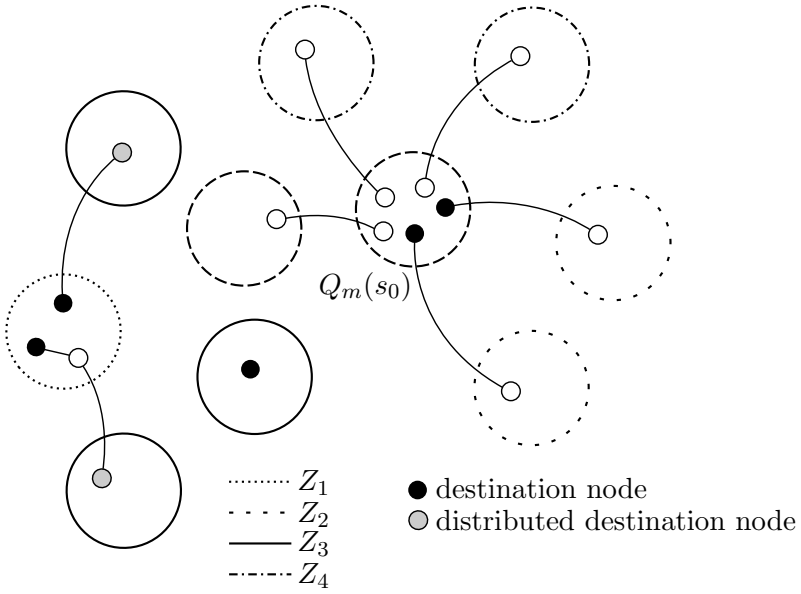

Figure 5.3: The four disjoint sets of subcube IDs Z_1, Z_2, Z_3 and Z_4.

$Q_m(s_0) \cap D = \emptyset$, one destination node can be distributed to a node in $Q_m(s_0)$.

Step 2 Let Z_3 be the union of the set of subcube IDs whose corresponding subcubes contain one distributed destination node, and the set of subcube IDs not in $Z_2 \cup \{s_0\}$ whose corresponding subcubes contain only one destination node. Formally $Z_3 = \{s'_i \mid s_i \in Z_1 \cup Z_2\} \cup \{s_i \mid s_i \notin Z_1 \cup Z_2 \cup \{s_0\}\}$. Let Z_4 be a set of subcube IDs neighbours of s_0 such that $Z_4 \subset N(s_0) \setminus (Z_1 \cup Z_2 \cup Z_3)$ and $|Z_4| = 2^m - |Z_1| - |Z_2| - |Z_3 \setminus \{s_0\}|$. If $|N(s_0) \setminus (Z_1 \cup Z_2 \cup Z_3)| > 2^m - |Z_1| - |Z_2| - |Z_3 \setminus \{s_0\}|$, we construct Z_4 so that $s_0 \oplus 2^{p_0} \notin Z_4$ holds. Note that Z_4 is not always unique. See Figure 5.3 for an illustration of the four disjoint sets Z_1, Z_2, Z_3 and Z_4.

Apply Cube-N2S in Q_{2^m} with Cube-N2S(Q_{2^m}, s_0, $Z_1 \cup (Z_3 \setminus \{s_0\})$, $Z_2 \cup Z_4$) to obtain a set $\mathcal{C}_0$ of cube-level disjoint paths. If $s_0 \in Z_3$, add into $\mathcal{C}_0$ the cube-level path of length zero s_0. We have $|\mathcal{C}_0| = |Z_1 \cup Z_3|$.

Step 3 We introduce an additional treatment for the case $s_0 \oplus 2^{p_0} \in Z_4$ holds. Otherwise go to Step 4. First select an arbitrary node $s_i = s_0 \oplus 2^q \in Z_3 \cap N(s_0)$. Then add into $\mathcal{C}_0$ the path $s_0 \to s_0 \oplus 2^{p_0} \to s_0 \oplus 2^{p_0} \oplus 2^q \to s_0 \oplus 2^q = s_i$ of length three and remove from $\mathcal{C}_0$ the path $s_0 \to s_i$ of length one.

Step 4 Remove from $\mathcal{C}_0$ the paths $s_0 \rightsquigarrow \sigma \in Z_1$ not including $s_0 \oplus 2^{p_0}$. If $\mathcal{C}_0$ includes a path $P : s_0 \to s_0 \oplus 2^{p_0} \rightsquigarrow \varsigma \to \sigma \in Z_1$, considering the

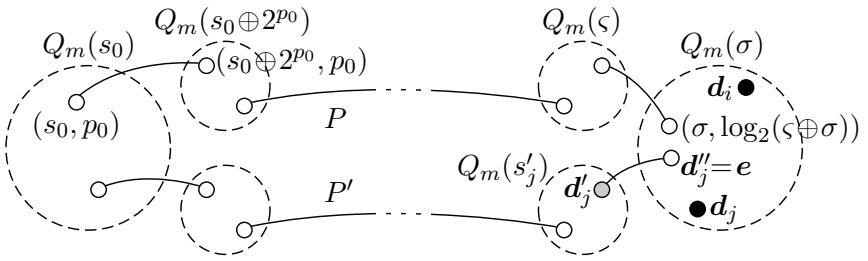

Figure 5.4: Two paths P and P'.

set $E = \{\boldsymbol{d}_i \mid \boldsymbol{d}_i \in Q_m(\sigma)\} \cup \{\boldsymbol{d}''_i \mid \boldsymbol{d}_i \to \boldsymbol{d}''_i \to \boldsymbol{d}'_i,\ \boldsymbol{d}_i, \boldsymbol{d}''_i \in Q_m(\sigma)\}$, let $\boldsymbol{e}(= \boldsymbol{d}_j$ or $\boldsymbol{d}''_j) \in E$ be the closest node of E to the node $(\sigma, \log_2(\varsigma \oplus \sigma))$. Since P will be used to connect $\boldsymbol{s}$ to $\boldsymbol{e}$ in Step 5, we remove from $\mathcal{C}_0$ the path $P' : s_0 \rightsquigarrow s'_j$ where $(s'_j, p'_j) = \boldsymbol{d}'_j$. See Figure 5.4. We have $|\mathcal{C}_0| = k - r$.

Step 5 First extend each path $(s_0 \rightsquigarrow s'_i) \in \mathcal{C}_0$ with an edge $s'_i \to s_i$ ($\mathcal{C}_0$ is updated). Next, convert the $k - r$ cube-level paths $P_i : (s_0 \rightsquigarrow s_i) \in \mathcal{C}_0, r + 1 \le i \le k$ back to HHC-level paths using CONV as follows.

Assume without loss of generality that P_{r+1} includes the edge $s_0 \to s_0 \oplus 2^{p_0}$. If P_{r+1} connects an element of Z_1 (i.e. $s_{r+1} \in Z_1$), that is P of Step 4 exists, its conversion requires special treatment. First apply CONV(P_{r+1}, p_0, π) where $(s_{r+1}, \pi) = \boldsymbol{e}$ ($\boldsymbol{e} \in \{\boldsymbol{d}_{r+1}, \boldsymbol{d}''_{r+1}\}$, see Step 4). Second, if $\boldsymbol{e} = \boldsymbol{d}''_{r+1}$, extend that path with the edge $\boldsymbol{e} \to \boldsymbol{d}_{r+1}$. Third, for each path $P_i, r + 2 \le i \le k : s_0 \to s_0 \oplus 2^{\pi_i} \rightsquigarrow s_i$ apply CONV(P_i, π_i, p_i). Now if $s_{r+1} \notin Z_1$, that is P does not exist, for each path $P_i, r + 1 \le i \le k : s_0 \to s_0 \oplus 2^{\pi_i} \rightsquigarrow s_i$ apply CONV(P_i, π_i, p_i).

Finally apply Cube-N2S inside $Q_m(s_0)$ to connect $\boldsymbol{s}$ to the $k-1$ ($\le m$) nodes of the set $(Q_m(s_0) \cap D) \cup \{(s_0, \pi_i) \mid s_0 \oplus 2^{\pi_i} \in P_i, r+2 \le i \le k\}$. See Figure 5.5.

5.3 Correctness and complexities

In this section we prove the correctness of HHC-N2S and estimate its time complexity as well as the theoretical maximum path length.

Lemma 4 *Case I generates disjoint paths of lengths at most* $m2^m + 2^m - m + 1$ *in* $O(m2^m)$ *time complexity.*

Proof: Cases I-a, I-b and I-c use Cube-N2S to disjointly connect $\boldsymbol{s}$ and k or $k - 1$ destination nodes in $Q_m(s_0)$. By Lemma 2, these disjoint paths are generated in $O(km)$ time complexity, and their lengths are at most

Algorithm 6 HHC-N2S(HHC_{2^m+m}, $\boldsymbol{s}$, $D = \{\boldsymbol{d}_1, \ldots, \boldsymbol{d}_k\}$)

Input: An HHC_{2^m+m}, a source node $\boldsymbol{s} = (s_0, p_0)$ and a set D of k ($k \leq m+1$) destination nodes $\boldsymbol{d}_i = (s_i, p_i)$ ($1 \leq i \leq k$).
Output: k disjoint paths $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}_i$ ($1 \leq i \leq k$).

if $|D \cap Q_m(s_0)| \geq k-1$ **then** /* *Case I - Assume* $\{\boldsymbol{d}_1, \ldots, \boldsymbol{d}_{k-1}\} \subset D \cap Q_m(s_0)$ */
  if $\boldsymbol{d}_k \in Q_m(s_0)$ and $k < m+1$ **then** /* *Case I-a* */
    $\mathcal{S}_0$:= Cube-N2S($Q_m(s_0)$, p_0, $\{p_1, \ldots, p_k\}$, $\emptyset$);
    $\mathcal{H} := \{(s_0, p_0) \rightsquigarrow (s_0, p_i) \mid \forall (p_0 \rightsquigarrow p_i) \in \mathcal{S}_0, 1 \leq i \leq k\}$
  else if $\boldsymbol{d}_k \in Q_m(s_0)$ and $k = m+1$ **then** /* *Case I-b* */
    $\mathcal{S}_0$:= Cube-N2S($Q_m(s_0)$, p_0, $\{p_1, \ldots, p_{k-1}\}$, $\emptyset$);
    $\mathcal{H} := \{(s_0, p_0) \rightsquigarrow (s_0, p_i) \mid \forall (p_0 \rightsquigarrow p_i) \in \mathcal{S}_0, 1 \leq i \leq k-1\}$; /* *If* $\boldsymbol{d}_k \in \mathcal{H}$, *additional processing required, but omitted.* */
    $\mathcal{H} := \mathcal{H} \cup \{\boldsymbol{s} \rightarrow (s_0 \oplus 2^{p_0}, p_0) \overset{\text{SPR}}{\rightsquigarrow} (s_0 \oplus 2^{p_0}, p_k) \rightarrow (s_0 \oplus 2^{p_0} \oplus 2^{p_k}, p_k) \overset{\text{SPR}}{\rightsquigarrow} (s_0 \oplus 2^{p_0} \oplus 2^{p_k}, p_0) \rightarrow (s_0 \oplus 2^{p_k}, p_0) \overset{\text{SPR}}{\rightsquigarrow} (s_0 \oplus 2^{p_k}, p_k) \rightarrow \boldsymbol{d}_k\}$
  else /* *Case I-c* */
    $\mathcal{S}_0$:= Cube-N2S($Q_m(s_0)$, p_0, $\{p_1, \ldots, p_{k-1}\}$, $\emptyset$);
    $\mathcal{H} := \{(s_0, p_0) \rightsquigarrow (s_0, p_i) \mid \forall (p_0 \rightsquigarrow p_i) \in \mathcal{S}_0, 1 \leq i \leq k-1\}$;
    $P := s_0 \oplus 2^{p_0} \rightsquigarrow s_k \oplus 2^{p_k}$; /* P *is a cube-level shortest-path between* $s_0 \oplus 2^{p_0}$ *and* $s_k \oplus 2^{p_k}$ *not including* s_0. */
    if $s_k \in P$ **then**
      Discard the subpath $s_k \rightsquigarrow s_k \oplus 2^{p_k}$ of P;
      $\mathcal{H} := \mathcal{H} \cup \{\boldsymbol{s} \rightarrow \text{CONV}(P, p_0, p_k)\}$
    else
      $\mathcal{H} := \mathcal{H} \cup \{\boldsymbol{s} \rightarrow \text{CONV}(P, p_0, p_k) \rightarrow \boldsymbol{d}_k\}$
    end if
  end if
else /* *Case II - Assume* $\{\boldsymbol{d}_1, \ldots, \boldsymbol{d}_r\} = D \cap Q_m(s_0)$ *with* $r \leq k-2$ */
  /* *Step 1* */
  $Z_1 := \{\sigma \mid |D \cap Q_m(\sigma)| \geq 2\} \setminus \{s_0\}$;
  $Z_2 := \{s_0 \oplus 2^{p1}, \ldots, s_0 \oplus 2^{p_r}\} \setminus Z_1$;
  $\mathcal{H}' :=$ DISTRIB(s_0, $\{\boldsymbol{d}_{r+1}, \ldots, \boldsymbol{d}_k\}$, $Z_1 \cup Z_2$);
  /* *Step 2* */
  $Z_3 := \{s'_i \mid s_i \in Z_1 \cup Z_2\} \cup \{s_i \mid s_i \notin Z_1 \cup Z_2 \cup \{s_0\}\}$; /* $(\boldsymbol{d}_i \rightarrow (\boldsymbol{d}''_i \rightarrow)\ \boldsymbol{d}'_i = (s'_i, p'_i)) \in \mathcal{H}'$ */
  $Z_4 := \{\sigma_j \mid 1 \leq j \leq 2^m - |Z_1 \cup Z_2 \cup (Z_3 \setminus \{s_0\})|, \sigma_j \in N(s_0) \setminus (Z_1 \cup Z_2 \cup Z_3)\}$;
    /* Z_4 *is not always unique and should avoid including* $s_0 \oplus 2^{p_0}$ *if possible.* */
  $\mathcal{C}_0$:= Cube-N2S(Q_{2^m}, s_0, $Z_1 \cup (Z_3 \setminus \{s_0\})$, $Z_2 \cup Z_4$);
  if $s_0 \in Z_3$ **then** $\mathcal{C}_0 := \mathcal{C}_0 \cup \{s_0\}$ **end if**
  /* *Step 3* */
  if $s_0 \oplus 2^{p_0} \in Z_4$ **then**
    Find a path P from $s_0 \oplus 2^{p_0}$ to an arbitrary $s_i \in Z_3 \cap N(s_0)$ of two edges not including s_0; /* $Z_3 \cap N(s_0) \neq \emptyset$ */
    $\mathcal{C}_0 := (\mathcal{C}_0 \setminus \{s_0 \rightarrow s_i\}) \cup \{s_0 \rightarrow P\}$
  end if
  /* *Step 4* */
  for $i = r+1$ to k **do**
    if $s_i \in Z_3$ **then** $P_i := (s_0 \rightsquigarrow s_i) \in \mathcal{C}_0$ **else** $P_i := (s_0 \rightsquigarrow s'_i) \in \mathcal{C}_0$ **end if**
  end for
  if $\exists P = (s_0 \rightsquigarrow s_i) \in \mathcal{C}_0$ with $s_0 \oplus 2^{p_0} \in P, s_i \in Z_1$ **then** $\mathcal{C}_1 := \{P\}$ **else** $\mathcal{C}_1 := \emptyset$ **end if**
  /* *The remaining paths* $s_0 \rightsquigarrow \sigma \in Z_1$ *not including* $s_0 \oplus 2^{p_0}$ *of* $\mathcal{C}_0$ *are discarded.* */
  /* *Step 5* */
  for $i = r+1$ **to** k **do if** $s_i \notin Z_3$ **then** $P_i := P_i \rightarrow s_i$ **end if end for**
  if $\mathcal{C}_1 \neq \emptyset$ **then** /* *Assume* $\mathcal{C}_1 = \{P = s_0 \rightsquigarrow \varsigma \rightarrow \sigma\}$ */
    Find $\boldsymbol{e} = (\sigma, \pi)$ the closest node in $\mathcal{H}' \cap Q_m(\sigma)$ to the node $(\sigma, \log_2(\varsigma \oplus \sigma))$;
    $P_j := P$; /* *Assume* $\boldsymbol{e} = \boldsymbol{d}_j$ or $\boldsymbol{d}''_j$ */
    $H_j :=$ CONV(P_j, p_0, π); **if** $\boldsymbol{e} = \boldsymbol{d}''_j$ **then** $H_j := H_j \rightarrow \boldsymbol{d}_j$ **end if**
  end if
  for $i = r+1$ to k **do** /* *Assume* $P_i = s_0 \rightarrow \sigma \rightsquigarrow s_i$ */
    if $\mathcal{C}_1 = \emptyset$ or $i \neq j$ **then** $H_i :=$ CONV(P_i, $\log_2(s_0 \oplus \sigma)$, p_i) **end if**
  end for
  /* *Assume* H_{r+1} *is the path starting with* $\boldsymbol{s} \rightarrow (s_0 \oplus 2^{p_0}, p_0)$ */
  $\mathcal{S}_0$:= Cube-N2S($Q_m(s_0)$, p_0, $\{p_1, \ldots, p_r\} \cup \{\pi_i \mid H_i = ((s_0, \pi_i) \rightsquigarrow \boldsymbol{d}_i), r+2 \leq i \leq k\}$, $\emptyset$);
  $\mathcal{H}_0 := \{(s_0, p_0) \rightsquigarrow (s_0, \pi) \mid \forall (p_0 \rightsquigarrow \pi) \in \mathcal{S}_0\}$;
  $\mathcal{H} :=$ Join the paths $\{(s_0, p_0) \rightsquigarrow (s_0, \pi_i) \mid H_i = ((s_0, \pi_i) \rightsquigarrow \boldsymbol{d}_i), r+2 \leq i \leq k\} \subset \mathcal{H}_0$ to the paths $H_i, r+2 \leq i \leq k$;
  $\mathcal{H} := \mathcal{H} \cup \{((s_0, p_0) \rightsquigarrow (s_0, p_i)) \in \mathcal{H}_0 \mid 1 \leq i \leq r\} \cup \{H_{r+1}\}$
end if
return $\mathcal{H}$

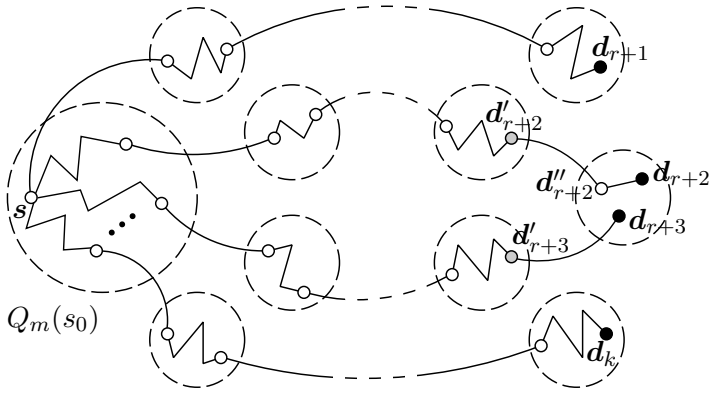

Figure 5.5: Disjoint paths generated by HHC-N2S.

$m+1$. Case I-b generates an additional path going outside $Q_m(s_0)$ of length at most $3m+4$ since it consists of four external edges and three subcube routings. That path can be constructed in $O(m)$ time complexity. Case I-c also generates an additional path going outside $Q_m(s_0)$, this time of length at most $(2^m+1)+m(2^m-1) = m2^m+2^m-m+1$ since it consists of at most 2^m+1 external edges and at most 2^m-1 subcube routings. That path can be constructed in $O(m2^m)$ time complexity. Regarding the additional path generated in Case I-b and I-c, all its nodes other than its end nodes $\boldsymbol{s}$ and $\boldsymbol{d}_k$ are outside $Q_m(s_0)$. Hence, this path is disjoint from the other paths. In Case I-b, checking if $\boldsymbol{d}_k$ is included on a path generated inside $Q_m(s_0)$ requires $O(km)$ time complexity since at most $k-1$ paths of lengths at most $m+1$ are checked. □

Lemma 5 *Step 1 of Case II requires $O(m^3)$ time complexity.*

Proof: The sets Z_1 and Z_2 can be created in $O(k^2)$ and $O(k)$ time complexity, respectively. By Lemma 3, all destination nodes included in subcubes whose corresponding subcube IDs are in $Z_1 \cup Z_2$ can be distributed to distinct subcubes by disjoint paths of lengths at most two in $O(m^3)$ time complexity. □

Lemma 6 *Step 2 of Case II requires $O(k2^m)$ time complexity and generates cube-level disjoint paths of lengths at most 2^m+1.*

Proof: First we show the construction of Z_4 is possible, that is $|Z_4| \geq 0$. Because we have $|Z_4| = 2^m - |Z_1| - |Z_2| - |Z_3 \setminus \{s_0\}|$, we need that $|Z_1|+|Z_2|+|Z_3 \setminus \{s_0\}| \leq 2^m$ holds. Let us count how many elements each set Z_1, Z_2 and $Z_3 \setminus \{s_0\}$ contains. Assume $Q_m(s_0) \cap D = \{\boldsymbol{d}_1, \ldots, \boldsymbol{d}_r\}, r \leq k-2$. Hence we have $|Z_2| \leq r$. Since $k-r$ destination nodes are outside $Q_m(s_0)$, $|Z_1| \leq \lfloor (k-r)/2 \rfloor$. Also we have $|Z_3| = k-r$ and $|Z_3 \setminus \{s_0\}| \leq k-r$. We

solve the following inequality ($|Z_1|+|Z_2|+|Z_3 \setminus \{s_0\}|$ is maximised for $r = 0$ and $k = m + 1$)

$$\begin{aligned} |Z_1| + |Z_2| + |Z_3 \setminus \{s_0\}| &\leq \lfloor (k-r)/2 \rfloor + r + (k-r) \\ &= \lfloor (k-r)/2 \rfloor + k \\ &\leq \lfloor (m+1)/2 \rfloor + (m+1) \\ &\leq 2^m \quad \Leftrightarrow \quad m \geq 2 \end{aligned}$$

Therefore $|Z_4| \geq 0$ holds for $m \geq 2$. Because we have $|Z_1|+|Z_2|+|Z_3\setminus\{s_0\}|+|Z_4| = 2^m$, we can apply Cube-N2S to solve the node-to-set disjoint-path routing problem in Q_{2^m}. If $m = 1$, the corresponding HHC is a cycle and it is then trivial to disjointly route to at most $m + 1 = 2$ destination nodes. The sets Z_3 and Z_4 can be created in $O(k)$ and $O(2^m)$ time complexity, respectively. By Lemma 2, the cube-level paths generated are disjoint, have a maximum length of $2^m + 1$ and require $O(k2^m)$ time complexity since $|Z_1|$ and $|Z_3 \setminus \{s_0\}|$ are $O(k)$. □

Lemma 7 *In Step 3 of Case II,* $s_0 \oplus 2^{p_0} \in Z_4 \Rightarrow Z_3 \cap N(s_0) \neq \emptyset$.

Proof: If $s_0 \in Z_3$ or $Z_1 \cup Z_2 \cup (Z_3 \setminus \{s_0\}) \not\subset N(s_0)$, then $|N(s_0) \setminus (Z_1 \cup Z_2 \cup Z_3)| > 2^m - |Z_1| - |Z_2| - |Z_3 \setminus \{s_0\}| = |Z_4|$ holds, that is Z_4 can be constructed so as not to include $s_0 \oplus 2^{p_0}$. It is a contradiction, hence $s_0 \notin Z_3$ and $Z_1 \cup Z_2 \cup (Z_3 \setminus \{s_0\}) = Z_1 \cup Z_2 \cup Z_3 \subset N(s_0)$. Now assume that there is a subcube ID σ in $Z_1 \cap N(s_0)$. Then $Q_m(\sigma)$ has multiple destination nodes, and $Q_m(s_0)$ can have at most one distributed destination node. Hence, at least one destination node $\boldsymbol{d}_i$ in $Q_m(\sigma)$ has to be distributed to $Q_m(s'_i)$ where $\boldsymbol{d}'_i = (s'_i, p'_i)$ is the distributed destination node of $\boldsymbol{d}_i$ and $s'_i \neq s_0$. In addition, $H(s_0, \sigma) = 1$ and $H(\sigma, s'_i) = 1$. Then $H(s_0, s'_i) \neq 1$, and it means $s'_i \notin N(s_0)$. Therefore, $s'_i \notin N(s_0) \cup \{s_0\}$ holds. Hence $Z_3 \setminus \{s_0\} \not\subset N(s_0)$. It is a contradiction, then $Z_1 \cap N(s_0) = \emptyset$. As a result from $Z_1 \subset N(s_0)$ and $Z_1 \cap N(s_0) = \emptyset$, we have $Z_1 = \emptyset$. If $Z_3 \cap N(s_0) = \emptyset$ then $|D \cap Q_m(s_0)| = k \geq k - 1$, but this situation must be handled in Case I. Hence $Z_3 \cap N(s_0) \neq \emptyset$. □

Lemma 8 *Step 3 of Case II requires* $O(2^m)$ *time complexity. If* $s_0 \oplus 2^{p_0} \in Z_4$ *then it replaces a path* $(s_0 \rightarrow s_i) \in \mathcal{C}_0$ *where* $s_i = s_0 \oplus 2^q \in Z_3 \cap N(s_0)$ *by the path of length three* $s_0 \rightarrow s_0 \oplus 2^{p_0} \rightarrow s_0 \oplus 2^{p_0} \oplus 2^q \rightarrow s_i$ *disjoint with the other paths of* $\mathcal{C}_0$.

Proof: Since $|Z_4|$ is $O(2^m)$, we can check if $s_0 \oplus 2^{p_0} \in Z_4$ holds in $O(2^m)$ time complexity. By Lemma 7 we have $Z_3 \cap N(s_0) \neq \emptyset$, hence we can select an arbitrary node $s_i = s_0 \oplus 2^q \in Z_3 \cap N(s_0)$. Also by Lemma 7 we have $Z_1 \cup Z_2 \cup Z_3 \subset N(s_0)$ and $Z_1 = \emptyset$, hence the destination nodes outside $Q_m(s_0)$ are in subcubes whose corresponding subcube IDs are in $N(s_0)$.

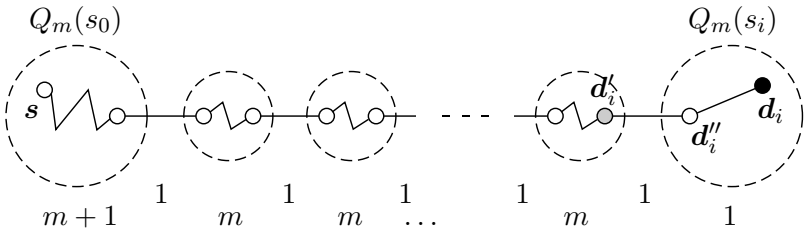

Figure 5.6: Counting the maximum number of edges.

Then the path $s_0 \to s_0 \oplus 2^{p_0} \to s_0 \oplus 2^{p_0} \oplus 2^q \to s_0 \oplus 2^q = s_i$ of length three is disjoint with the other paths $(s_0 \to s_j) \in \mathcal{C}_0, i \neq j$. Removing the path $s_0 \to s_i$ from $\mathcal{C}_0$ requires $O(k)$ time complexity since $|\mathcal{C}_0| = |Z_1 \cup Z_3|$ holds, and $|Z_1|$ and $|Z_3 \setminus \{s_0\}|$ are $O(k)$. □

Lemma 9 *Step 4 of Case II requires $O(k^2)$ time complexity.*

Proof: Since $|Z_1|$ and $|Z_3|$ are $O(k)$, we can check for each path of $\mathcal{C}_0$ if it should be discarded or not in $O(k)$ time complexity. Because $|\mathcal{C}_0| = |Z_1 \cup Z_3|$, the total time complexity of Step 4 is $O(k^2)$. □

Lemma 10 *Step 5 of Case II generates HHC-level disjoint paths of lengths at most $m2^m + 2^m + 2m + 4$ in $O(km2^m)$ time complexity.*

Proof: By Lemma 6, all the subcubes used by CONV are distinct. Hence the HHC-level paths generated by CONV are disjoint. The paths generated inside $Q_m(s_0)$ are disjoint by Lemma 2. An HHC-level path generated by Cube-N2S inside $Q_m(s_0)$ and an HHC-level path generated based on Cube-N2S and CONV outside $Q_m(s_0)$ are disjoint since they do not share any node except for a common end node in $Q_m(s_0)$. Therefore Step 5 generates HHC-level disjoint paths.

By Lemma 6, the paths of $\mathcal{C}_0$, which are cube-level paths, have lengths at most $2^m + 1$. Hence applying CONV to one such path requires $O(m2^m)$. Since $|\mathcal{C}_0|$ is $O(k)$, the total time complexity of Step 5 is $O(km2^m)$.

Some of the paths of $\mathcal{C}_0$ may be extended by one edge. Hence the cube-level paths $P_{r+1}, \ldots, P_k$ have lengths at most $2^m + 2$. For a path of length $2^m + 2$, Cube-N2S generates a path of length at most $m + 1$ in $Q_m(s_0)$. Inside the $2^m + 1$ intermediate subcubes, shortest-path routings generate paths of length at most m. In the last subcube, a path of one edge is generated. See Figure 5.6. Therefore Step 5 generates paths of lengths at most $(m + 2) + (m + 1)(2^m + 1) + 1 = m2^m + 2^m + 2m + 4$. □

Theorem 2 *In an HHC$_{2^m+m}$, given a node $\boldsymbol{s}$ and a set of k ($k \leq m + 1$) nodes $D = \{\boldsymbol{d}_1, \ldots, \boldsymbol{d}_k\}$, we can find k disjoint paths $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}_i, 1 \leq i \leq k$ of*

lengths at most $m2^m + 2^m + 2m + 4$ *in* $O(km2^m)$ *time complexity.*

Proof: It can be deduced from Lemmas 4, 5, 6, 8, 9 and 10. □

We should note that this theoretical maximum path length is not reachable. Let us consider a cube-level path $\sigma_0 \to \sigma_1 \to \ldots \to \sigma_\lambda$, and let $b_i = \log_2(\sigma_{i-1} \oplus \sigma_i), 1 \leq i \leq \lambda$. Assume each subcube routing inside $Q_m(\sigma_i)$ requires m internal edges, then $H(b_i, b_{i+1}) = m$. Hence $b_1 = b_3 = b_5 = \ldots$ and $b_2 = b_4 = b_6 = \ldots$ hold. It means $\sigma_0 = \sigma_4$ which indicates the presence of a cycle inside that path, which is a contradiction. Therefore at least one subcube routing takes less than m edges.

Regarding the lower bound for the maximum path length and time complexity, assume $H(s_0, s_1) = 2^m$ for $\boldsymbol{d}_1 = (s_1, p_1)$, and $H(s_0, s_i) = 2^m - 1$ for other $\boldsymbol{d}_i = (s_i, p_i), 2 \leq i \leq k$. Then $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}_1$ has at least 2^m external edges and at least $2^m - 1$ internal edges. Hence the lower bound of the maximum path length is $2 \cdot 2^m - 1$. Also the paths $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}_i, 2 \leq i \leq k$ have at least $2^m - 1$ external edges and at least $2^m - 1$ internal edges because we should count at least one internal edge inside $Q_m(s_0)$. Hence k paths $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}_i, 1 \leq i \leq k$ include at least $2 \cdot 2^m - 1 + (k-1)(2 \cdot 2^m - 2) = 2k2^m - 2k + 1$ edges in total. Therefore the lower bound of the time complexity is $\Omega(k2^m)$.

As an example, we solve a node-to-set disjoint-path routing problem inside an HHC_{11} ($m = 3$), using the previously introduced HHC-N2S algorithm. For clarity reasons, we represent all numbers in binary format. Let the source node be $\boldsymbol{s} = (00000000, 000)$ and let the set of destination nodes be $D = \{\boldsymbol{d}_1 = (00001010, 000), \boldsymbol{d}_2 = (00001010, 001), \boldsymbol{d}_3 = (00111000, 100), \boldsymbol{d}_4 = (10000010, 010)\}$. One should note that $\boldsymbol{d}_1$ and $\boldsymbol{d}_2$ are inside the same subcube $Q_m(00001010)$ and will thus need to be distributed to $\boldsymbol{d}'_1$ and $\boldsymbol{d}'_2$, respectively. The disjoint paths returned by HHC-N2S are given in Table 5.1.

5.4 Empirical evaluation

In this section we empirically measure the algorithm HHC-N2S described in Section 5.2 to inspect its practical behaviour. The algorithm has been implemented using the Scheme functional programming language under the development environment DrScheme 4.2.5 [32, 33, 34] (now DrRacket).

First we measured the average execution time of this algorithm for different values of m from 2 to 9. Then, for each value of m, we measured the average and the maximum of all the maximal path lengths, each maximal length being collected when solving one routing problem for this value of m.

Practically, we solved 10,000 routing problems for each value of m. We bounded m by $2 \leq m \leq 9$, that is using natural integers of up to $2^9 = 512$ bits and routing inside perfect hierarchical hypercubes as large as an HHC_{521}. Such big integers are natively handled by Scheme as the integer

Table 5.1: Routing example inside an HHC_{11}.

Cube-level path	HHC-level path	Cube-level path	HHC-level path
00000000	(00000000, 000) $\boldsymbol{s}$	00000000	(00000000, 000) $\boldsymbol{s}$
00000001	(00000001, 000)		(00000000, 010)
	(00000001, 001)		(00000000, 011)
	(00000001, 101)	00001000	(00001000, 011)
00100001	(00100001, 101)		(00001000, 001) $\boldsymbol{d}'_2$
	(00100001, 001)	00001010	(00001010, 001) $\boldsymbol{d}_2$
00100011	(00100011, 001)		
	(00100011, 011)		
00101011	(00101011, 011)		
	(00101011, 001)		
	(00101011, 101)		
00001011	(00001011, 101)		
	(00001011, 100)		
	(00001011, 000) $\boldsymbol{d}'_1$		
00001010	(00001010, 000) $\boldsymbol{d}_1$		

Cube-level path	HHC-level path	Cube-level path	HHC-level path
00000000	(00000000, 000) $\boldsymbol{s}$	00000000	(00000000, 000) $\boldsymbol{s}$
	(00000000, 100)		(00000000, 001)
	(00000000, 110)		(00000000, 101)
01000000	(01000000, 110)		(00000000, 111)
	(01000000, 111)	10000000	(10000000, 111)
	(01000000, 011)		(10000000, 101)
01001000	(01001000, 011)		(10000000, 001)
	(01001000, 010)	10000010	(10000010, 001)
	(01001000, 000)		(10000010, 000)
	(01001000, 100)		(10000010, 010) $\boldsymbol{d}_4$
01011000	(01011000, 100)		
	(01011000, 101)		
01111000	(01111000, 101)		
	(01111000, 100)		
	(01111000, 110)		
00111000	(00111000, 110)		
	(00111000, 100) $\boldsymbol{d}_3$		

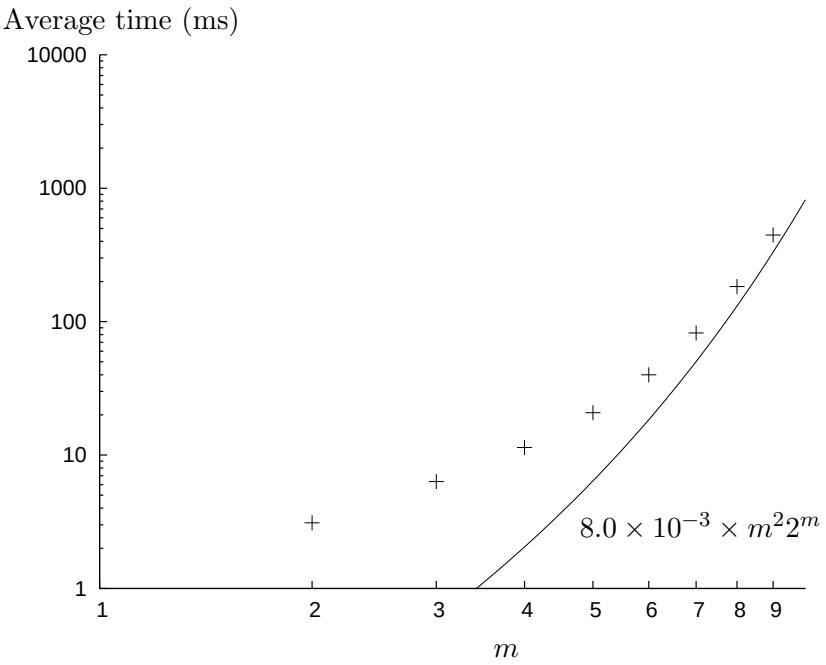

Figure 5.7: Average execution time for each value of m, in milliseconds (logarithmic scale).

datatype. The source node and the destination nodes were generated randomly and are always distinct. The number of destination nodes k is always set to $m + 1$.

Figure 5.7 illustrates for each value of m the average time in milliseconds required to solve a node-to-set disjoint-path routing problem. We see the measured average time converges to $O(m^2 2^m)$ time complexity. Figure 5.8 then illustrates the average and maximum maximal path length for each value of m. The theoretical maximum path length of the algorithm HHC-N2S $m2^m + 2^m + 2m + 4$ is also represented for comparison. As m increases, the probability to generate a path of maximum length decreases, which explains the divergence between the results and the theoretical maximum path length.

5.5 Improvement

We understand that the length of the paths generated by the algorithm HHC-N2S of Section 5.2 directly depends on the length the cube-level paths when performing routing inside Q_{2^m}. In other words, the number of external edges in a path generated by HHC-N2S is represented by the length of the corresponding cube-level path in Q_{2^m}. Hence, a natural approach to

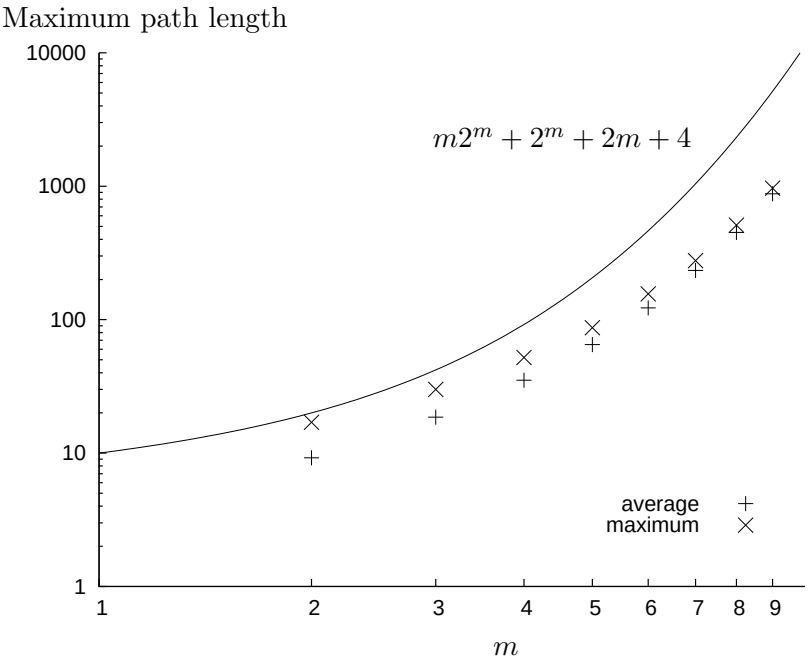

Figure 5.8: Average and maximum of the maximal path lengths collected for each value of m (logarithmic scale).

shorten the HHC-level paths generated by HHC-N2S is to reduce the cube-level paths lengths, that is the number of external edges in an HHC-level path. We proposed in [9] an improvement to the algorithm HHC-N2S so that the number of external edges in a path is significantly reduced. That is, we described a new hypercube node-to-set disjoint-path routing algorithm optimizing the paths lengths by using when possible a shortest-path routing algorithm flipping bits according to a Gray Code order [44]. Following an m-bit Gray Code when flipping bits in a shortest-path routing allows us to traverse all the 2^m bit positions of a node address in a Q_{2^m} using at most $2^m - 1$ bit flips, thus reducing the number of external edges required. Concretely, such a shortest-path routing finds a shortest path between any two nodes a, b in a hypercube with $H(a, b) = h$ and $a \oplus b = \sum_{i=0}^{h-1} 2^{\delta_i}$ by successively flipping the bits at the positions $\delta_0, \delta_1, \ldots, \delta_{h-1}$. This improvement idea has been detailed, and its effects onto HHC node-to-set disjoint-path routing have been empirically measured, in [9].

5.6 Summary

We have introduced in this chapter a routing algorithm HHC-N2S solving the node-to-set disjoint-path routing problem in perfect hierarchical hypercubes.

Inside an HHC_{2^m+m}, given a source node and a set of k ($k \leq m+1$) destination nodes, HHC-N2S finds k disjoint paths between the source node and the destination nodes in $O(km2^m)$ time complexity. The generated paths have lengths of at most $m2^m + 2^m + 2m + 4$. An improvement idea aiming at shortening paths lengths has also been briefly introduced.

Chapter 6

Node-to-set disjoint-path routing in metacubes

In this chapter we describe a node-to-set disjoint-path routing algorithm MC-N2S in metacubes. This part is organized as follows. Section 6.1 introduces or recalls several definitions and lemmas. Then Section 6.2 formally describes the algorithm MC-N2S and gives its pseudocode. The proof of the correctness of MC-N2S as well as its complexities are addressed in Section 6.3. Also, an example of the execution of MC-N2S is given. An empirical evaluation of MC-N2S is performed in Section 6.4. Section 6.5 summarizes this chapter.

6.1 Preliminaries

The problem of finding fault-free node-disjoint paths from a set of source nodes to a set of destination nodes is known as the fault-tolerant set-to-set disjoint-path routing problem. An algorithm Cube-S2S solving this problem inside a hypercube [46] is recalled in Lemma 11.

Lemma 11 *In a Q_n, given a set of k ($k \leq n$) non-faulty source nodes $S = \{s_1, s_2, \ldots, s_k\}$, a set of k non-faulty destination nodes $D = \{d_1, d_2, \ldots, d_k\}$ and a set F of at most $n-k$ faulty nodes, we can find k fault-free disjoint paths between the source and destination nodes of lengths at most $n+k$ in $O(kn \log k)$ time complexity.*

As will be described in Section 6.2, we may need to distribute destination nodes so that each k-cube contains at most one destination node. Hence we introduce the following lemma.

Lemma 12 *In an $MC(k,m)$, for a set of $k+m$ nodes $D = \{\boldsymbol{d}_1, \boldsymbol{d}_2, \ldots, \boldsymbol{d}_{k+m}\}$, we can find $k+m$ disjoint paths $\boldsymbol{d}_i \rightsquigarrow \boldsymbol{d}'_i$ of lengths at most two such that $\bar{c}(\boldsymbol{d}'_i) \neq \bar{c}(\boldsymbol{d}'_j)$ for any $j(\neq i)$, and $\bar{c}(\boldsymbol{d}'_i) \neq \bar{c}(\boldsymbol{d}_j)$ for any $j(\neq i)$.*

Proof For any $\boldsymbol{d}_i \in D$, if $\bar{c}(\boldsymbol{d}_i) \neq \bar{c}(\boldsymbol{d}_j)$ for any $j(\neq i)$, we can choose $\boldsymbol{d}_i$ as $\boldsymbol{d}'_i$. Otherwise, we search $\boldsymbol{d}'_i$ among $\boldsymbol{d}_i^{(j)}$ $(0 \leq j \leq m-1)$ and $\boldsymbol{d}_i^{(j,l)}$ $(m \leq j \leq k+m-1,\ 0 \leq l \leq m-1)$. See Figure 6.1. If $\boldsymbol{d}'_i$ is selected from $\boldsymbol{d}_i^{(j)}$ $(0 \leq j \leq m-1)$, the conditions $c(\boldsymbol{d}_i) = c(\boldsymbol{d}'_i)$ and $H(\bar{c}(\boldsymbol{d}_i), \bar{c}(\boldsymbol{d}'_i)) = 1$ hold. Otherwise, the condition $H(\bar{c}(\boldsymbol{d}_i), \bar{c}(\boldsymbol{d}'_i)) = H(c(\boldsymbol{d}_i), c(\boldsymbol{d}'_i)) = 1$ holds.

Now, let $M(\boldsymbol{d}_i) = \{\boldsymbol{d}_i^{(j)} \mid 0 \leq j \leq m-1\}$, $K(\boldsymbol{d}_i) = \{\boldsymbol{d}_i^{(j)} \mid m \leq j \leq k+m-1\}$ and $\bar{c}(M(\boldsymbol{d}_i)) = \{\bar{c}(\boldsymbol{u}) \mid \boldsymbol{u} \in M(\boldsymbol{d}_i)\}$, $\bar{c}(K(\boldsymbol{d}_i)) = \{\bar{c}(\boldsymbol{u}) \mid \boldsymbol{u} \in K(\boldsymbol{d}_i)\}$. For any $\boldsymbol{d}_p \in D \setminus \{\boldsymbol{d}_i\}$, since $\boldsymbol{d}'_p$ is selected so that $\bar{c}(\boldsymbol{d}'_p) \neq \bar{c}(\boldsymbol{d}_i)$, and since for any $\boldsymbol{u} \in K(\boldsymbol{d}_i)$ we have $\bar{c}(\boldsymbol{u}) = \bar{c}(\boldsymbol{d}_i)$, then $\bar{c}(\boldsymbol{d}'_p) \notin \bar{c}(K(\boldsymbol{d}_i))$. For any two nodes $\boldsymbol{u} \in K(\boldsymbol{d}_i)$ and $\boldsymbol{v} \in M(\boldsymbol{d}_i)$, there is no path of length at most two between them other than $\boldsymbol{u} \to \boldsymbol{d}_i \to \boldsymbol{v}$. Therefore, if $\bar{c}(\boldsymbol{d}_p) \in \bar{c}(K(\boldsymbol{d}_i))$, $\bar{c}(\boldsymbol{d}'_p)$ cannot be included $\bar{c}(M(\boldsymbol{d}_i))$. For any distinct nodes $\boldsymbol{u}, \boldsymbol{v} \in M(\boldsymbol{d}_i)$, $H(\bar{c}(\boldsymbol{u}), \bar{c}(\boldsymbol{v})) = 2$. Hence, $\bar{c}(\boldsymbol{d}_p)$ and $\bar{c}(\boldsymbol{d}'_p)$ cannot be included in $\bar{c}(M(\boldsymbol{d}_i))$ simultaneously since $H(\bar{c}(\boldsymbol{d}_p), \bar{c}(\boldsymbol{d}'_p)) = 1$. Thus for any $\boldsymbol{d}_p \in D \setminus \{\boldsymbol{d}_i\}$, $\bar{c}(\boldsymbol{d}_p)$ and $\bar{c}(\boldsymbol{d}'_p)$ cannot be included in $\bar{c}(M(\boldsymbol{d}_i)) \cup \bar{c}(K(\boldsymbol{d}_i))$ simultaneously if $\bar{c}(\boldsymbol{d}_p) \neq \bar{c}(\boldsymbol{d}'_p)$.

If there is a node $\boldsymbol{u} \in M(\boldsymbol{d}_i)$ such that $\bar{c}(\boldsymbol{u}) \neq \bar{c}(\boldsymbol{d}_p)$ and $\bar{c}(\boldsymbol{u}) \neq \bar{c}(\boldsymbol{d}'_p)$ for any node $\boldsymbol{d}_p \in D \setminus \{\boldsymbol{d}_i\}$, we can select $\boldsymbol{u}$ as $\boldsymbol{d}'_i$. Otherwise, from the above discussion, there is at least one node $\boldsymbol{d}_i^{(j^*)} \in K(\boldsymbol{d}_i)$ such that $\bar{c}(\boldsymbol{d}_i^{(j^*)}) \neq \bar{c}(\boldsymbol{d}_p)$ and $\bar{c}(\boldsymbol{d}_i^{(j^*)}) \neq \bar{c}(\boldsymbol{d}'_p)$ for any node $\boldsymbol{d}_p \in D \setminus \{\boldsymbol{d}_i\}$.

For a node $\boldsymbol{u} \in M(\boldsymbol{d}_i)$ and a node $\boldsymbol{d}_i^{(j,l)} \in \bigcup_{\boldsymbol{v} \in K(\boldsymbol{d}_i)} M(\boldsymbol{v})$, $H(\bar{c}(\boldsymbol{u}), \bar{c}(\boldsymbol{d}_i^{(j,l)})) = 2$. Therefore, if $\bar{c}(\boldsymbol{d}_p)$ (or $\bar{c}(\boldsymbol{d}'_p)$) $\in \bar{c}(M(\boldsymbol{d}_i))$, $\bar{c}(\boldsymbol{d}'_p)$ (or $\bar{c}(\boldsymbol{d}_p)$) cannot be included in $\{\bar{c}(\boldsymbol{d}_i^{(j,l)}) \mid \boldsymbol{d}_i^{(j,l)} \in \bigcup_{\boldsymbol{v} \in K(\boldsymbol{d}_i)} M(\boldsymbol{v})\}$. Furthermore, for two distinct nodes $\boldsymbol{d}_i^{(j_1)}, \boldsymbol{d}_i^{(j_2)} \in K(\boldsymbol{d}_i)$, $H(c(\boldsymbol{d}_i^{(j_1)}), c(\boldsymbol{d}_i^{(j_2,l)})) = 2$ for any $\boldsymbol{d}_i^{(j_2,l)} \in M(\boldsymbol{d}_i^{(j_2)})$. Hence, if $\bar{c}(\boldsymbol{d}_p) = \bar{c}(\boldsymbol{d}_i^{(j_1)})$ for some $\boldsymbol{d}_i^{(j_1)} \in K(\boldsymbol{d}_i)$, $\bar{c}(\boldsymbol{d}'_p)$ cannot be included in $\{\bar{c}(\boldsymbol{d}_i^{(j_2,l)}) \mid \boldsymbol{d}_i^{(j_2,l)} \in \bigcup_{\boldsymbol{v} \in K(\boldsymbol{d}_i)} M(\boldsymbol{v}), j_2 \neq j_1\}$.

Consequently, for the node $\boldsymbol{d}_i^{(j^*)}$ above, we can select a node $\boldsymbol{d}_i^{(j^*,l^*)} \in M(\boldsymbol{d}_i^{(j^*)})$ such that $\bar{c}(\boldsymbol{d}_i^{(j^*,l^*)}) \neq \bar{c}(\boldsymbol{d}_p)$ and $\bar{c}(\boldsymbol{d}_i^{(j^*,l^*)}) \neq \bar{c}(\boldsymbol{d}'_p)$ for any node $\boldsymbol{d}_p \in D \setminus \{\boldsymbol{d}_i\}$ as $\boldsymbol{d}'_i$.

We proved that we can find $k+m$ disjoint paths $\boldsymbol{d}_i \rightsquigarrow \boldsymbol{d}'_i$ of lengths at most two. □

6.2 Node-to-set disjoint-path routing algorithm

We describe in this section a node-to-set disjoint-path routing algorithm MC-N2S in a metacube. In an $MC(k,m)$, given a node $\boldsymbol{s}$ and a set of n $(n \leq k+m)$ nodes $D = \{\boldsymbol{d}_1, \boldsymbol{d}_2, \ldots, \boldsymbol{d}_n\}$, MC-N2S finds n node-disjoint paths $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}_j$ $(1 \leq j \leq n)$. The pseudocode of MC-N2S is given in Algorithm 7.

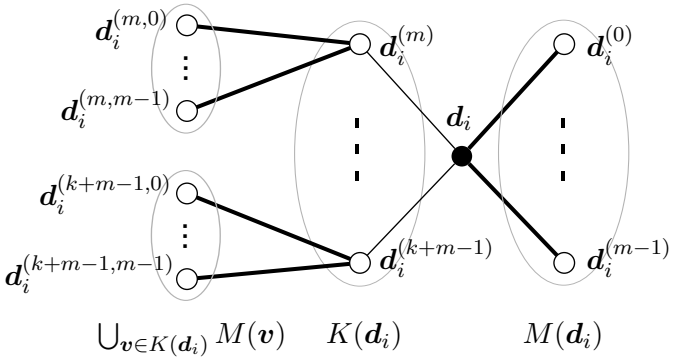

Figure 6.1: Distributed destination node candidates for $\boldsymbol{d}_i$.

The main idea is to reduce the metacube node-to-set disjoint-path routing problem to the hypercube fault-tolerant set-to-set disjoint-path routing problem. This is achieved with a 2^k-to-1 mapping of an $MC(k,m)$ onto an $m2^k$-cube Q_{m2^k}. Concretely we map each k-cube of an $MC(k,m)$ to a single node of a Q_{m2^k} as shown on Figure 6.2. On this figure, an $MC(2,1)$ is represented by grouping k-cubes each other, whereas Figure 3.5 groups clusters together. We change the focus to easily visualize the mapping operation of one k-cube to one node of a Q_{m2^k}.

Assume we use a node-to-set disjoint-path routing algorithm to find paths inside Q_{m2^k} (called cube-level paths). For example, in an $MC(2,2)$ with $\boldsymbol{s} = 00\ 00\ 00\ 00\ 00$, two cube-level paths could include the edges $\bar{c}(\boldsymbol{s}) \rightarrow 00\ 00\ 01\ 00\ (= q_1)$ and $\bar{c}(\boldsymbol{s}) \rightarrow 00\ 00\ 10\ 00\ (= q_2)$, respectively. Thus, when routing inside $Q_k(\boldsymbol{s})$ from $\boldsymbol{s}$ toward the k-cubes q_1 and q_2, the two paths would both include the node 01 00 00 00 00 and thus be not disjoint. Instead we use a set-to-set, rather than a node-to-set, disjoint-path routing algorithm in a hypercube to control the selection of the cube-level edges incident to $\bar{c}(\boldsymbol{s})$.

We now describe MC-N2S. Let $Q_k(\boldsymbol{s}) \cap D = \{\boldsymbol{d}_1, \boldsymbol{d}_2, \ldots, \boldsymbol{d}_r\}$ with $r \leq n$.

Step 1 If $r > k$, we apply Cube-N2S inside $Q_k(\boldsymbol{s})$ to connect $\boldsymbol{s}$ to $\boldsymbol{d}_1, \ldots, \boldsymbol{d}_k$. For every path $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}_j$ $(1 \leq j \leq k)$ produced that includes a node $\boldsymbol{d}_i$ $(k+1 \leq i \leq r)$, we discard its subpath $\boldsymbol{d}_i \rightsquigarrow \boldsymbol{d}_j$ and swap the indices of $\boldsymbol{d}_i$ and $\boldsymbol{d}_j$. Hence we can assume that the $r-k$ nodes $\boldsymbol{d}_{k+1}, \ldots, \boldsymbol{d}_r$ are not included in any of the paths $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}_i$ $(1 \leq i \leq k)$.

Step 2 We distribute each of the nodes $\boldsymbol{d}_j$ $(\min(k,r)+1 \leq j \leq n)$ with $|Q_k(\boldsymbol{d}_j) \cap D| > 1$, to a node $\boldsymbol{d}'_j$ (called distributed destination node for $\boldsymbol{d}_j$) with disjoint paths of length at most two, such that $\boldsymbol{d}'_j$ is inside a k-cube containing no other destination node $\boldsymbol{d}_i$ nor distributed destination node $\boldsymbol{d}'_i$ $(1 \leq i \leq n, i \neq j)$. Let $\tilde{D}$ be the set of all the distributed destination nodes $\boldsymbol{d}'_j$. Let D' be the set containing ini-

tially the destination nodes alone in their k-cube, plus the distributed destination nodes $\boldsymbol{d}'_j$ which are by definition similarly alone in their k-cube. We note that one destination node can be distributed into $Q_k(\boldsymbol{s})$ if $Q_k(\boldsymbol{s}) \cap D = \emptyset$. Let $D_s = Q_k(\boldsymbol{s}) \cap (D \cup \tilde{D})$ be the set of the destination nodes and distributed destination nodes of $Q_k(\boldsymbol{s})$. We remove from D' all the nodes of D_s. Formally,

$$D' = (\{\boldsymbol{d}_j \mid |Q_k(\boldsymbol{d}_j) \cap D| = 1\} \cup \tilde{D}) \setminus D_s$$

Step 3 We prepare the set of source nodes to be used by Cube-S2S in Step 2. We select $l = n - \min(k, |D_s|)$ nodes $\boldsymbol{s}_i$ with paths $\boldsymbol{s} \rightsquigarrow \boldsymbol{s}_i$ of lengths at most two such that the following conditions hold.

1. $\boldsymbol{s} \rightsquigarrow \boldsymbol{s}_i$ and $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}_j$ $(1 \leq j \leq \min(k, r))$ are disjoint
2. $\forall i, 1 \leq i \leq l, H(c(\boldsymbol{s}), c(\boldsymbol{s}_i)) \leq H(\bar{c}(\boldsymbol{s}), \bar{c}(\boldsymbol{s}_i)) = H(m_{c(\boldsymbol{s}_i)}(\boldsymbol{s}), m_{c(\boldsymbol{s}_i)}(\boldsymbol{s}_i)) = 1$
3. $\forall i, 1 \leq i \leq l, \forall \boldsymbol{u} \in D_s \setminus \{\boldsymbol{s}\}, c(\boldsymbol{s}_i) \neq c(\boldsymbol{u})$
4. $\forall i, j, 1 \leq i, j \leq l, i \neq j$, $c(\boldsymbol{s}_i) \neq c(\boldsymbol{s}_j)$ or $c(\boldsymbol{s}_i) = c(\boldsymbol{s}_j) = c(\boldsymbol{s})$
5. $\forall i, 1 \leq i \leq l$, if $\boldsymbol{s}(= \boldsymbol{d}'_j) \in D_s$ then $\boldsymbol{s}_i \notin (\boldsymbol{d}'_j \rightsquigarrow \boldsymbol{d}_j)$

Condition 1 forbids the inclusion of a neighbour of $\boldsymbol{s}$ included in a path $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}_j$ $(1 \leq j \leq \min(k, r))$ inside a path $\boldsymbol{s} \rightsquigarrow \boldsymbol{s}_i$. Condition 2 represents a set of k nodes $\boldsymbol{s}_i$ distant from one edge to $\boldsymbol{s}$ where $c(\boldsymbol{s}) = c(\boldsymbol{s}_i)$ and $H(m_{c(\boldsymbol{s}_i)}(\boldsymbol{s}), m_{c(\boldsymbol{s}_i)}(\boldsymbol{s}_i)) = 1$, and of m nodes $\boldsymbol{s}_i$ distant from two edges to $\boldsymbol{s}$ where $H(c(\boldsymbol{s}), c(\boldsymbol{s}_i)) = 1$ and $H(m_{c(\boldsymbol{s}_i)}(\boldsymbol{s}), m_{c(\boldsymbol{s}_i)}(\boldsymbol{s}_i)) = 1$. Condition 3 ensures that the $\boldsymbol{s}_i$ nodes are not linked to $\boldsymbol{s}$ with a path containing a destination node or a distributed destination node of $Q_k(\boldsymbol{s})$. Condition 4 guarantees that each node $\boldsymbol{s}_i$ is located inside a distinct k-cube. Finally, Condition 5 is to avoid a collision in the special case where a destination node $\boldsymbol{d}_j$ is distributed onto a node $\boldsymbol{d}'_j$ with $\boldsymbol{d}'_j = \boldsymbol{s}$. In such case, Condition 5 ensures that no $\boldsymbol{s}_i$ will be on the path $\boldsymbol{d}'_j \rightsquigarrow \boldsymbol{d}_j$.

Note that all cube-level nodes $\bar{c}(\boldsymbol{s}_i)$ are adjacent to the cube-level node $\bar{c}(\boldsymbol{s})$. Let S be the set of all the cube-level nodes $\bar{c}(\boldsymbol{s}_i)$.

Step 4 Let F be the set of all the k-cubes containing at least two destination nodes, plus the k-cube $Q_k(\boldsymbol{s})$. Formally,

$$F = \{\bar{c}(\boldsymbol{d}_i) \mid \boldsymbol{d}_i \in D, |Q_k(\boldsymbol{d}_i) \cap D| \geq 2\} \cup \{\bar{c}(\boldsymbol{s})\}$$

Now for every $\boldsymbol{s}_i$ with $|Q_k(\boldsymbol{s}_i) \cap D| > 1$, we connect $\boldsymbol{s}_i$ to the closest destination node $\boldsymbol{d}_j$ of $Q_k(\boldsymbol{s}_i) \cap D$ with an SPR, remove the corresponding node $\boldsymbol{d}'_j$ from D' and $\bar{c}(\boldsymbol{s}_i)$ from S, and discard the path $\boldsymbol{d}_j \rightsquigarrow \boldsymbol{d}'_j$. Only after this task complete, we can remove from F the elements of S if any, that is $F = F \setminus S$.

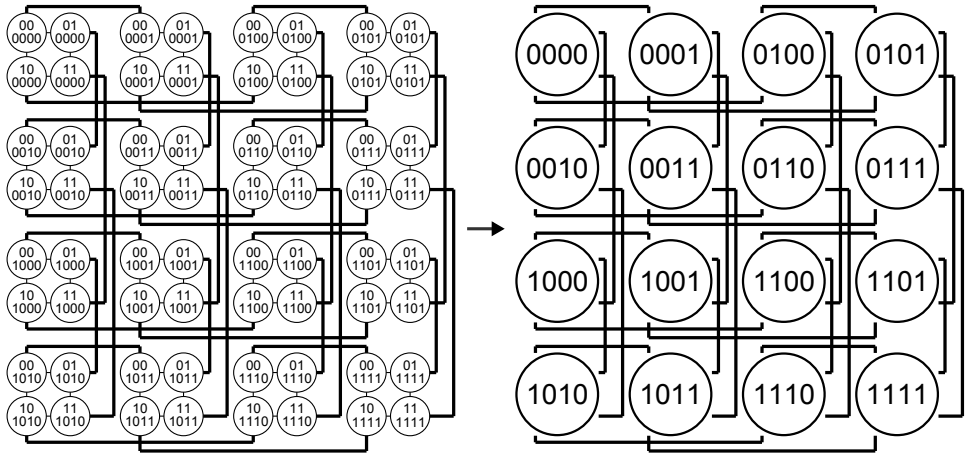

Figure 6.2: Mapping an $MC(2,1)$ onto a Q_4.

Step 5 We apply Cube-S2S in Q_{m2^k} to connect each $\bar{c}(\boldsymbol{s}_i)$ to one distinct $\bar{c}(\boldsymbol{d}'_j)$ ($\boldsymbol{d}'_j \in D'$), avoiding faulty nodes of F.

Step 6 We convert the cube-level paths back to paths in $MC(k,m)$ (called metacube-level paths) by performing routings inside each k-cube traversed by a cube-level path generated in Step 5.

First we route inside $Q_k(\boldsymbol{s})$. If $r > k$ then the local routing inside $Q_k(\boldsymbol{s})$ has already been performed in Step 1a. If $r \leq k$, let Z be the set containing the destination nodes and distributed destination nodes of $Q_k(\boldsymbol{s})$ plus the nodes $\boldsymbol{s}'_i \in Q_k(\boldsymbol{s})$ of the paths $(\boldsymbol{s} \rightarrow \boldsymbol{s}'_i \rightarrow \boldsymbol{s}_i) \in \mathcal{S}$ of lengths exactly two selected in Step 2 (i.e. $c(\boldsymbol{s}'_i) = c(\boldsymbol{s}_i) \neq c(\boldsymbol{s})$). Formally,

$$Z = D_s \cup \{\boldsymbol{s}'_i \mid (\boldsymbol{s} \rightarrow \boldsymbol{s}'_i \rightarrow \boldsymbol{s}_i) \in \mathcal{S}\}$$

We apply Cube-N2S inside $Q_k(\boldsymbol{s})$ to connect $\boldsymbol{s}$ to each of the n elements of Z. Note that $\boldsymbol{s}$ can always be connected to $\boldsymbol{s}'_i$ in one edge. If Cube-N2S generates another (longer) path to $\boldsymbol{s}'_i$, it is directly replaced by the trivial path $\boldsymbol{s} \rightarrow \boldsymbol{s}'_i$ of length one. This concludes routing inside $Q_k(\boldsymbol{s})$.

Inside the other k-cubes, we apply an SPR as follows. We assume without loss of generality that P_i is a cube-level path connecting $\bar{c}(\boldsymbol{s}_i)$ and $\bar{c}(\boldsymbol{d}'_i)$. Formally, for each path P_i: $q_{i,0}(= \bar{c}(\boldsymbol{s}_i)) \rightarrow q_{i,1} \rightarrow \ldots \rightarrow q_{i,\lambda_i-1} \rightarrow q_{i,\lambda_i}(= \bar{c}(\boldsymbol{d}'_i))$, let $c_{i,j}$ be a k-bit classID that satisfies $H(m_{c_{i,j}}((c_{i,j}, q_{i,j-1})), m_{c_{i,j}}((c_{i,j}, q_{i,j}))) = 1$. Hence for each cube-level path P_i we construct the corresponding metacube-level path $(c_{i,1}, q_{i,0}) \rightarrow (c_{i,1}, q_{i,1}) \overset{\text{SPR}}{\rightsquigarrow} (c_{i,2}, q_{i,1}) \rightarrow \ldots \rightarrow (c_{i,\lambda_i-1}, q_{i,\lambda_i-1}) \overset{\text{SPR}}{\rightsquigarrow} (c_{i,\lambda_i}, q_{i,\lambda_i-1}) \rightarrow (c_{i,\lambda_i}, q_{i,\lambda_i}) = (c_{i,\lambda_i}, \bar{c}(\boldsymbol{d}'_i)) \overset{\text{SPR}}{\rightsquigarrow} (c(\boldsymbol{d}'_i), \bar{c}(\boldsymbol{d}'_i)) = \boldsymbol{d}'_i$ using an SPR inside each k-cube of P_i to connect every $(c_{i,j}, q_{i,j})$ to

$(c_{i,j+1}, q_{i,j})$. The path is completed by connecting $\boldsymbol{s}_i$ to $(c_{i,1}, q_{i,0} = \bar{c}(\boldsymbol{s}_i))$ again with an SPR, and by lastly joining it to the subpath $\boldsymbol{d}'_i \rightsquigarrow \boldsymbol{d}_i$ of length at most two. See Figure 6.3.

6.3 Correctness and complexities

In this section, we prove the correctness of MC-N2S and estimate its time complexity as well as the theoretical maximum length of the generated paths. As previously, let $|D| = n \leq k + m$.

Lemma 13 *In $Q_k(\boldsymbol{s})$, Step 1a finds k disjoint paths $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}_j$ $(1 \leq j \leq k)$ of lengths at most $k + 1$ in $O(k^2 n)$ time complexity.*

Proof By Lemma 2, the paths generated inside $Q_k(\boldsymbol{s})$ by Cube-NS2 in $O(k^2)$ time complexity are disjoint and of lengths at most $k + 1$. Checking if one path generated by Cube-N2S include other destination nodes requires $O(kn)$ time complexity. Hence checking the k paths requires $O(k^2 n)$ time complexity. □

Lemma 14 *Step 1b generates disjoint paths $\boldsymbol{d} \rightsquigarrow \boldsymbol{d}'_j$ of lengths at most two in $O((k + m)n^2)$ time complexity.*

Proof By Lemma 12 we can find disjoint paths of length at most two for the distribution of the nodes $\boldsymbol{d}_j$ $(\min(k, r) + 1 \leq j \leq n)$ with $|Q_k(\boldsymbol{d}_j) \cap D| > 1$. For each destination node, there are $k+m$ candidate k-cubes for hosting the corresponding distributed destination node. Because we need to find candidate k-cubes not including any destination node or distributed destination node, each destination node can thus be distributed in $O((k + m)n)$ time. Hence in total, the destination node distribution task requires $O((k+m)n^2)$ time complexity. □

Lemma 15 *Step 1c generates disjoint paths $\boldsymbol{s} \rightsquigarrow \boldsymbol{s}_i$ of lengths at most two in $O(l)$ time complexity.*

Proof Let us show that we can always find as many $\boldsymbol{s}_i$ nodes as needed by the algorithm.

Starting from $\boldsymbol{s}$ and using at most two edges, by definition of $MC(k, m)$ we can reach $m + k$ distinct k-cubes, each of them being a candidate to host one $\boldsymbol{s}_i$. One of these k-cubes must be removed from the candidates list as soon as it has been chosen to host one $\boldsymbol{s}_i$ node. Also, it is important to note that a destination node inside $Q_k(\boldsymbol{s})$ being neighbour of $\boldsymbol{s}$ can block the access to at most one k-cube. In total, there are at most k such blocking destination nodes since $Q_k(\boldsymbol{s})$ is a k-dimensional hypercube. However, for

Algorithm 7 MC-N2S($MC(k,m)$, $\boldsymbol{s}$, D)

Input: An $MC(k,m)$, a source node $\boldsymbol{s}$ and a set of n ($n \leq k+m$) destination nodes $D = \{\boldsymbol{d}_1, \boldsymbol{d}_2, \ldots, \boldsymbol{d}_n\}$.
Output: n disjoint paths $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}_i$ ($1 \leq i \leq n$). The binary exclusive-or operation is denoted $\oplus$.

$R := \emptyset$; $\{\boldsymbol{d}_1, \boldsymbol{d}_2, \ldots, \boldsymbol{d}_r\} := Q_k(\boldsymbol{s}) \cap D$;
/* *Step 1 - Preprocessing* */
if $r > k$ **then** /* *Step 1a* */
  $R' :=$ Cube-N2S($Q_k(\boldsymbol{s})$, $\boldsymbol{s}$, $\{\boldsymbol{d}_1, \boldsymbol{d}_2, \ldots, \boldsymbol{d}_k\}$);
  For each $\boldsymbol{d}_i$ ($k+1 \leq i \leq r$) included inside a path $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}_j$ of R', discard the subpath $\boldsymbol{d}_i \rightsquigarrow \boldsymbol{d}_j$ and swap the indices of $\boldsymbol{d}_i, \boldsymbol{d}_j$;
end if
$\tilde{D} := \emptyset$; /* *Step 1b* */
for all $\boldsymbol{d}_j, min(k,r)+1 \leq i,j \leq n, |Q_k(\boldsymbol{d}_j) \cap D| > 1$ **do**
  Find a path $\boldsymbol{d}_j \rightsquigarrow \boldsymbol{d}'_j$ of at most two edges such that
    $\forall(\boldsymbol{d}_i, \boldsymbol{d}_j), i \neq j, \bar{c}(\boldsymbol{d}_i) = \bar{c}(\boldsymbol{d}_j), c(\boldsymbol{d}'_i) \neq c(\boldsymbol{d}'_j)$ and
    $\forall \boldsymbol{d}'_j, Q_k(\boldsymbol{d}'_j) \cap D = \emptyset$ and
    $\forall \boldsymbol{d}'_i, \boldsymbol{d}'_j, i \neq j, \boldsymbol{d}'_i \notin Q_k(\boldsymbol{d}'_j)$
  hold;
  $\tilde{D} := \tilde{D} \cup \{\boldsymbol{d}'_j\}$
end for
$D' = (\{\boldsymbol{d}_j \mid |Q_k(\boldsymbol{d}_j) \cap D| = 1\} \cup \tilde{D}) \setminus Q_k(\boldsymbol{s})$;
$D_s := Q_k(\boldsymbol{s}) \cap (D \cup \tilde{D})$; /* *Step 1c* */
$l := n - \min(k, |D_s|)$;
$\{c_1, c_2, \ldots, c_z\} := \{c(\boldsymbol{s}) \oplus 2^i, 0 \leq i \leq k-1\} \setminus \{c(\boldsymbol{u}) \mid \boldsymbol{u} \in D_s \setminus \{\boldsymbol{s}\}\}$;
if $\boldsymbol{s}(= \boldsymbol{d}'_j) \in D_s$ **then** $X := \{(\boldsymbol{d}'_j \rightsquigarrow \boldsymbol{d}_j)\}$ **else** $X := \emptyset$ **end if**
$C_1 := \{\boldsymbol{s}_i \mid \forall i, 1 \leq i \leq m, \boldsymbol{s}_i := (c(\boldsymbol{s}), m_{h-1}, \ldots, m_{c(\boldsymbol{s})} \oplus 2^{i-1}, \ldots, m_0), \boldsymbol{s}_i \notin P, P \in X\}$;
$C_2 := \{\boldsymbol{s}_i \mid \forall i, 1 \leq i \leq z, \boldsymbol{s}_i := (c_i, m_{h-1}, \ldots, m_{c_i} \oplus 2^0, \ldots, m_0)\}$;
Take l elements $\{\boldsymbol{s}_1, \ldots, \boldsymbol{s}_l\}$ inside the set $C_1 \cup C_2$;
$S := \{\bar{c}(\boldsymbol{s}_i) \mid \boldsymbol{s}_i \in \tilde{S}\}$; /* *Step 1d* */
$F := \{\bar{c}(\boldsymbol{u}) \mid \boldsymbol{u} \in MC(k,m), |Q_k(\boldsymbol{u}) \cap D| > 1\} \cup \{\bar{c}(\boldsymbol{s})\}$;
$R'' := \emptyset$;
for all $\boldsymbol{s}_i$ with $|Q_k(\boldsymbol{s}_i) \cap D| > 1$ **do**
  $R'' := R'' \cup \{$(connect $\boldsymbol{s}_i$ to the closest destination node $\boldsymbol{d}_j$ of $Q_k(\boldsymbol{s}_i) \cap D$)$\}$;
  $D' := D' \setminus \{\boldsymbol{d}'_j\}$;
  $S := S \setminus \{\bar{c}(\boldsymbol{s}_i)\}$
end for
$F := F \setminus S$;
/* *Step 2 - Set-to-set disjoint-path routing in a* Q_{m2^k} */
$\{P_1, P_2, \ldots, P_{l-|R''|}\} :=$ Cube-S2S(Q_{m2^k}, S, $\bar{c}(D')$, F);
/* *Step 3 - Cube-level paths to metacube-level paths conversion* */
if $r \leq k$ **then**
  $Z := D_s \cup \{\boldsymbol{s}'_i \mid \boldsymbol{s} \rightarrow \boldsymbol{s}'_i \rightarrow \boldsymbol{s}_i\}$;
  $R' :=$ Cube-N2S($Q_k(\boldsymbol{s})$, $\boldsymbol{s}$, Z)
end if
for all P_i ($1 \leq i \leq l - |R''|$): $q_{i,0}(= \bar{c}(\boldsymbol{s}_i)) \rightarrow q_{i,1} \rightarrow \ldots \rightarrow q_{i,\lambda_i - 1} \rightarrow q_{i,\lambda_i}(= \bar{c}(\boldsymbol{d}'_i))$ **do**
  Compute all $c_{i,j}$ such that $H(m_{c_{i,j}}((c_{i,j}, q_{i,j-1})), m_{c_{i,j}}((c_{i,j}, q_{i,j}))) = 1$;
  $P_i := (c_{i,1}, q_{i,0}) \rightarrow (c_{i,1}, q_{i,1}) \overset{\text{SPR}}{\rightsquigarrow} \ldots \overset{\text{SPR}}{\rightsquigarrow} (c_{i,\lambda_i}, q_{i,\lambda_i - 1}) \rightarrow (c_{i,\lambda_i}, q_{i,\lambda_i}) = (c_{i,\lambda_i}, \bar{c}(\boldsymbol{d}'_i)) \overset{\text{SPR}}{\rightsquigarrow} (c(\boldsymbol{d}'_i), \bar{c}(\boldsymbol{d}'_i)) = \boldsymbol{d}'_i$;
  $P_i := (\boldsymbol{s} \rightsquigarrow \boldsymbol{s}_i) \in R' \rightarrow (\boldsymbol{s}_i \overset{\text{SPR}}{\rightsquigarrow} (c_{i,1}, q_{i,0})) \rightarrow P_i$;
  $P_i := P_i \rightarrow (\boldsymbol{d}'_i \rightsquigarrow \boldsymbol{d}_i)$;
  $R := R \cup \{P_i\}$
end for
for all $(\boldsymbol{s}_i \rightsquigarrow \boldsymbol{d}_j) \in R''$ **do**
  $R := R \cup \{(\boldsymbol{s} \rightsquigarrow \boldsymbol{s}_i) \in R' \rightarrow (\boldsymbol{s}_i \rightsquigarrow \boldsymbol{d}_j)\}$
end for
$R' := R' \setminus \{\boldsymbol{s} \rightarrow \boldsymbol{s}'_i \mid 1 \leq i \leq l\}$;
if $R' = \{(\boldsymbol{s} \rightsquigarrow \boldsymbol{d}'_i \in \tilde{D})\}$ **then**
  return $R \cup X \cup \{(\boldsymbol{s} \rightsquigarrow \boldsymbol{d}'_i) \in R' \rightarrow (\boldsymbol{d}'_i \rightsquigarrow \boldsymbol{d}_i)\}$
else
  return $R \cup X \cup R'$
end if

every such blocking destination node, it is one $\boldsymbol{s}_i$ less needed since these blocking destination nodes shall be linked with routing in $Q_k(\boldsymbol{s})$ in Step 3. Hence we can assume without loss of generality that no destination of $Q_k(\boldsymbol{s})$ is blocking, that is $D \cap N(\boldsymbol{s}) \cap Q_k(\boldsymbol{s}) = \emptyset$ holds. Therefore, because at most $k+m$ nodes $\boldsymbol{s}_i$ are needed, and because there are $k+m$ available candidates, we can always select the required number of $\boldsymbol{s}_i$ nodes.

Step 1c selects l $(= n - \min(k, |D_s|))$ nodes $\boldsymbol{s}_i$. Let us consider the worst case: we assume there exists a path of length two $\boldsymbol{d}'_j (= \boldsymbol{s}) \rightsquigarrow \boldsymbol{d}_j$. By definition of the destination node distribution process, only the candidate $\boldsymbol{s}_i$ nodes distant of exactly one edge to $\boldsymbol{s}$ can be on that path and moreover, at most one of these candidates can be on that path. Let the path $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}_j$ be $\boldsymbol{s} \rightarrow \boldsymbol{s}^{(\alpha)} \rightarrow \boldsymbol{d}_j$, it thus takes constant time to select one $\boldsymbol{s}_i$ node distant of one edge to $\boldsymbol{s}$ by simply selecting one node from $\boldsymbol{s}^{(\beta)}$ $(0 \leq \beta \leq m-1, \alpha \neq \beta)$. Now regarding the $\boldsymbol{s}_i$ nodes candidates distant of exactly two edges to $\boldsymbol{s}$, there are km, but only k of them, one per class, can be selected as explained in Step 1b. Hence we can directly take one $\boldsymbol{s}_i$ node candidate per class in constant time. Hence in total Step 1c takes $O(l)$ time complexity. □

Lemma 16 *Step 1d generates disjoint paths $\boldsymbol{s} \rightsquigarrow \boldsymbol{s}_i \rightsquigarrow \boldsymbol{d}_j$ of lengths at most $k+2$ in $O(k)$ time complexity.*

Proof By Lemma 12, the path $\boldsymbol{s} \rightsquigarrow \boldsymbol{s}_i$ is disjoint with any other selected path and of length at most two. The node $\boldsymbol{s}_i$ is connected to the closest node $\boldsymbol{d}_j$ of $Q_k(\boldsymbol{s}_i) \cap D$ using an SPR, hence requiring at most k edges and $O(k)$ time complexity. □

Lemma 17 *Step 2 generates disjoint cube-level paths of lengths at most $m2^k + n \leq m2^k + k + m$ in $O(nm2^k \log n)$ time complexity.*

Proof Step 2 applies Cube-S2S in a Q_{m2^k}. As required by Cube-S2S, the number of faulty nodes plus the number of destination nodes must not exceed the dimension of the hypercube. We have at most $k+m$ destination nodes and at most $1 + \lfloor (k+m)/2 \rfloor$ faulty nodes as described in Step 1. It is easy to check that $(k+m) + (1 + \lfloor (k+m)/2 \rfloor) \leq m2^k$ holds for every k, m excepted in the three following cases: $k = m = 1$, $k = 2, m = 1$ and $k = 1, m = 2$. In the first case $k = m = 1$, the corresponding metacube is a cycle and it is then trivial to solve the node-to-set disjoint-path routing problem. In the second case $k = 2, m = 1$, the corresponding metacube is a perfect hierarchical hypercube HHC_6, and a node-to-set disjoint-path routing algorithm has already been proposed in Chapter 5. Finally, in the third case $k = 1, m = 2$, the corresponding metacube is a dual-cube D_3, and a node-to-set disjoint-path routing algorithm has already been described in [68].

By Lemma 11, the at most n cube-level paths generated by Cube-S2S

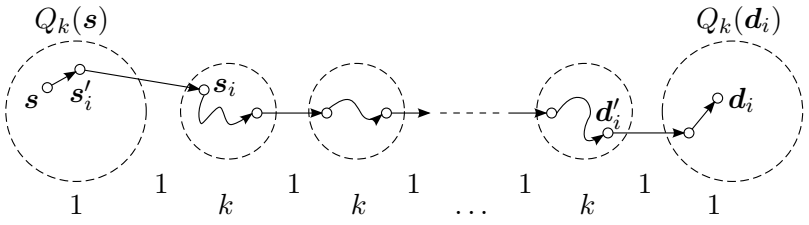

Figure 6.3: A path of maximum length.

in Step 2 are disjoint, have lengths at most $m2^k + n \leq m2^k + k + m$, and require $O(nm2^k \log n)$ time complexity. □

Lemma 18 *Step 3 generates disjoint paths of lengths at most* $(m2^k + k + m)(k + 1) + k + 4$ *in* $O(nkm2^k)$ *time complexity.*

Proof Step 3 performs inside each k-cube (excepted $Q_k(\boldsymbol{s})$) of each path a linear time $O(k)$ SPR, thus requiring $O(k(m2^k + n))$ $(= O(km2^k)$ since $n \leq m2^k)$ time complexity for each path, that is in total for all paths $O(nkm2^k)$ time complexity. Step 3 performs routings inside each k-cube included in a cube-level path generated in Step 2. Concretely, each k-cube routing is achieved by using an SPR, hence requiring at most k edges inside each k-cube, plus one supplementary edge to link the next k-cube in the path. Because paths contain one node more than their number of edges, we need to perform one additional local routing. See Figure 6.3 for an illustration of a path of maximum length. From this discussion, the maximum path length can be expressed as follows.

$$2 + 2 + (m2^k + n)(k + 1) + k$$

$$\leq (m2^k + k + m)(k + 1) + k + 4$$

□

Theorem 3 *In an* $MC(k, m)$*, given a node* $\boldsymbol{s}$ *and a set of* n $(n \leq k + m)$ *nodes* $D = \{\boldsymbol{d}_1, \boldsymbol{d}_2, \ldots, \boldsymbol{d}_n\}$*, we can find* n *disjoint paths* $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}_j$ $(1 \leq j \leq n)$ *of length at most* $(m2^k + n)(k + 1) + k + 4$ *in* $O(nm2^k(\log n + k))$ *time complexity.*

Proof It can be deduced from Lemmas 13, 14, 15, 16, 17 and 18. □

As an example, we use MC-N2S to solve a node-to-set disjoint-path routing problem inside an $MC(2, 2)$, with a source node $\boldsymbol{s} =$ 00 00 00 00 00 and a set of destination nodes $D = \{\boldsymbol{d}_1 =$ 01 01 01 01 01, $\boldsymbol{d}_2 =$ 11 01 01 01 01, $\boldsymbol{d}_3 =$ 11 00 00 00 00, $\boldsymbol{d}_4 =$ 10 11 10 11 00$\}$. This example illustrates a non-trivial

Table 6.1: Routing example inside an $MC(2,2)$.

cube-level	metacube-level	cube-level	metacube-level
	00 00 00 00 00 $\boldsymbol{s}$		00 00 00 00 00 $\boldsymbol{s}$
00 00 00 10	00 00 00 00 10	00 00 00 01	00 00 00 00 01
00 00 00 11	00 00 00 00 11		10 00 00 00 01
	01 00 00 00 11	00 01 00 01	10 00 01 00 01
00 00 01 11	01 00 00 01 11		11 00 01 00 01
	00 00 00 01 11	01 01 00 01	11 01 01 00 01
	10 00 00 01 11		01 01 01 00 01 $\boldsymbol{d}'_1$
00 01 01 11	10 00 01 01 11		01 01 01 01 01 $\boldsymbol{d}_1$
	00 00 01 01 11		
00 01 01 01	00 00 01 01 01		
	01 00 01 01 01		
	11 00 01 01 01 $\boldsymbol{d}'_2$		
	11 01 01 01 01 $\boldsymbol{d}_2$		

cube-level	metacube-level	cube-level	metacube-level
	00 00 00 00 00 $\boldsymbol{s}$		00 00 00 00 00 $\boldsymbol{s}$
	01 00 00 00 00		10 00 00 00 00
00 00 01 00	01 00 00 01 00		11 00 00 00 00 $\boldsymbol{d}_3$
00 00 11 00	01 00 00 11 00		
	00 00 00 11 00		
	10 00 00 11 00		
00 10 11 00	10 00 10 11 00		
	11 00 10 11 00		
01 10 11 00	11 01 10 11 00		
11 10 11 00	11 11 10 11 00		
	10 11 10 11 00 $\boldsymbol{d}_4$		

situation: $\boldsymbol{d}_1$ and $\boldsymbol{d}_2$ are located inside the same k-cube, and $\boldsymbol{d}_3$ is located inside the k-cube of $\boldsymbol{s}$ the source node. The four disjoint paths are given in Table 6.1.

6.4 Empirical evaluation

We conduct in this section an empirical evaluation of the MC-N2S algorithm described in this chapter. MC-N2S has been implemented using the functional programming language Scheme under the DrScheme 4.2.5 development environment [34]. This implementation has been used to solve node-to-set disjoint-path routing problems inside metacubes $MC(k,m)$ with $1 \leq k, m \leq 9$. The largest metacube used in this experiment is an $MC(7,7)$, whose node addresses are sequences of $7 + 7 \times 2^7 = 903$ bits. Inside each metacube $MC(k,m)$ (i.e. for each pair (k,m)), we solved 10,000 node-to-set

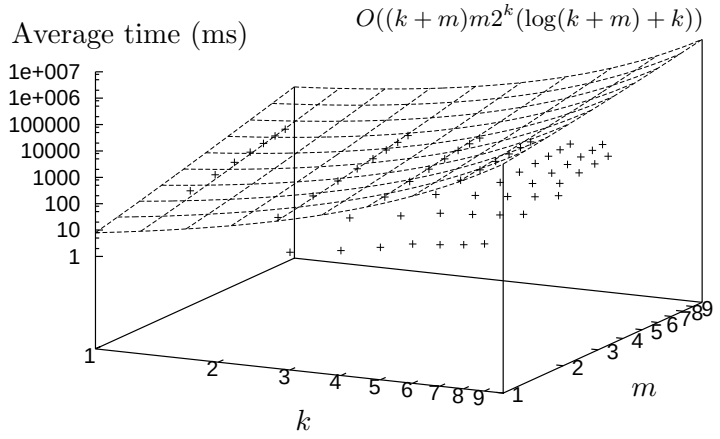

Figure 6.4: Average execution time for each couple (k, m), in milliseconds (logarithmic scale). In this experiment we have $n = k + m$.

disjoint-path routing problems, each time with the number of destination nodes n set to $k + m$.

This experiment is divided into two distinct measurements. First, Figure 6.4 focuses on the average time required to complete one routing problem inside $MC(k, m)$; the theoretical time complexity is also represented. Second, Figure 6.5 and Figure 6.6 respectively represent the average and the maximum maximal path lengths for all the 10,000 routing problems solved inside an $MC(k, m)$. Concretely, for each $MC(k, m)$, we took the average and the maximum of the 10,000 computed maximal path lengths.

We can note that the average time complexity obtained from this experiment is significantly lower than the theoretical worst-case time complexity. Also, the maximum maximal path lengths obtained are also significantly lower than the theoretical maximum path length.

6.5 Summary

We described in this chapter a metacube node-to-set disjoint-path routing algorithm. In an $MC(k, m)$, given one source node and a set of $n \leq k + m$ destination nodes, this algorithm finds n disjoint paths between the source node and the destination nodes of lengths at most $(m2^k + n)(k + 1) + k + 4$ in $O(nm2^k(\log n + k))$ time complexity.

Enhancing the fault tolerance of this algorithm by describing a fault-tolerant node-to-set disjoint-path routing algorithm in a metacube should be considered as future work. Furthermore, cluster fault tolerance is an

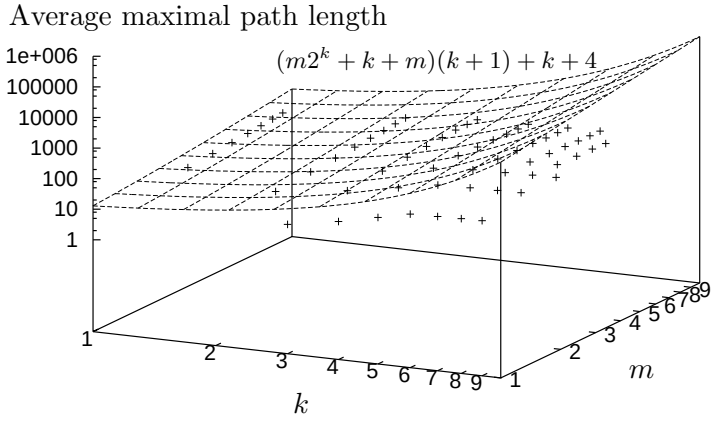

Figure 6.5: Average maximal path lengths for each couple (k, m) (logarithmic scale). In this experiment we have $n = k + m$.

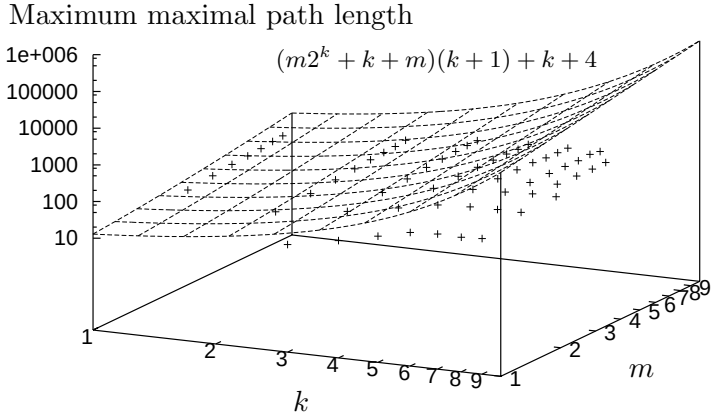

Figure 6.6: Maximum maximal path lengths for each couple (k, m) (logarithmic scale). In this experiment we have $n = k + m$.

interesting topic due to the cluster-based nature of metacubes.

Chapter 7

Set-to-set disjoint-path routing in perfect hierarchical hypercubes

We describe in this chapter a set-to-set disjoint-path routing algorithm HHC-S2S in perfect hierarchical hypercubes. The presentation of HHC-S2S is structured as follows. Section 7.1 introduces or recalls several definitions and lemmas. Then Section 7.2 formally describes HHC-S2S and finally gives its pseudocode. The proof of the correctness of HHC-S2S as well as its complexities are addressed in Section 7.3. Section 7.3 also contains an example of the execution of HHC-S2S. An empirical evaluation of HHC-S2S is performed in Section 7.4. Section 7.5 summarizes this chapter. An appendix solving the set-to-set disjoint-path routing problem in a special case is given in Section 7.6.

7.1 Preliminaries

Given one set of source nodes and one set of destination nodes of same cardinality k, a set-to-set disjoint-path routing algorithm finds k disjoint paths between the k source nodes and the k destination nodes. We recall in Lemma 19 a fault-tolerant set-to-set disjoint-path routing algorithm in hypercubes Cube-S2S described in [46].

Lemma 19 *In a Q_m, given two sets $S = \{s_1, \ldots, s_k\}$, $D = \{d_1, \ldots, d_k\}$ of k ($k \leq m$) nodes and a set F of at most $m - k$ faulty nodes, we can find k fault-free disjoint paths $s_i \rightsquigarrow d_{j_i}$ ($1 \leq i \leq k$) where $\{j_1, j_2, \ldots, j_k\} = \langle k \rangle$ ($= \{1, 2, \ldots, k\}$), of lengths at most $m + k$ in $O(km \log k)$ time complexity.*

Now let us recall a few definitions regarding cluster fault-tolerant routing. First, a cluster is a connected subgraph of a Q_m. A faulty cluster is a cluster

whose nodes are all faulty. A cluster fault-tolerant node-to-node routing algorithm in hypercubes was proposed in [49].

Lemma 20 *Given two non-faulty nodes $s, d \in Q_m$ and at most $m-1$ faulty clusters of diameter at most one with at most $2m-3$ faulty nodes in total, we can find a fault-free path $s \rightsquigarrow d$ of length at most $m+2$ in $O(m)$ time complexity.*

We recall that a path inside an HHC_{2^m+m} is called an HHC-level path, and a path inside a Q_{2^m} is called a cube-level path, made of cube-level nodes which correspond to subcube IDs of an HHC_{2^m+m}.

Finally, we describe in Lemma 21 an extended hypercube set-to-set disjoint-path routing algorithm.

Lemma 21 *In a Q_m, for a set $S = \{s_1, s_2, \ldots, s_{k_1}\}$ of k_1 ($k_1 \leq m$) nodes, a set $D = \{d_1, d_2, \ldots, d_{k_2}\}$ of k_2 ($k_1 < k_2 \leq m+1$ and $k_1 < m+1$) nodes, and k_2 disjoint paths of lengths at most one $d_j \rightarrow d''_j$ ($1 \leq j \leq k_2$), we can find in $O(k_1 {k_2}^2 m)$ time complexity k_1 disjoint paths $s_i \rightsquigarrow d_{j_i}$ ($1 \leq i \leq k_1$) of lengths at most $m + k_1$ that are also disjoint with the paths $d_j \rightarrow d''_j$ ($1 \leq j \leq k_2$ and $j \notin \{j_1, j_2, \ldots, j_{k_1}\}$).*

Proof. We describe an algorithm solving this problem. First, apply Cube-S2S to find k_1 disjoint paths $P_i : s_i \rightsquigarrow d_{j_i}$ ($1 \leq i \leq k_1$) between S and $\{d_1, d_2, \ldots, d_{k_1}\}$ where $\{j_1, j_2, \ldots, j_{k_1}\} = \langle k_1 \rangle$ (see Figure 7.1a). Second, for each path P_i including a node of $\{d_{k_1+1}, d_{k_1+2}, \ldots, d_{k_2}\}$, find d_l ($k_1 + 1 \leq l \leq k_2$) on P_i which is closest to s_i, swap the indices of d_l, d_{j_i} and d''_l, d''_{j_i}, and discard the subpath $d_{j_i} \rightsquigarrow d_l$ (see Figure 7.1b). Third, while there exists a path P_i such that $P_i \cap \{d''_{k_1+1}, \ldots, d''_{k_2}\} \neq \emptyset$, do as follows. Let d''_l be the closest node of $P_i \cap \{d''_{k_1+1}, \ldots, d''_{k_2}\}$ to s_i, select the path $s_i \rightsquigarrow d''_l \rightarrow d_l$, swap the indices of d_l, d_{j_i} and d''_l, d''_{j_i}, and discard the subpath $d''_{j_i} \rightsquigarrow d_l$ (see Figure 7.1c).

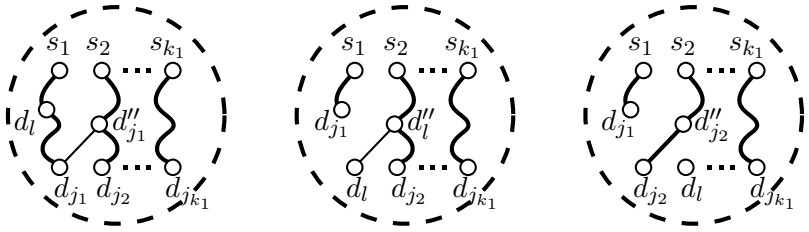

Figure 7.1: The three steps of the algorithm for Lemma 21 (all the paths $d_j \rightarrow d''_j$ ($1 \leq j \leq k_2$) are not represented).

By Lemma 19, Cube-S2S generates disjoint paths of lengths at most $m + k_1$ in $O(k_1 m \log k_1)$ time complexity. The inclusion of a node d_l or d''_l

$(k_1 + 1 \leq l \leq k_2)$ on a path P_i triggers the discarding of a subpath of P_i and the adding of at most one edge. Hence the lengths of the generated paths do not increase. Each path P_i is first traversed to check for inclusion of a node d_l $(k_1 + 1 \leq l \leq k_2)$, thus requiring in total $O(k_1 k_2 m)$ time complexity. Then each path P_i is traversed to check for inclusion of a node d''_l $(k_1 + 1 \leq l \leq k_2)$. If P_i includes such a d''_l, a new traversal of the paths P_i $(1 \leq i \leq k_1)$ is required. Such a node d''_l can be included on at most one path P_i and can thus trigger only once a new traversal of the paths P_i. Hence this algorithm is of $O(k_2(k_1 k_2 m))$ time complexity. □

7.2 Set-to-set disjoint-path routing algorithm

In this section, we describe a set-to-set disjoint-path routing algorithm HHC-S2S in an HHC_{2^m+m}. The main idea of this algorithm is to reduce the set-to-set disjoint-path routing problem in an HHC to the fault-tolerant set-to-set disjoint-path routing problem in a hypercube. In practice, this is achieved by mapping each subcube of an HHC_{2^m+m} to a single node of a Q_{2^m}. Let $S = \{\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_k\}$ be the set of source nodes and $D = \{\boldsymbol{d}_1, \boldsymbol{d}_2, \ldots, \boldsymbol{d}_k\}$ be the set of destination nodes where $k \leq m + 1$.

Case I: $\exists Q_m(\sigma), S \cup D \subset Q_m(\sigma)$
Case I-a: $k \leq m$
Apply Cube-S2S inside $Q_m(\sigma)$ to disjointly connect the nodes of S to the nodes of D.
Case I-b: $k = m + 1$
Apply Cube-S2S inside $Q_m(\sigma)$ to disjointly connect $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_{k-1}$ to $\boldsymbol{d}_1, \ldots, \boldsymbol{d}_{k-1}$. If $\boldsymbol{s}_k$ is included on one of the generated paths, say $\boldsymbol{s}_i \rightsquigarrow \boldsymbol{d}_{j_i}$, swap the indices of $\boldsymbol{s}_i$ and $\boldsymbol{s}_k$. Similarly if $\boldsymbol{d}_k$ is included on one of the generated paths, say $\boldsymbol{s}_i \rightsquigarrow \boldsymbol{d}_{j_i}$, swap the indices of $\boldsymbol{d}_{j_i}$ and $\boldsymbol{d}_k$. Hence we can assume $\boldsymbol{s}_k = (\sigma, \pi_{s_k})$ and $\boldsymbol{d}_k = (\sigma, \pi_{d_k})$ are not included on one of the generated paths inside $Q_m(\sigma)$. Connect $\boldsymbol{s}_k$ to $\boldsymbol{d}_k$ with the path $\boldsymbol{s}_k \rightarrow (\sigma \oplus 2^{\pi_{s_k}}, \pi_{s_k}) \overset{\text{SPR}}{\rightsquigarrow} (\sigma \oplus 2^{\pi_{s_k}}, \pi_{d_k}) \rightarrow (\sigma \oplus 2^{\pi_{s_k}} \oplus 2^{\pi_{d_k}}, \pi_{d_k}) \overset{\text{SPR}}{\rightsquigarrow} (\sigma \oplus 2^{\pi_{s_k}} \oplus 2^{\pi_{d_k}}, \pi_{s_k}) \rightarrow (\sigma \oplus 2^{\pi_{d_k}}, \pi_{s_k}) \overset{\text{SPR}}{\rightsquigarrow} (\sigma \oplus 2^{\pi_{d_k}}, \pi_{d_k}) \rightarrow \boldsymbol{d}_k$. See Figure 7.2.

Case II: $\forall Q_m(\sigma), S \cup D \not\subset Q_m(\sigma)$
Define $S(\sigma) = Q_m(\sigma) \cap S$ and $D(\sigma) = Q_m(\sigma) \cap D$. Assume the sets F, Z_s, Z_d and C are all initially empty. In Step 1, F tracks already processed subcubes, and in Step 3 then plays the role of a set of faulty nodes. Z_d (resp. Z_s) contains subcubes including at least two source (resp. destination) nodes and no destination (resp. source) node, where each source (resp. destination) node has already been connected to a destination (resp. source) node. C contains nodes part of already established paths.

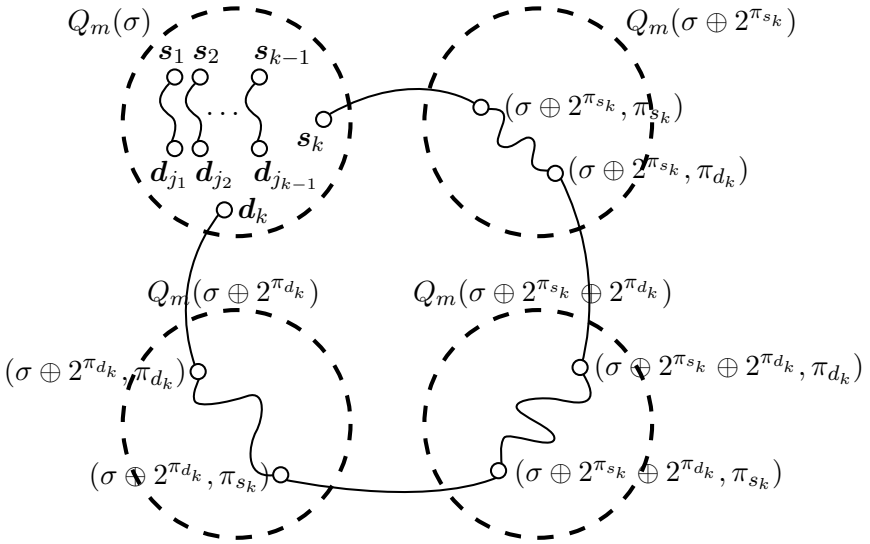

Figure 7.2: Case I-b ($\{j_1, j_2, \ldots, j_{k-1}\} = \langle k-1 \rangle$).

Step 1. For each subcube $Q_m(\sigma)$, $\sigma \notin F$ with $|S(\sigma)| + |D(\sigma)| \geq 2$, proceed in six steps as follows. Assume $|S(\sigma)| \leq |D(\sigma)|$. We can assume without loss of generality that $S(\sigma) = \{\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_{|S(\sigma)|}\}$.

The case $|S(\sigma)| > |D(\sigma)|$ is handled similarly by exchanging the roles of the source and destination nodes where Z_d is used instead of Z_s, and τ instead of ρ.

1. Add σ in F.

2. For all $\boldsymbol{d}_j \in D(\sigma) \setminus C$, find by enumeration of the $m+1$ paths

$$\begin{cases} \boldsymbol{d}_j = (\sigma, \pi) \rightarrow (\sigma \oplus 2^{\pi}, \pi) \\ \boldsymbol{d}_j = (\sigma, \pi) \rightarrow (\sigma, \pi \oplus 2^h) \rightarrow (\sigma \oplus 2^{\pi \oplus 2^h}, \pi \oplus 2^h) \quad (0 \leq h \leq m-1) \end{cases}$$

   mutually disjoint paths $\rho_j : \boldsymbol{d}_j \rightsquigarrow \boldsymbol{d}'_j \in Q_m(\sigma'_j)$ of lengths at most two such that all σ'_j are distinct, $D(\sigma'_j) = \emptyset$, $\sigma'_j \notin Z_d$ and ρ_j disjoint with any path already constructed between one source and one destination node (initially none). The paths ρ_j are of the form $\boldsymbol{d}_j \rightarrow \boldsymbol{d}''_j \rightarrow \boldsymbol{d}'_j$ or $\boldsymbol{d}_j (= \boldsymbol{d}''_j) \rightarrow \boldsymbol{d}'_j$.

3. Apply Lemma 21 to $S(\sigma)$ and $D(\sigma)$ in $Q_m(\sigma)$ to find disjoint paths $\boldsymbol{s}_i \rightsquigarrow \boldsymbol{d}_{j_i}$ $(1 \leq i \leq |S(\sigma)|)$ also disjoint with the paths ρ_j $(j \in J_1)$ where $J_1 = \{j \mid \boldsymbol{d}_j \in D(\sigma) \setminus \{j_1, j_2, \ldots, j_{|S(\sigma)|}\}\}$. See Figure 7.3.

4. For each path $\boldsymbol{s}_i \rightsquigarrow \boldsymbol{d}_{j_i}$ $(1 \leq i \leq |S(\sigma)|)$, update ρ_{j_i} to the path of length zero $\boldsymbol{d}_{j_i}$ and alias $\boldsymbol{d}_{j_i}$ with $\boldsymbol{d}'_{j_i}$.

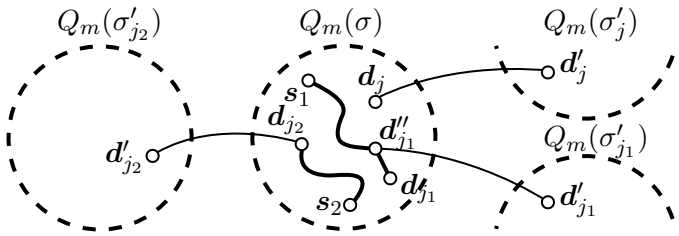

Figure 7.3: Case II, Step 1 - Set-to-set disjoint-path routing by Lemma 21.

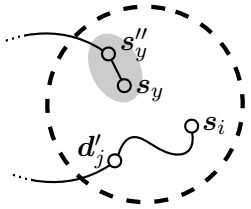

Figure 7.4: Case II, Step 1 - Connection of $\boldsymbol{d}'_j$ (elements of C are grayed).

5. For each $\boldsymbol{d}'_j$ ($j \in J_1$), say $\boldsymbol{d}'_j \in Q_m(\sigma'_j)$, with $|S(\sigma'_j)| \geq 2$, distinguish three cases as follows.

   $\sigma'_j \notin F$**:** Find $\boldsymbol{s}_i \in S(\sigma'_j)$ closest to $\boldsymbol{d}'_j$ and select the path $\boldsymbol{d}'_j \overset{\text{SPR}}{\rightsquigarrow} \boldsymbol{s}_i$. Add $\boldsymbol{d}_j \to \boldsymbol{d}''_j$ and $\boldsymbol{s}_i$ into C.

   $\sigma'_j \in F$ **and** $\boldsymbol{d}'_j \notin C$**:** Apply Lemma 20 to obtain a path between $\boldsymbol{d}'_j$ and an arbitrary $\boldsymbol{s}_i \in S(\sigma'_j) \setminus C$ avoiding nodes in C (see Figure 7.4). If $\exists \tau_l : \boldsymbol{s}_l \rightsquigarrow \boldsymbol{s}'_l$ with $\boldsymbol{s}_l \in S(\sigma'_j) \setminus (C \cup \{\boldsymbol{s}_i\})$ that is not disjoint with $\boldsymbol{d}'_j \rightsquigarrow \boldsymbol{s}_i$, find $\tilde{\boldsymbol{s}} \in \tau_l \cap Q_m(\sigma'_j)$ closest to $\boldsymbol{d}'_j$, discard the subpath $\tilde{\boldsymbol{s}} \rightsquigarrow \boldsymbol{s}_i$, update τ_l to the path of length zero $\boldsymbol{s}_l$, alias $\boldsymbol{s}_l$ with $\boldsymbol{s}'_l$, and if $\tilde{\boldsymbol{s}} \neq \boldsymbol{s}_l$, connect $\boldsymbol{d}'_j \rightsquigarrow \tilde{\boldsymbol{s}}$ to $\tilde{\boldsymbol{s}} \to \boldsymbol{s}_l$. Otherwise, that is $\forall \tau_l : \boldsymbol{s}_l \rightsquigarrow \boldsymbol{s}'_l$ with $\boldsymbol{s}_l \in S(\sigma'_j) \setminus (C \cup \{\boldsymbol{s}_i\})$ the path $\boldsymbol{d}'_j \rightsquigarrow \boldsymbol{s}_i$ is disjoint with τ_l, update τ_i to the path of length zero $\boldsymbol{s}_i$ and alias $\boldsymbol{s}_i$ with $\boldsymbol{s}'_i$. See Figure 7.5. Add $\boldsymbol{d}_j \to \boldsymbol{d}''_j$ and $\boldsymbol{s}_i$ into C.

   $\sigma'_j \in F$ **and** $\boldsymbol{d}'_j \in C$ **(i.e.** $\boldsymbol{d}'_j \in S(\sigma'_j)$**):** Assume $\boldsymbol{d}'_j = \boldsymbol{s}_i$, replace $\boldsymbol{s}_i \to \boldsymbol{s}''_i$ by $\boldsymbol{s}_i$ in C, discard the edge $\boldsymbol{s}_i \to \boldsymbol{s}''_i$, and then proceed similarly as in the previous case to connect $\boldsymbol{s}''_i$ to an arbitrary node $\boldsymbol{s}_y \in S(\sigma'_j) \setminus C$. See Figure 7.6. Add $\boldsymbol{d}_j \to \boldsymbol{d}''_j$ and $\boldsymbol{s}_i$ into C.

6. If $D(\sigma) \subset C$, add σ into Z_s.

Let $S' = \{\boldsymbol{s}'_i \mid \tau_i : \boldsymbol{s}_i \in Q_m(\sigma) \rightsquigarrow \boldsymbol{s}'_i \in Q_m(\sigma'), \sigma \in F, \sigma' \notin F\}$ and $D' = \{\boldsymbol{d}'_j \mid \rho_j : \boldsymbol{d}_j \in Q_m(\sigma) \rightsquigarrow \boldsymbol{d}'_j \in Q_m(\sigma'), \sigma \in F, \sigma' \notin F\}$. Define $S'(\sigma) = Q_m(\sigma) \cap S'$ and $D'(\sigma) = Q_m(\sigma) \cap D'$.

Step 2. For each of all $\boldsymbol{s}'_i \in S'$, say $\boldsymbol{s}'_i \in Q_m(\sigma)$, with $|D(\sigma)| + |D'(\sigma)| = 1$, say $D(\sigma) \cup D'(\sigma) = \{\boldsymbol{d}'_j\}$, add σ into F and connect $\boldsymbol{s}'_i$ to $\boldsymbol{d}'_j$ with an SPR.

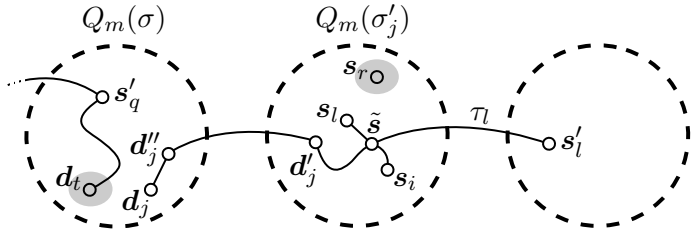

Figure 7.5: Case II, Step 1 - Connection of $\boldsymbol{d}'_j$ with $\boldsymbol{d}'_j \notin C$ (elements of C are grayed).

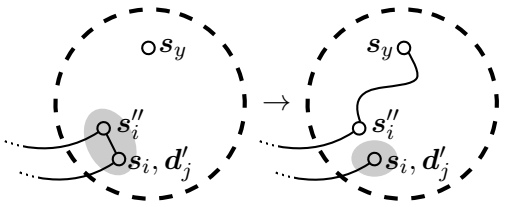

Figure 7.6: Case II, Step 1 - Connection of $\boldsymbol{s}''_i$ (elements of C are grayed).

For each of all $\boldsymbol{d}'_j \in D'$, say $\boldsymbol{d}'_j \in Q_m(\sigma)$, with $|S(\sigma)| = 1$, say $S(\sigma) = \{\boldsymbol{s}_i\}$, add σ into F and connect $\boldsymbol{s}_i$ to $\boldsymbol{d}'_j$ with an SPR.

Step 3. Apply Lemma 19 to find a set $\mathcal{P}$ of cube-level disjoint paths connecting the cube-level nodes $\tilde{S} = \{\sigma \mid (\sigma, \pi) \in S \cup S'\} \setminus F$ to the cube-level nodes $\tilde{D} = \{\sigma \mid (\sigma, \pi) \in D \cup D'\} \setminus F$ without including any node of F.

Step 4. For each path $P : \sigma_0 \rightsquigarrow \sigma_n$ of $\mathcal{P}$, assume $(S \cup S') \cap Q_m(\sigma_0) = \{\boldsymbol{s}'_i = (\sigma_0, \pi_{s'_i})\}$ and $(D \cup D') \cap Q_m(\sigma_n) = \{\boldsymbol{d}'_j = (\sigma_n, \pi_{d'_j})\}$. The HHC-level path corresponding to P is given by $\text{CONV}(\sigma_{s_i} \rightarrow P \rightarrow \sigma_{d_j}, \pi_{s_i}, \pi_{d_j})$ where $\boldsymbol{s}_i = (\sigma_{s_i}, \pi_{s_i})$ and $\boldsymbol{d}_j = (\sigma_{d_j}, \pi_{d_j})$.

7.3 Correctness and complexities

We show in this section the correctness and time complexity of HHC-S2S, and we establish an upper bound for the maximum path length.

Lemma 22 *Case I generates disjoint paths of lengths at most* $3m + 4$ *in* $O(km \log k)$ *time complexity.*

Proof. By Lemma 19, Cube-S2S generates k (Case I-a) or $k - 1$ (Case I-b) disjoint paths inside $Q_m(\sigma)$ of lengths at most $m + k$ in $O(km \log k)$ time complexity. Case I-b generates an additional path going outside of $Q_m(\sigma)$ of length at most $3m + 4$ since it consists of four external edges and three subcube shortest-path routings. That path can be constructed in

$O(m)$ time complexity. All its internal nodes are outside $Q_m(\sigma)$, hence this additional path is disjoint from the other paths. In Case I-b, checking if $\boldsymbol{s}_k$ or $\boldsymbol{d}_k$ are included on a path generated inside $Q_m(\sigma)$ requires $O(km)$ time complexity since there are $k-1$ paths of lengths at most $2m$. □

Lemma 23 *Step 1.2 of Case II generates disjoint paths ρ (resp. τ) of lengths at most two in $O(m^2)$ time complexity.*

PROOF. Subcubes $Q_m(\sigma)$ with $|S(\sigma)|+|D(\sigma)| \geq 2$ can be found in $O(m^2 2^m)$ time complexity. Assume $|S(\sigma)| \leq |D(\sigma)|$. A similar discussion holds for the case $|S(\sigma)| > |D(\sigma)|$. We show that we can find a path ρ_i for a node $\boldsymbol{d}_i \in D(\sigma) \setminus C$. Assume a path $\boldsymbol{s}_q \rightsquigarrow \boldsymbol{d}_t$ has already been established and that its subpath $\boldsymbol{s}'_q \rightsquigarrow \boldsymbol{d}_t$ is inside $Q_m(\sigma)$ (i.e. $\boldsymbol{d}_t \in D(\sigma) \cap C$). For parity reasons, a path ρ_j for a $\boldsymbol{d}_j$ $(i \neq j)$ can block at most one of the $m+1$ candidate paths for ρ_i. Since $H(\boldsymbol{s}'_q, \boldsymbol{d}_t) \leq H(\boldsymbol{s}'_q, \boldsymbol{d}_i)$, then $\boldsymbol{d}_t \in N(\boldsymbol{d}_j)$ and $N(\boldsymbol{d}_j) \cap N(\boldsymbol{d}_t) \neq \emptyset$ cannot hold simultaneously for parity reason. Hence the path $\boldsymbol{s}'_q \rightsquigarrow \boldsymbol{d}_t$ can block at most one candidate for ρ_i.

Assume $|D(\sigma) \setminus \{\boldsymbol{d}_i\}| = \alpha$ and $|\bigcup_{H(\sigma,\sigma'_j)=1} D(\sigma'_j)| = \beta$ where for any $\rho_j : \boldsymbol{d}_j \rightsquigarrow \boldsymbol{d}'_j$ $(i \neq j)$, $\boldsymbol{d}'_j \in Q_m(\sigma'_j)$. The nodes of $D(\sigma) \setminus \{\boldsymbol{d}_i\}$ block at most α candidates for ρ_i, and the nodes of $\bigcup_{H(\sigma,\sigma'_j)=1} D(\sigma'_j)$ block at most β candidates for ρ_i. By the remaining $m-\alpha-\beta$ destination nodes, at most $\lfloor (m-\alpha-\beta)/4 \rfloor$ subcubes are in Z_d since $\sigma' \in Z_d$ implies $S(\sigma') \subset C$. See Figure 7.7. If $S(\sigma') \subset C$, then for each of all $\boldsymbol{s}_l \in S(\sigma')$, we have, according to $\tau_l : \boldsymbol{s}_l \rightsquigarrow \boldsymbol{s}'_l$, that $\boldsymbol{s}'_l \in Q_m(\sigma'_l)$ with $|D(\sigma'_l)| \geq 2$. Because $H(\sigma, \sigma') = 1$ and $H(\sigma, \sigma'_l) = 1$ cannot hold simultaneously for parity reason, there is no candidate path for ρ_i connecting $\boldsymbol{d}_i$ to a node in any of the subcubes $Q_m(\sigma'_l)$. Hence, a subcube $Q_m(\sigma')$ with $\sigma' \in Z_d$ can block at most one candidate path for ρ_i but at the same time it implies that at least four destination nodes cannot block a candidate path for ρ_i. Therefore at least one candidate for ρ_i remains unblocked.

As for the time complexity for finding a path ρ_i, the candidate paths blocked are detected in $O(\alpha+\beta+m) = O(m)$ time complexity. Hence finding a path ρ for each of all nodes of $D(\sigma) \setminus C$ requires $O(m^2)$ total time complexity. □

Lemma 24 *Step 1 of Case II generates disjoint paths of lengths at most $2m+5$ in $O(m^2 2^m)$ time complexity.*

PROOF. By Lemma 21 Step 1.3 generates disjoint paths of lengths at most $2m$ in $O(m^5)$ time complexity.

Finally, a node $\boldsymbol{d}'_j \in Q_m(\sigma')$ may be connected to a node $\boldsymbol{s}_i \in Q_m(\sigma') \setminus C$ according to Lemma 20. Let us show that we can apply Lemma 20 inside $Q_m(\sigma')$, that is showing that $Q_m(\sigma')$ contains at most $m-1$ faulty clusters and at most $2m-3$ faulty nodes in total. A faulty cluster of diameter one $\boldsymbol{s}_i \rightarrow \boldsymbol{s}''_i$ or of diameter zero $\boldsymbol{s}_i$ is created only when $|D(\sigma'_i)| \geq 2$ holds, where

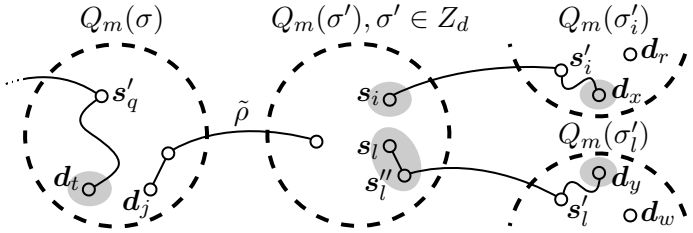

Figure 7.7: A candidate path $\tilde{\rho}$ for ρ_j blocked by a subcube $Q_m(\sigma')$ with $\sigma' \in Z_d$ (elements of C are grayed).

$\boldsymbol{s}'_i \in Q_m(\sigma'_i)$. Hence $Q_m(\sigma')$ can contain at most $\lfloor (m+1)/2 \rfloor$ faulty clusters of diameter at most one. Then for $m \geq 4$ we can apply Lemma 20 inside $Q_m(\sigma')$ since $Q_m(\sigma')$ can contain at most $\lfloor (m+1)/2 \rfloor \leq m-2$ faulty clusters of diameter at most one, that is at most $2m-4$ faulty nodes in total. Now if $m = 3$, $Q_m(\sigma')$ can contain at most two faulty clusters of diameter at most one. Then if $Q_m(\sigma')$ contains two faulty clusters of diameter exactly one, we cannot apply Lemma 20 inside $Q_m(\sigma')$. However, for $Q_m(\sigma')$ to contain two faulty clusters of diameter one $\boldsymbol{s}_i \rightarrow \boldsymbol{s}''_i$ and $\boldsymbol{s}_j \rightarrow \boldsymbol{s}''_j$, it means that $|D(\sigma'_i)| = |D(\sigma'_j)| = 2$ holds, where $\boldsymbol{s}'_i \in Q_m(\sigma'_i)$ and $\boldsymbol{s}'_j \in Q_m(\sigma'_j)$. Because the only external edge between $Q_m(\sigma')$ and $Q_m(\sigma'_i)$ is the edge $\boldsymbol{s}''_i \rightarrow \boldsymbol{s}'_i$ which is included in the path $\tau_i : \boldsymbol{s}_i \rightsquigarrow \boldsymbol{s}'_i$, the node $\boldsymbol{d}'_q$ according to the path $\rho_q : \boldsymbol{d}_q \rightsquigarrow \boldsymbol{d}'_q$ for the single node $\boldsymbol{d}_q \in D(\sigma'_i) \setminus C$ cannot be inside $Q_m(\sigma')$ as we impose ρ_q to be disjoint with the path $\boldsymbol{s}'_i \rightsquigarrow \boldsymbol{d}_l$. A similar discussion also holds for the nodes of $D(\sigma'_j)$. Hence we will never apply Lemma 20 inside $Q_m(\sigma')$, and HHC-S2S remains applicable for $m = 3$. See Figure 7.8. The same discussion holds for the case $m = 2$.

In a Q_m, Lemma 20 generates a path $\boldsymbol{s}_i \rightsquigarrow \boldsymbol{d}'_j$ of length at most $m+2$ in $O(m)$ time complexity. Taking into consideration the path $\rho_j : \boldsymbol{d}_j \rightsquigarrow \boldsymbol{d}'_j$ of length at most two, we obtain a path of length at most $m+4$. Now if there is a $\boldsymbol{d}'_j \in C$, say $\boldsymbol{d}'_j = \boldsymbol{s}_i$, $\boldsymbol{s}''_i$ is connected to a node $\boldsymbol{s}_y$ with Lemma 20, thus obtaining a path $\boldsymbol{d}_l \rightsquigarrow \boldsymbol{s}'_i \rightarrow \boldsymbol{s}''_i \rightsquigarrow \boldsymbol{s}_y$ of length at most $(m+2)+1+(m+2) = 2m+5$ since we apply twice Lemma 20. Checking if the path $\boldsymbol{s}_i \rightsquigarrow \boldsymbol{d}'_j$ (or $\boldsymbol{s}''_i \rightsquigarrow \boldsymbol{s}_y$) includes a node $\tilde{\boldsymbol{s}}$ requires $O(m^2)$ time complexity. Hence the total time complexity of Step 1 is $O(m^2)$. □

Lemma 25 *Step 2 of Case II generates disjoint paths of lengths at most $m+4$ in $O(km)$ time complexity.*

Proof. Subcubes $Q_m(\sigma)$, $\sigma \notin F$ with $S' \cap Q_m(\sigma) \neq \emptyset$ and $|(D \cup D') \cap Q_m(\sigma)| = 1$, or $D' \cap Q_m(\sigma) \neq \emptyset$ and $|S \cap Q_m(\sigma)| = 1$, can be found in $O(k^2)$ time complexity. A shortest-path routing in a subcube generates a path of length at most m in $O(m)$ time complexity. Hence, considering the paths ρ and τ, Step 2 generates paths of lengths at most $m+4$ in $O(km)$

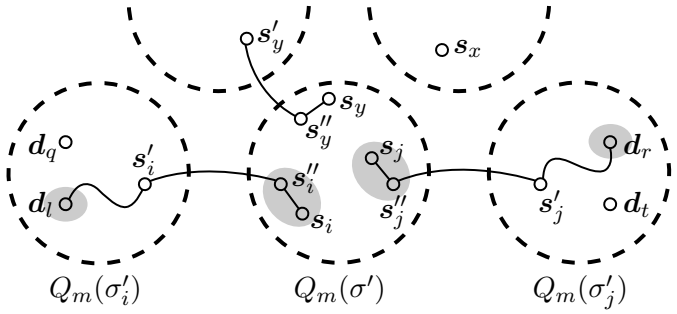

Figure 7.8: A subcube with two faulty clusters of diameter one for $m = 3, k = 4$ (elements of C are grayed).

time complexity. By Lemma 3 and the fact that either $S' \cap Q_m(\sigma) \neq \emptyset$ and $|(D \cup D') \cap Q_m(\sigma)| = 1$, or $D' \cap Q_m(\sigma) \neq \emptyset$ and $|S \cap Q_m(\sigma)| = 1$ holds, such paths are necessarily disjoint with other constructed paths. □

Lemma 26 *Step 3 of Case II generates cube-level disjoint paths of lengths at most* $2^m + k$ *in* $O(k2^m \log k)$ *time complexity.*

Proof. To apply Cube-S2S in a Q_{2^m}, we need that the number of destination nodes (equal to the number of source nodes) plus the number of faulty nodes be at most 2^m. Let us give an upper bound for the sum $|\tilde{D}|+|F|$ where $\tilde{D}$ is the set of cube-level destination nodes and F is the set of faulty nodes when calling Cube-S2S.

From the description of HHC-S2S in Section 7.2, we distinguish three types of faulty nodes: F_0 is the set of subcube IDs whose corresponding subcubes contain one source node and one destination node. F_1 is the set of subcube IDs whose corresponding subcubes contain at least two source nodes or at least two destination nodes. F_2 is the set of subcube IDs whose corresponding subcubes contain one node of S' and one node of D', or one node of S' and one destination node, or one source node and one node of D'. Hence we have $|\tilde{D}|+|F| \leq (k-|F_0|-|F_2|)+(|F_0|+|F_1|+|F_2|) = k+|F_1| \leq k + 2\lfloor k/2 \rfloor \leq (m+1) + 2\lfloor (m+1)/2 \rfloor$. Therefore $|\tilde{D}| + |F| \leq 2^m$ holds for $m \geq 3$. If $m = 1$, the corresponding HHC is a cycle and it is thus trivial to disjointly connect at most $m+1 = 2$ source and destination nodes. If $m = 2$ and $k \leq 2$ then $|\tilde{D}| + |F| \leq 2^m$ still holds. If $m = 2$ and $k = m + 1 = 3$ then we solve the problem case by case depending on the repartition of the source and destination nodes. See Section 7.6.

Hence by Lemma 19, Cube-S2S applied in a Q_{2^m} generates disjoint paths of lengths at most $2^m + k$ in $O(k2^m \log k)$ time complexity. □

Lemma 27 *Step 4 of Case II generates HHC-level disjoint paths of lengths at most* $(m+1)(2^m + k + 1) + 3$ *in* $O(km2^m)$ *time complexity.*

PROOF. By Lemma 26, the cube-level paths generated by Cube-S2S are disjoint and have lengths at most $2^m + k$. Hence the HHC-level paths generated in Step 4 by using CONV are disjoint. Also, a cube-level path of length at most $2^m + k$ corresponds to at most $2^m + k$ external edges and at most $2^m + k + 1$ cube-level nodes. CONV applies an SPR inside each subcube (i.e. cube-level node) included in a path. Therefore, considering at most two external edges and at most two internal edges for source and destination node paths ρ, τ, we obtain the following maximum path length.

$$\underbrace{2^m + k + 2}_{\text{external edges}} + \underbrace{m(2^m + k + 1) + 2}_{\text{internal edges}}$$

$$= (m + 1)(2^m + k + 1) + 3$$

Applying CONV to one cube-level path of length $O(2^m)$ requires $O(m2^m)$ time complexity. Hence converting $O(k)$ such paths takes $O(km2^m)$ time complexity, which is the dominant time complexity of HHC-S2S. □

Theorem 4 *In an HHC_{2^m+m}, given two sets of k ($k \leq m + 1$) nodes $S = \{\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_k\}$ and $D = \{\boldsymbol{d}_1, \boldsymbol{d}_2, \ldots, \boldsymbol{d}_k\}$, we can find k disjoint paths $\boldsymbol{s}_i \rightsquigarrow \boldsymbol{d}_{j_i}$ ($1 \leq i \leq k$) where $\{j_1, j_2, \ldots, j_k\} = \langle k \rangle$, of lengths at most $(m + 1)(2^m + k + 1) + 3$ in $O(km2^m)$ time complexity.*

PROOF. It can be deduced from Lemmas 22, 24, 25, 26 and 27. □

One should note this theoretical maximum path length is not attainable. Let us consider a cube-level path $\sigma_0 \rightarrow \sigma_1 \rightarrow \ldots \rightarrow \sigma_\lambda$, and let $b_i = \log_2(\sigma_{i-1} \oplus \sigma_i)$ $(1 \leq i \leq \lambda)$. Assume each shortest-path routing inside $Q_m(\sigma_i)$ requires m internal edges, then $H(b_i, b_{i+1}) = m$. Hence $b_1 = b_3 = b_5 = \ldots$ and $b_2 = b_4 = b_6 = \ldots$ hold. It means $\sigma_0 = \sigma_4$ which indicates the presence of a cycle inside that path, which is a contradiction. Therefore at least one shortest-path routing inside a subcube requires fewer than m edges.

As an example, we find in an HHC_{11} ($m = 3$) with HHC-S2S four disjoint paths between the two sets $S = \{\boldsymbol{s}_1 = (00000000, 010), \boldsymbol{s}_2 = (00000000, 000), \boldsymbol{s}_3 = (00000011, 101), \boldsymbol{s}_4 = (00000011, 010)\}$ and $D = \{\boldsymbol{d}_1 = (00000001, 001), \boldsymbol{d}_2 = (00000001, 011), \boldsymbol{d}_3 = (11000000, 111), \boldsymbol{d}_4 = (00001111, 001)\}$. The paths generated are given in Table 7.1.

7.4 Empirical evaluation

In this section we empirically measure the algorithm HHC-S2S described in Section 7.2 to inspect its practical behaviour. This algorithm has been implemented and tested under the DrScheme 4.2.5 development environment [34]. We used this implementation of HHC-S2S to solve set-to-set disjoint-path

Table 7.1: Routing example in an HHC_{11}.

Cube-level path	HHC-level path	Cube-level path	HHC-level path
00000000	(00000000, 010) $\boldsymbol{s}_1$	00000000	(00000000, 000) $\boldsymbol{s}_2$
00000100	(00000100, 010) $\boldsymbol{s}'_1$	00000001	(00000001, 000) $\boldsymbol{s}'_2$
	(00000100, 011)		(00000001, 001) $\boldsymbol{d}_1$
	(00000100, 111)		
10000100	(10000100, 111)		
	(10000100, 110)		
11000100	(11000100, 110)		
	(11000100, 010)		
11000000	(11000000, 010)		
	(11000000, 011)		
	(11000000, 111) $\boldsymbol{d}_3$		

Cube-level path	HHC-level path	Cube-level path	HHC-level path
00000011	(00000011, 101) $\boldsymbol{s}_3$	00000011	(00000011, 010) $\boldsymbol{s}_4$
00100011	(00100011, 101) $\boldsymbol{s}'_3$	00000111	(00000111, 010) $\boldsymbol{s}'_4$
	(00100011, 001)		(00000111, 011)
	(00100011, 011)	00001111	(00001111, 011)
00101011	(00101011, 011)		(00001111, 001) $\boldsymbol{d}_4$
	(00101011, 001)		
	(00101011, 101)		
00001011	(00001011, 101)		
	(00001011, 001)		
00001001	(00001001, 001)		
	(00001001, 011) $\boldsymbol{d}'_2$		
00000001	(00000001, 011) $\boldsymbol{d}_2$		

routing problems inside perfect hierarchical hypercubes HHC_{2^m+m} where $3 \leq m \leq 9$.

First we measured the average execution time of HHC-S2S with m ranging from 3 to 9 as explained previously. Second we measured the average and maximum of the obtained maximal path lengths. Each maximal path length is retrieved when solving one routing problem and analyzing the paths generated.

In practice, we solved 10,000 set-to-set disjoint-path routing problems for each value of m. That is, we used nodes with addresses taking up to $2^9 = 512$ bits. The set of source nodes and the set of destination node are both randomly (uniform) generated so that they always contain the maximum $m + 1$ nodes.

Figure 7.9 represents for each value of m the average time in milliseconds used to solve one set-to-set disjoint-path routing problem. The theoretical worst-case time complexity is also represented for comparison. Figure 7.10 represents the average and maximum maximal path length for each value of m. Similarly, the theoretical maximum path length of HHC-S2S is also represented for comparison. We can note that as the size of the graph increases, the probability to generate a path of maximum length becomes lower, which explains the divergence between the empirical results and the theoretical maximum path length.

7.5 Summary

We described in this paper an HHC set-to-set disjoint-path routing algorithm HHC-S2S. In an HHC_{2^m+m}, given two sets S and D of k ($k \leq m+1$) nodes, HHC-S2S finds k disjoint paths between the nodes of S and the nodes of D of lengths at most $(m + 1)(2^m + k + 1) + 3$ in $O(km2^m)$ time complexity.

7.6 Appendix - Case $m = 2$ and $k = 3$

In the case $m = 2$ and $k = m + 1 = 3$, we solve the set-to-set disjoint-path routing problem by distinguishing several cases based on the repartition of the source and destination nodes. All cases not mentioned can be reduced to the ones described by exchanging the roles of source and destination nodes. We recall $\tilde{D}$ and F are the set of the destination nodes and the set of faulty nodes, respectively, when applying Lemma 19 in Q_{2^m}. Also note that Lemma 19 is applicable in Q_{2^m} if $|\tilde{D}| + |F| \leq 2^m$.

If $\exists \sigma, \sigma'$ with $|S(\sigma)| = 1, |D(\sigma)| = 3$ and $|S(\sigma')| = 2$, apply Case II ($|\tilde{D}| = 2$ and $F = \{\sigma, \sigma'\}$, then $|\tilde{D}| + |F| \leq 2^m$). For each σ with $|S(\sigma)| = |D(\sigma)| \leq 2$, apply Lemma 19 inside $Q_m(\sigma)$ and add σ into F. Since in this case $|\tilde{D}|$ decreases by as many as $|F|$ increases, $|\tilde{D}| + |F| \leq 2^m$ holds. Then we can assume without loss of generality that $\forall \sigma, S(\sigma) = \emptyset$ or $D(\sigma) = \emptyset$.

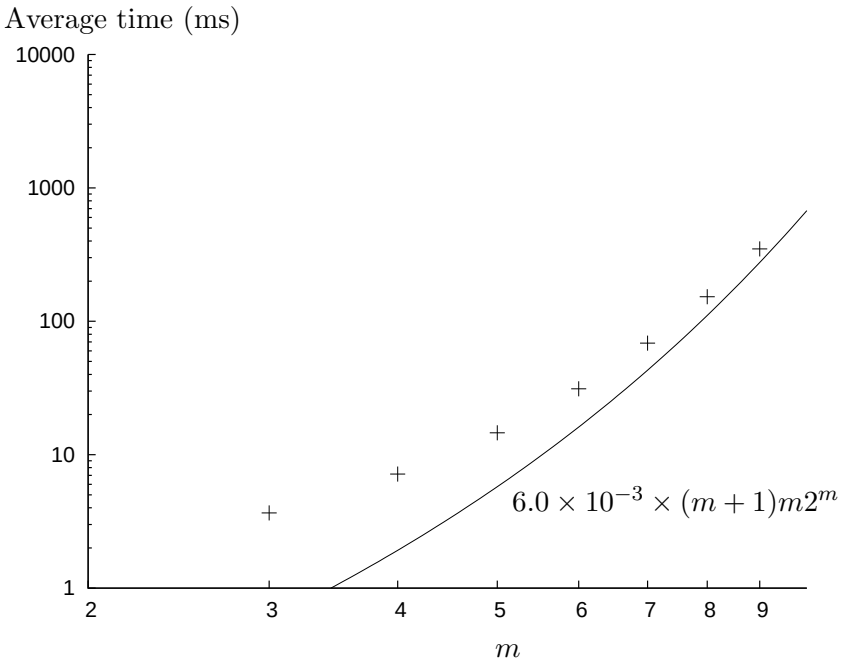

Figure 7.9: Average execution time for each value of m, in milliseconds (logarithmic scale).

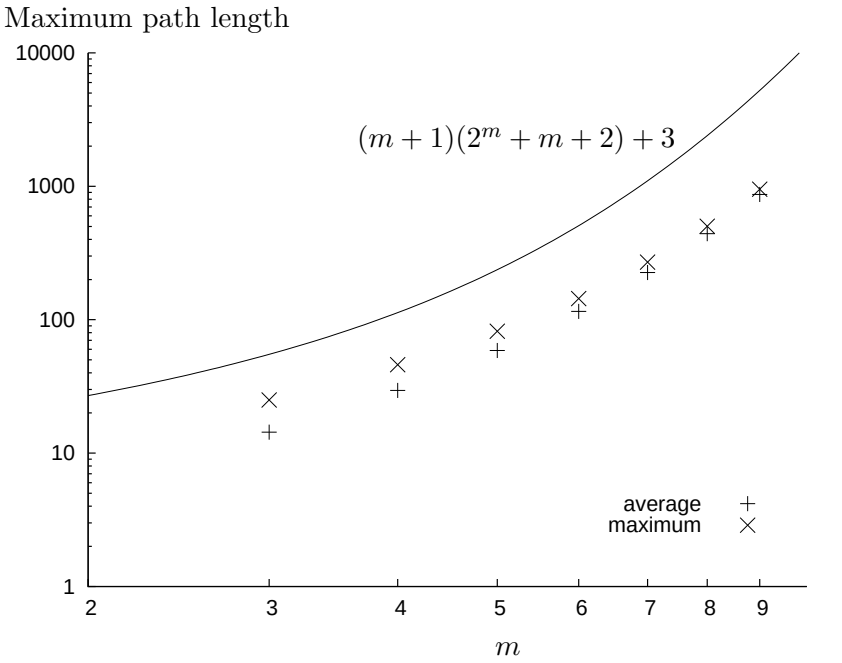

Figure 7.10: Average and maximum of the maximal path lengths collected for each value of m (logarithmic scale).

Let $S = \{s_1 = (\sigma_{s_1}, \pi_{s_1}), s_2 = (\sigma_{s_2}, \pi_{s_2}), s_3 = (\sigma_{s_3}, \pi_{s_3})\}$ and $D = \{d_1 = (\sigma_{d_1}, \pi_{d_1}), d_2 = (\sigma_{d_2}, \pi_{d_2}), d_3 = (\sigma_{d_3}, \pi_{d_3})\}$. Let (x, y) be a pair with $x = |\{\sigma_{s_1}, \sigma_{s_2}, \sigma_{s_3}\}|$ and $y = |\{\sigma_{d_1}, \sigma_{d_2}, \sigma_{d_3}\}|$. We distinguish six cases.

$(3, 3)$ Find three cube-level disjoint paths $\sigma_{s_i} \rightsquigarrow \sigma_{d_{j_i}}$ $(1 \leq i \leq 3)$ by applying Lemma 19 to $\{\sigma_{s_1}, \sigma_{s_2}, \sigma_{s_3}\}$ and $\{\sigma_{d_1}, \sigma_{d_2}, \sigma_{d_3}\}$ in Q_{2^m}.

$(3, 2)$ Assume without loss of generality $\sigma_{d_1} = \sigma_{d_2}$. For one node of $\{d_1, d_2\}$, say d_2, find a path $\rho_2 : d_2 \rightsquigarrow d'_2 \in Q_m(\sigma_{d'_2})$. Find three cube-level disjoint paths by applying Lemma 19 to $\{\sigma_{s_1}, \sigma_{s_2}, \sigma_{s_3}\}$ and $\{\sigma_{d_1}, \sigma_{d'_2}, \sigma_{d_3}\}$ in Q_{2^m}.

$(3, 1)$ For each of all nodes d_j $(1 \leq j \leq 3)$, find a path $\rho_j : d_j \rightsquigarrow d'_j \in Q_m(\sigma_{d'_j})$. Find three fault-free cube-level disjoint paths considering σ_{d_1} faulty by applying Lemma 19 to $\{\sigma_{s_1}, \sigma_{s_2}, \sigma_{s_3}\}$ and $\{\sigma_{d'_1}, \sigma_{d'_2}, \sigma_{d'_3}\}$ in Q_{2^m}, and with $F = \{\sigma_{d_1}\}$.

$(2, 2)$ Assume without loss of generality $\sigma_{s_1} = \sigma_{s_2}$ and $\sigma_{d_1} = \sigma_{d_2}$. For one node of $\{s_1, s_2\}$, say s_2, find a path $\tau_2 : s_2 \rightsquigarrow s'_2 \in Q_m(\sigma_{s'_2})$, and for one node of $\{d_1, d_2\}$, say d_2, find a path $\rho_2 : d_2 \rightsquigarrow d'_2 \in Q_m(\sigma_{d'_2})$. Find three cube-level disjoint paths by applying Lemma 19 to $\{\sigma_{s_1}, \sigma_{s'_2}, \sigma_{s_3}\}$ and $\{\sigma_{d_1}, \sigma_{d'_2}, \sigma_{d_3}\}$ in Q_{2^m}.

$(2, 1)$ Assume without loss of generality $\sigma_{s_1} = \sigma_{s_2}$. For one node of $\{s_1, s_2\}$, say s_2, find a path $\tau_2 : s_2 \rightsquigarrow s'_2 \in Q_m(\sigma_{s'_2})$, and for each of all nodes d_j $(1 \leq j \leq 3)$, find a path $\rho_j : d_j \rightsquigarrow d'_j \in Q_m(\sigma_{d'_j})$. Find three fault-free cube-level disjoint paths considering σ_{d_1} faulty by applying Lemma 19 to $\{\sigma_{s_1}, \sigma_{s'_2}, \sigma_{s_3}\}$ and $\{\sigma_{d'_1}, \sigma_{d'_2}, \sigma_{d'_3}\}$ in Q_{2^m}, and with $F = \{\sigma_{d_1}\}$.

$(1, 1)$ Assume $N(s_2) \cap Q_m(\sigma_{s_2}) = \{s_1, s_3\}$ and $N(d_2) \cap Q_m(\sigma_{d_2}) = \{d_1, d_3\}$. For two nodes of S including s_2, say s_2, s_3, find two paths $\tau_2 : s_2 \rightsquigarrow s'_2 \in Q_m(\sigma_{s'_2})$ and $\tau_3 : s_3 \rightsquigarrow s'_3 \in Q_m(\sigma_{s'_3})$. For two nodes of D including d_2, say d_2, d_3, find two paths $\rho_2 : d_2 \rightsquigarrow d'_2 \in Q_m(\sigma_{d'_2})$ and $\rho_3 : d_3 \rightsquigarrow d'_3 \in Q_m(\sigma_{d'_3})$. Find three cube-level disjoint paths by applying Lemma 19 to $\{\sigma_{s_1}, \sigma_{s'_2}, \sigma_{s'_3}\}$ and $\{\sigma_{d_1}, \sigma_{d'_2}, \sigma_{d'_3}\}$ in Q_{2^m}.

The cube-level paths obtained are then converted to HHC-level paths by applying CONV as in Step 4 of Case II. Note that two shortest paths may exist for CONV inside a subcube $Q_m(\sigma)$ with either $|S(\sigma)| = 2$ or $|D(\sigma)| = 2$, at most one of them may be blocked by another destination node, but another path can be used.

The maximum path length remains unchanged to that of HHC-S2S. Time complexity is constant since $m = 2$.

Chapter 8

Conclusion

Throughout this work we designed several disjoint-path routing algorithms for different interconnection networks. As required by following chapters, we first proposed in Chapter 4 a node-to-set disjoint-path routing algorithm in hypercubes. Inside an n-dimensional hypercube, given a source node s, a set D of k ($k \leq n$) destination nodes and a set of at most $n-k$ faulty neighbours of s, this algorithm finds k fault-free disjoint paths between s and each node of D of lengths at most $n+1$ in $O(kn)$ time complexity. Then in Chapter 5 we described a node-to-set disjoint-path routing algorithm in perfect hierarchical hypercubes. Inside an HHC_{2^m+m}, given a source node and a set of k ($k \leq m+1$) destination nodes, this algorithm finds k disjoint paths between the source node and the destination nodes of lengths at most $m2^m + 2^m + 2m + 4$ in $O(km2^m)$ time complexity. In Chapter 6 we introduced a node-to-set disjoint-path routing algorithm in metacubes. Inside a metacube $MC(k, m)$, given one source node and a set of n ($n \leq k+m$) destination nodes, this algorithm finds n disjoint paths between the source node and the destination nodes of lengths at most $(m2^k + n)(k + 1) + k + 4$ in $O(nm2^k(\log n + k))$ time complexity. Finally, we proposed in Chapter 7 a set-to-set disjoint-path routing algorithm in perfect hierarchical hypercubes. Inside an HHC_{2^m+m}, for a set of source nodes and a set of destination nodes, both of cardinality k ($k \leq m+1$), this algorithm finds k disjoint paths between the source nodes and the destination nodes of lengths at most $(m+1)(2^m + k + 1) + 3$ in $O(km2^m)$ time complexity. As empirical evaluations were conducted to challenge theoretical results, we could measure the practical efficiency of the experimented algorithms, sometimes witnessing significant gain over previous results [9].

We can summarize our research by noting that inside hypercube-based networks, a common method to solve disjoint-path routing problems is to make use of hypercube disjoint-path routing algorithms like the one described in Chapter 4. However, depending on the topology considered, it is rarely possible to directly apply such a hypercube algorithm onto the

original topology. Actually, in order to retain path disjointness, an important and often difficult (see Chapter 7) preprocessing task is required. Thus much attention is ought to be paid on this preliminary part before any further developments of the algorithm can be made. Nevertheless, while this preprocessing task may drain much of the researcher efforts and algorithm difficulty, the time complexity of the algorithm is usually overshadowed by the conversion of the paths found by the hypercube routing algorithm back to paths inside the original topology. This can be easily understood as the number of nodes included in the original topology is a lot greater than its degree, that is the maximum number of disjoint paths that can be selected.

Regarding future works, we can note that many interconnection networks briefly introduced in Chapter 2 do not have any solution to common routing problems such as node-to-set, set-to-set or k-pairwise disjoint-path routing [17, 84, 73, 82]. Also, because faults are likely to occur in modern supercomputers including hundred of thousands of CPU nodes, fault-tolerance as well as cluster-fault tolerance are two important topics to be addressed. Even if the path disjointness property of the algorithms introduced in Chapter 5, 6 and 7 can be considered as one solution to fault-tolerant routing by constructing disjoint paths, alternative approaches, natively designed to detect and avoid faults when generating paths, would be more efficient regarding fault-free path generation. Effectively, disjoint-path routing algorithms can be used in faulty environments: once a set of disjoint paths has been generated, an additional step checking each path for the inclusion of faulty nodes can be easily implemented to find a fault-free path, if any. However this approach is not optimal regarding efficiency as it first constructs as many disjoint paths as possible and only then checks the paths for fault inclusion. Hence, a routing algorithm directly handling faults without waiting for the paths to be generated is very likely to be more efficient; no faulty path would be vainly generated, thus increasing efficiency.

Acknowledgments

I would like to thank particularly Pr. Keiichi Kaneko for his continuous help throughout the years spent in his laboratory as Ph.D. student.

I am also deeply appreciative toward the reviewers of my research Pr. Hironori Nakajo, Pr. Matsuaki Terada, Pr. Konosuke Kawashima and Pr. Mario Nakamori who gave me precious advices and suggestions to construct and improve my thesis.

I would like to express my sincere gratitude to Pr. John Boxall and Pr. Yoshimi Egawa who supervised my research activities for my master degree.

Finally, I warmly thank my family for their unfailing support during my college years.

Thank you.

Bibliography

[1] Mokhtar A. Aboelaze. Mlh: A hierarchical hypercube network. *Networks*, Vol. 28, No. 3, pp. 157–165, October 1996.

[2] Sheldon B. Akers, Dov Harel, and Balakrishnan Krishnamurthy. The star graph: an attractive alternative to the n-cube. In *Proceedings of the International Conference on Parallel Processing (ICPP-87)*, pp. 393–400, 1987.

[3] Sheldon B. Akers and Balakrishnan Krishnamurthy. A group-theoretic model for symmetric interconnection networks. *IEEE Transactions on Computers*, Vol. C-38, No. 4, pp. 555–566, April 1989.

[4] Jean-Claude Bermond and Claudine Peyrat. De Bruijn and Kautz networks: A competitor for the hypercube? *Hypercube and Distributed Computers*, 1989.

[5] Laxmi N. Bhuyan and Dharma P. Agrawal. Generalized hypercube and hyperbus structures for a computer network. *IEEE Transactions on Computers*, Vol. C-33, No. 4, pp. 323–333, April 1984.

[6] Stefan Birrer and Fabian E. Bustamante. Resilient peer-to-peer multicast from the ground up. In *Proceedings of the 3rd IEEE International Symposium on Network Computing and Applications (NCA)*, pp. 351–355, August 2004.

[7] Arthur S. Bland, Ricky A. Kendall, Douglas B. Kothe, James H. Rogers, and Galen M. Shipman. Jaguar: The world's most powerful computer. In *Proceedings of the Cray User Group Conference*, May 2009.

[8] Antoine Bossard and Keiichi Kaneko. Optimal node-to-set disjoint-path routing in hypercube. *Submitted to Information Processing Letters*, February 2011.

[9] Antoine Bossard, Keiichi Kaneko, and Shietung Peng. A new node-to-set disjoint-path algorithm in perfect hierarchical hypercubes. *The Computer Journal*, Vol. 54, No. 8, pp. 1372–1381, August 2011.

[10] Dick Brownell. *SGI Origin 3000 Series Technical Configuration Owner's Guide*. Silicon Graphics, Inc, January 2001. 007-4311-002.

[11] Rocky K. C. Chang and Hong Y. Wang. Routing properties of a recursive interconnection network. *Journal of Parallel and Distributed Computing*, Vol. 61, No. 6, pp. 838–849, June 2001.

[12] Gen-Huey Chen and Dyi-Rong Duh. Topological properties, communication and computation on wk-recursive networks. *Networks*, Vol. 24, No. 6, pp. 303–317, September 1994.

[13] Wei-Kuo Chiang and Rong-Jaye Chen. Topological properties of hierarchical cubic networks. *Journal of Systems and Architecture*, Vol. 42, No. 4, pp. 289–307, November 1996.

[14] Sheshayya A. Choudoum and V. Sunitha. Augmented cubes. *Networks*, Vol. 40, No. 2, pp. 71–84, September 2002.

[15] Tein Y. Chung and Dharma P. Agrawal. Design and analysis of multidimensional manhattan street networks. *IEEE Transactions on Communications*, Vol. 41, No. 2, pp. 295–298, February 1993.

[16] David S. Cohen and Manuel Blum. On the problem of sorting burnt pancakes. *Discrete Applied Mathematics*, Vol. 61, No. 2, pp. 105–120, July 1995.

[17] Peter F. Corbett. Rotator graphs: An efficient topology for point-to-point multiprocessor networks. *IEEE Transactions Parallel and Distributed Systems*, Vol. 3, No. 5, pp. 622–626, September 1992.

[18] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition*. MIT Press and McGraw-Hill, 2001.

[19] Paul Cull and Shawn M. Larson. The möbius cubes. *IEEE Transactions on Computers*, Vol. 44, No. 5, pp. 647–659, May 1995.

[20] William J. Dally. Performance analysis of k-ary n-cube interconnection networks. *IEEE Transactions on Computers*, Vol. C-39, No. 6, pp. 775–785, June 1990.

[21] William J. Dally. Express cubes: Improving the performance of k-ary n-cube interconnection networks. *IEEE Transactions on Computers*, Vol. C-40, No. 9, pp. 1016–1023, September 1991.

[22] Debasish Das, Mallika De, and Bhabani P. Sinha. A new network topology with multiple meshes. *IEEE Transactions on Computers*, Vol. 48, No. 5, pp. 536–551, May 1999.

[23] Nicolaas G. de Bruijn. A combinatorial problem. *Koninklijke Nederlandse Akademie v. Wetenschappen*, Vol. 49, pp. 758–764, 1946.

[24] Reinhard Diestel. *Graph Theory*. Springer-Verlag Heidelberg, 2005.

[25] Jose Duato, Sudhakar Yalamanchili, and Lionel M. Ni. *Interconnection Networks: An Engineering Approach*. IEEE Computer Society Press, 1997.

[26] Dyi-Rong Duh and Gen-Huey Chen. Topological properties of wk-recursive networks. *Journal of Parallel and Distributed Computing*, Vol. 23, No. 3, pp. 468–474, December 1994.

[27] Dyi-Rong Duh, Gen-Huey Chen, and Jywe-Fei Fang. Algorithms and properties of a new two-level network with folded hypercubes as basic modules. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 6, No. 7, pp. 714–723, July 1995.

[28] Jack Edmonds and Richard M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, Vol. 19, No. 2, pp. 248–264, 1972.

[29] Kemal Efe. The crossed cube architecture for parallel computation. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 3, No. 5, pp. 513–524, September 1992.

[30] Ahmed El-Amawy and Shahram Latifi. Properties and performance of folded hypercubes. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 2, No. 1, pp. 31–42, January 1991.

[31] Vance Faber and Jim W. Moore. High-degree, low-diameter interconnection networks with vertex symmetry: The directed case. *Los Alamos National Lab.*, Vol. LA-UR-88-1051, 1988.

[32] Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, and Shriram Krishnamurthi. The drscheme project: An overview. *SIGPLAN Notices*, Vol. 33, No. 6, pp. 17–23, June 1998.

[33] Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, and Shriram Krishnamurthi. The teachscheme! project: Computing and programming for every student. *Computer Science Education*, Vol. 14, No. 1, pp. 55–77, January 2004.

[34] Robert Bruce Findler, John Clements, Cormac Flanagan, Matthew Flatt, Shriram Krishnamurthi, Paul Steckler, and Matthias Felleisen. Drscheme: a programming environment for scheme. *Journal of Functional Programming*, Vol. 12, No. 2, pp. 159–182, March 2002.

[35] Lester Randolph Ford and Delbert Ray Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, Vol. 8, pp. 399–404, 1956.

[36] Ada W. Fu and Siu-Cheung Chau. Cyclic-cubes: A new family of interconnection networks of even fixed-degrees. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 9, No. 12, pp. 1253–1268, December 1998.

[37] Jung-Sheng Fu, Gen-Huey Chen, and Dyi-Rong Duh. Node-disjoint paths and related problems on hierarchical cubic networks. *Networks*, Vol. 40, No. 3, pp. 142–154, October 2002.

[38] Alan Gara, Matthias A. Blumrich, Dong Chen, George L.-T. Chiu, Paul Coteus, Mark E. Giampapa, Ruud A. Haring, Philip Heidelberger, Dirk Hoenicke, Gerard V. Kopcsay, Thomas A. Liebsch, Martin Ohmacht, Burkhard D. Steinmacher-Burow, Todd Takken, and Pavlos Vranas. Overview of the Blue Gene/L system architecture. *IBM Journal of Research and Development*, Vol. 49, No. 2,3, pp. 195–212, March 2005.

[39] Luisa Gargano, Ugo Vaccaro, and Angela Vozella. Fault tolerant routing in the star and pancake interconnection networks. *Information Processing Letters*, Vol. 45, No. 6, pp. 315–320, April 1993.

[40] William H. Gates and Christos H. Papadimitriou. Bounds for sorting by prefix reversal. *Discrete Mathematics*, Vol. 27, pp. 47–57, 1979.

[41] Kanad Ghose and Kiran R. Desai. Hierarchical cubic networks. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 6, No. 4, pp. 427–435, April 1995.

[42] Andrew V. Goldberg and Robert E. Tarjan. A new approach to the maximum flow problem. In *Proceedings of the 18th annual ACM symposium on Theory of computing*, pp. 136–146, 1986.

[43] Oded Goldreich, Arnold L. Rosenberg, and Alan L. Selman. *Theoretical Computer Science: Essays in Memory of Shimon Even*. Springer, 2006. Yefim Dinitz, Dinitz' Algorithm: The Original Version and Even's Version, pages 218–240.

[44] Frank Gray. Pulse code communication. Patent 2,632,058, U.S., March 1953. (filed Nov. 1947).

[45] Jonathan Gross and Jay Yellen. *Graph Theory and Its Applications*. CRC Press, 1999.

[46] Qian-Ping Gu, Satoshi Okawa, and Shietung Peng. Set-to-set fault tolerant routing in hypercubes. *IEICE Transactions on Fundamentals*, Vol. E79-A, No. 4, pp. 483–488, April 1996.

[47] Qian-Ping Gu and Shietung Peng. Linear time algorithms for fault tolerant routing in hypercubes and star graphs. *IEICE Transactions on Information and Systems*, Vol. E78-D, No. 9, pp. 1171–1177, September 1995.

[48] Qian-Ping Gu and Shietung Peng. Node-to-node cluster fault tolerant routing in star graphs. *Information Processing Letters*, Vol. 56, No. 1, pp. 29–35, October 1995.

[49] Qian-Ping Gu and Shietung Peng. An efficient algorithm for node-to-node routing in hypercubes with faulty clusters. *The Computer Journal*, Vol. 39, No. 1, pp. 14–19, November 1996.

[50] Qian-Ping Gu and Shietung Peng. Optimal algorithms for node-to-node fault tolerant routing in hypercubes. *The Computer Journal*, Vol. 39, No. 7, pp. 626–629, 1996.

[51] Qian-Ping Gu and Shietung Peng. Set-to-set fault tolerant routing in star graphs. *IEICE Transactions on Information and Systems*, Vol. E79-D, No. 4, pp. 282–289, April 1996.

[52] Qian-Ping Gu and Shietung Peng. k-pairwise cluster fault tolerant routing in hypercubes. *IEEE Transactions on Computers*, Vol. 46, No. 9, pp. 1042–10, September 1997.

[53] Qian-Ping Gu and Shietung Peng. Node-to-set disjoint paths with optimal length in star graphs. *IEICE Transactions on Information and Systems*, Vol. E80-D, No. 4, pp. 425–433, April 1997.

[54] Qian-Ping Gu and Shietung Peng. An efficient algorithm for k-pairwise disjoint paths in star graphs. *Information Processing Letters*, Vol. 67, No. 6, pp. 283–287, September 1998.

[55] Qian-Ping Gu and Shietung Peng. Node-to-set and set-to-set cluster fault tolerant routing in hypercubes. *Parallel Computing*, Vol. 24, No. 8, pp. 1245–1261, August 1998.

[56] Qian-Ping Gu and Shietung Peng. Cluster fault-tolerant routing in star graphs. *Networks*, Vol. 35, No. 1, pp. 83–90, January 2000.

[57] Qian-Ping Gu and Shietung Peng. An efficient algorithm for the k-pairwise disjoint paths problem in hypercubes. *Journal of Parallel and Distributed Computing*, Vol. 60, No. 6, pp. 764–774, June 2000.

[58] John Hayes, Trevor Mudge, Quentin Stout, Stephen Colley, and John Palmer. A microprocessor-based hypercube supercomputer. *IEEE Micro*, Vol. 6, No. 5, pp. 6–17, October 1986.

[59] Wen-Jing Hsu. Fibonacci cubes - a new interconnection topology. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 4, No. 1, pp. 3–12, January 1993.

[60] Tatsuya Iwasaki and Keiichi Kaneko. Fault-tolerant routing in burnt pancake graphs. *Information Processing Letters*, Vol. 110, No. 14-15, pp. 535–538, July 2010.

[61] Nagateru Iwasawa, Antoine Bossard, and Keiichi Kaneko. Set-to-set disjoint path routing algorithm in burnt pancake graphs. In *Proceedings of the 26th International Conference on Computers and Their Applications (CATA)*, pp. 21–26, March 2011.

[62] Nagateru Iwasawa, Tatsuro Watanabe, Tatsuya Iwasaki, and Keiichi Kaneko. Cluster-fault-tolerant routing in burnt pancake graphs. In *Proceedings of the 10th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP)*, pp. 264–274, May 2010.

[63] Keiichi Kaneko. An algorithm for node-to-set disjoint paths problem in burnt pancake graphs. *IEICE Transactions on Information and Systems*, Vol. E86-D, No. 12, pp. 2588–2594, December 2003.

[64] Keiichi Kaneko. Internally-disjoint paths problem in bi-rotator graphs. *IEICE Transactions on Information and Systems*, Vol. E88-D, No. 7, pp. 1678–1684, July 2005.

[65] Keiichi Kaneko. An algorithm for node-to-set disjoint paths problem in bi-rotator graphs. *IEICE Transactions on Information and Systems*, Vol. E89-D, No. 2, pp. 647–653, February 2006.

[66] Keiichi Kaneko. Routing problems in incomplete pancake graphs. In *Proceedings of the 7th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD)*, pp. 151–156, June 2006.

[67] Keiichi Kaneko and Shietung Peng. Disjoint paths routing in pancake graphs. In *Proceedings of the 7th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, pp. 254–259, December 2006.

[68] Keiichi Kaneko and Shietung Peng. Node-to-set disjoint paths routing in dual-cube. In *Proceedings of the 9th International Symposium on*

Parallel Architectures, Algorithms, and Networks (ISPAN), pp. 77–82, May 2008.

[69] Keiichi Kaneko and Shietung Peng. Set-to-set disjoint paths routing in dual-cubes. In *Proceedings of the 9th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, pp. 129–136, May 2008.

[70] Keiichi Kaneko and Naoki Sawada. An algorithm for node-to-node disjoint paths problem in burnt pancake graphs. *IEICE Transactions on Information and Systems*, Vol. E90-D, No. 1, pp. 306–313, January 2007.

[71] Keiichi Kaneko and Yasuto Suzuki. Node-to-set disjoint paths problem in rotator graphs. In *Proceedings of the 6th Asian Computing Science Conference (ASIAN)*, volume 1961 of *Lecture Notes in Computer Science*, pp. 119–132. Springer, November 2000.

[72] Keiichi Kaneko and Yasuto Suzuki. Node-to-set disjoint paths problem in pancake graphs. *IEICE Transactions on Information and Systems*, Vol. E86-D, No. 9, pp. 1628–1633, September 2003.

[73] Keiichi Kaneko and Yasuto Suzuki. Node-to-node internally disjoint paths problem in bubble-sort graphs. In *Proceedings of the 10th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC)*, pp. 173–182, March 2004.

[74] J. Mohan Kumar and Lalit M. Patnaik. Extended hypercube: A hierarchical interconnection network of hypercubes. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 3, No. 1, pp. 45–57, January 1992.

[75] Shahram Latifi, Hyosun Ko, and Pradip K. Srimani. Node-to-set vertex disjoint paths in hypercube networks. Technical Report CS-98-107, Colorado State University, 1998.

[76] James Laudon and Daniel Lenoski. System overview of the SGI Origin 200/2000 product line. In *Proceedings of the IEEE Compcon '97*, pp. 150–156, February 1997.

[77] Yamin Li and Shietung Peng. Dual-cubes: a new interconnection network for high-performance computer clusters. In *Proceedings of the 2000 International Computer Symposium, Workshop on Computer Architecture*, pp. 51–57, December 2000.

[78] Yamin Li and Shietung Peng. Fault-tolerant routing and disjoint paths in dual-cube: a new interconnection network. In *Proceedings of the*

8th International Conference on Parallel and Distributed Systems (ICPADS), pp. 315–322, June 2001.

[79] Yamin Li, Shietung Peng, and Wanming Chu. Fault-tolerant routing in metacube. In *Proceedings of the 3rd International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, pp. 343–350, September 2002.

[80] Yamin Li, Shietung Peng, and Wanming Chu. Multinode broadcasting in metacube. In *Proceedings of the 3rd International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pp. 401–408, June 2002.

[81] Yamin Li, Shietung Peng, and Wanming Chu. Disjoint paths in metacube. In *Proceedings of the 15th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS)*, pp. 43–50, November 2003.

[82] Yamin Li, Shietung Peng, and Wanming Chu. Recursive dual-net: A new versatile network for supercomputers of the next generation. *Journal of the Chinese Institute of Engineers*, Vol. 32, No. 7, pp. 931–938, 2009.

[83] Yamin Li, Shietung Peng, and Wanming Chu. Metacube - a versatile family of interconnection networks for extremely large-scale supercomputers. *The Journal of Supercomputing*, Vol. 53, No. 2, pp. 329–351, August 2010.

[84] Hon-Ren Lin and Chiun-Chieh Hsu. Topological properties of birotator graphs. *IEICE Transactions on Information and Systems*, Vol. E86-D, No. 10, pp. 2172–2178, October 2003.

[85] Zhen Liu and Ting-Yi Sung. Routing and transmitting problems in de bruijn networks. *IEEE Transactions on Computers*, Vol. 45, No. 9, pp. 1056–1062, September 1996.

[86] Qutaibah M. Malluhi and Magdy A. Bayoumi. The hierarchical hypercube: A new interconnection topology for massively parallel systems. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 5, No. 1, pp. 17–30, January 1994.

[87] Jyh-Wen Mao and Chang-Biau Yang. Shortest path routing and fault-tolerant routing on de bruijn networks. *Networks*, Vol. 35, No. 3, pp. 207–215, May 2000.

[88] Takumi Maruyama, Toshio Yoshida, Ryuji Kan, Iwao Yamazaki, Shuji Yamamura, Noriyuki Takahashi, Mikio Hondou, and Hiroshi Okano.

Sparc64 VIIIfx: A new-generation octocore processor for petascale computing. *IEEE Micro*, Vol. 30, No. 2, pp. 30–40, March-April 2010.

[89] Nicholas F. Maxemchuk. Routing in the manhattan street network. *IEEE Transactions on Communications*, Vol. COM-35, No. 5, pp. 503–512, May 1987.

[90] Oliver A. McBryan and Eric F. van de Velde. Hypercube algorithms and implementations. *SIAM Journal on Scientific and Statistical Computing*, Vol. 8, No. 2, pp. 227–287, March 1987.

[91] Dikran S. Meliksetian and C. Y. Roger Chen. Optimal routing algorithm and the diameter of the cube-connected cycles. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 4, No. 10, pp. 1172–1178, October 1993.

[92] Karl Menger. Zur allgemeinen Kurventheorie. *Fundamenta Mathematicae*, Vol. 10, pp. 96–115, 1927.

[93] Chwei-King Mok and Nader F. Mir. An efficient interconnection network for large-scale computer communications applications. *Journal of Network and Computer Applications*, Vol. 23, No. 2, pp. 59–75, April 2000.

[94] San Murugesan. Harnessing green it: Principles and practices. *IT Professional*, Vol. 10, No. 1, pp. 24–33, February 2008.

[95] Sabine Öhring and Sajal K. Das. Folded petersen cube networks: New competitors for the hypercubes. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 7, No. 2, pp. 151–168, February 1996.

[96] Shietung Peng and Keiichi Kaneko. Set-to-set disjoint paths routing in pancake graphs. In *Proceedings of the 18th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS)*, pp. 253–258, November 2006.

[97] Franco P. Preparata and Jean Vuillemin. The cube-connected cycles: a versatile network for parallel computation. *Communications of the ACM*, Vol. 24, No. 5, pp. 300–309, May 1981.

[98] Bob Quinn and Kevin Almeroth. Ip multicast applications: Challenges and solutions. RFC 3170, University of California, Santa Barbara, 2001.

[99] Michael O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the ACM*, Vol. 36, No. 2, pp. 335–348, April 1989.

[100] Youcef Saad and Martin H. Schultz. Topological properties of hypercubes. *IEEE Transactions on Computers*, Vol. 37, No. 7, pp. 867–872, July 1988.

[101] Charles L. Seitz. The cosmic cube. *Communications of the ACM*, Vol. 28, No. 1, pp. 22–33, January 1985.

[102] SGI. *Origin2000 Rackmount owner's guide, 007-3456-003*, 1997. http://techpubs.sgi.com/.

[103] Ming-Yang Su, Gen-Huey Chen, and Dyi-Rong Duh. A shortest-path routing algorithm for incomplete wk-recursive networks. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 8, No. 4, pp. 367–379, April 1997.

[104] Ming-Yang Su, Hui-Ling Huang, Gen-Huey Chen, and Dyi-Rong Duh. Node-disjoint paths in incomplete wk-recursive networks. *Parallel Computing*, Vol. 26, No. 13-14, pp. 1925–1944, December 2000.

[105] Yuzhong Sun, Paul Y.S. Cheung, and Xiaola Lin. Recursive cube of rings: A new topology for interconnection networks. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 11, No. 3, pp. 275–286, March 2000.

[106] Yasuto Suzuki and Keiichi Kaneko. An algorithm for node-disjoint paths in pancake graphs. *IEICE Transactions on Information and Systems*, Vol. E86-D, No. 3, pp. 610–615, March 2003.

[107] Yasuto Suzuki and Keiichi Kaneko. An algorithm for disjoint paths in bubble-sort graphs. *Systems and Computers in Japan*, Vol. 37, No. 12, pp. 27–32, November 2006.

[108] Toshinori Takabatake, Keiichi Kaneko, and Hideo Ito. Hcc: Generalized hierarchical completely-connected networks. *IEICE Transactions on Information and Systems*, Vol. E83-D, No. 6, pp. 1216–1224, June 2000.

[109] The New York Times. Japanese 'K' computer is ranked most powerful. http://www.nytimes.com/2011/06/20/technology/20computer.html, June 2011. Last accessed July 2011.

[110] TOP500. June 2010 list. http://www.top500.org/list/2010/06/100, June 2010. Last accessed January 2011.

[111] TOP500. Japan reclaims top ranking on latest TOP500 list of world's supercomputers. http://www.top500.org/lists/2011/06/press-release, June 2011. Last accessed July 2011.

[112] Emmanouel A. Varvarigos. Optimal communication algorithms for manhattan street networks. *Discrete Applied Mathematics*, Vol. 83, No. 1-3, pp. 303–326, March 1998.

[113] Gennaro D. Vecchia and C. Sanges. An optimized broadcasting technique for wk-recursive topologies. *Future Generation Computer Systems*, Vol. 5, No. 3, pp. 353–357, January 1990.

[114] Jie Wu and Dharma P. Agrawal. Guest editors' introduction: Challenges in designing fault-tolerant routing in networks. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 10, No. 10, pp. 961–963, October 1999.

[115] Jie Wu and Xian-He Sun. Optimal cube-connected cube multicomputers. *Journal of Microcomputer Applications*, Vol. 17, No. 2, pp. 135–146, April 1994.

[116] Ruei-Yu Wu, Gen-Huey Chen, Yu-Liang Kuo, and Gerard J. Chang. Node-disjoint paths in hierarchical hypercube networks. *Information Sciences*, Vol. 177, No. 19, pp. 4200–4207, October 2007.

[117] Xiaofan Yang, Graham M. Megson, and David J. Evans. An oblivious shortest-path routing algorithm for fully connected cubic networks. *Journal of Parallel and Distributed Computing*, Vol. 66, No. 10, pp. 1294–1303, October 2006.

[118] Yulu Yang, Akira Funahashi, Akiya Jouraku, Hiroaki Nishi, Hideharu Amano, and Toshinori Sueyoshi. Recursive diagonal torus: An interconnection network for massively parallel computers. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 12, No. 7, pp. 701–715, July 2001.

[119] Chi-Hsiang Yeh and Emmanouel A. Varvarigos. Macro-star networks: Efficient low-degree alternatives to star graphs. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 9, No. 10, pp. 987–1003, October 1998.

[120] Sang K. Yun and Kyu H. Park. Comments on "hierarchical cubic networks". *IEEE Transactions on Parallel and Distributed Systems*, Vol. 9, No. 4, pp. 410–414, April 1998.

Index

Appendix A

Related paper

The Computer Journal Advance Access published May 27, 2011

doi:10.1093/comjnl/bxr047

A New Node-to-Set Disjoint-Path Algorithm in Perfect Hierarchical Hypercubes

ANTOINE BOSSARD[1,*], KEIICHI KANEKO[1] AND SHIETUNG PENG[2]

[1]*Graduate School of Engineering, Tokyo University of Agriculture and Technology, Japan*
[2]*Faculty of Computer and Information Science, Hosei University, Japan*
[*]*Corresponding author: 50008834706@st.tuat.ac.jp*

The perfect hierarchical hypercube (HHC) interconnection network, also known as the cube-connected cube, was introduced as a topology for large parallel computers. One of its interesting properties is that it can connect many nodes while retaining a small diameter and a low degree. The first node-to-set disjoint-path routing algorithm in perfect HHCs was previously introduced by Bossard *et al.* [(2011) Node-to-Set Disjoint-Path Routing in Perfect Hierarchical Hypercubes. *Proc. 11th Int. Conf. Computational Science*, Tsukuba, Japan, June 1–3. Elsevier, Amsterdam]. In this paper, we propose a novel solution to the node-to-set disjoint-path routing problem in HHC. Inside a $(2^m + m)$-dimensional HHC, we shall describe an algorithm that can find disjoint paths between a source node and at most $m + 1$ destination nodes of maximum length $O(2^m)$, significantly shorter than the maximum path length $O(m2^m)$ of Bossard *et al.* [(2011) Node-to-Set Disjoint-Path Routing in Perfect Hierarchical Hypercubes. *Proc. 11th Int. Conf. Computational Science*, Tsukuba, Singapore, June 1–3. Elsevier, Amsterdam].

Keywords: cube-connected cube; hierarchical hypercube; interconnection network; disjoint routing; polynomial algorithm

Received 17 December 2010; revised 15 March 2011
Handling editor: Alberto Apostolico

1. INTRODUCTION

As the number of processors bundled in modern supercomputers continuously grows, interconnection networks have become a critical topic. We should take advantage of the huge number of CPU cores in such massively parallel systems without facing heavy overloads or any other obstacle due to the underlying network topology complexity. For this purpose an efficient interconnection network and efficient data routing algorithms are today two major components that must be taken into account when designing such systems.

Beyond simple network structures like hypercubes or rings, several high-performance topologies particularly adapted to massively parallel systems have been proposed: dual-cubes [1], metacubes [2] or pancake graphs [3] are some examples.

The perfect hierarchical hypercube (HHC) interconnection network was introduced by Malluhi and Bayoumi in [4]. Concerned by the interesting properties of such a structure, Wu and Sun described separately the same topology in [5] as the cube-connected cube. The most attracting property of the HHC network is that it can connect many nodes while retaining a small diameter as well as a low degree, compared, for example, to a hypercube of the same size.

Wu *et al.* described in [6] a node-to-node disjoint-path routing algorithm in HHC. Bossard *et al.* proposed in [7] a node-to-set disjoint-path routing algorithm in HHC, which is called HHC-N2S in this paper. HHC-N2S generates paths of maximum length $m2^m+2^m+2m+4$. We shall describe in this paper a node-to-set disjoint-path routing algorithm HHC-iN2S in a (2^m+m)-dimensional HHC improving the maximum path length from $O(m2^m)$ in [7] to $O(2^m)$.

As for different topologies, a similar problem has been solved in dual-cubes by Kaneko and Peng in [1] and, amongst others, in star graphs by Gu and Peng in [8].

We first review in Section 2 some definitions as well as established results used afterwards. Section 3 describes HHC-iN2S. Section 4 proves the algorithm correctness and estimates

its time complexity as well as the theoretical maximum path length. Section 5 performs an empirical evaluation of the algorithm and discuss the results obtained. Finally, Section 6 concludes this paper.

2. PRELIMINARIES

In this section, we first introduce several general notations. Then we recall the definition of the HHC interconnection network. Finally, a few lemmas used in this paper will be proposed.

A path P in a graph is defined as a sequence of edges $(a_1, a_2), (a_2, a_3), \ldots, (a_{k-1}, a_k)$ with each node of the graph appearing in P at most once. P can also be written as $a_1 \rightarrow a_2 \rightarrow \cdots \rightarrow a_k$ or more concisely $a_1 \rightsquigarrow a_k$. The nodes a_1, a_k are called end nodes and the nodes $a_2, \ldots, a_{k-1}$ are called internal nodes of P. The length of a path P, denoted by $L(P)$, is the number of edges in P. The set of the neighbours of a node a is denoted by $N(a)$. The operator $\oplus$ represents the binary exclusive-or operation.

An n-dimensional hypercube Q_n consists of 2^n nodes, and each node has an n-bit unique address. There is an edge between a pair of nodes a and b if and only if their addresses differ by one single bit. We recall that a hypercube has a recursive structure: for any dimension $\delta (0 \leq \delta \leq n-1)$, a Q_n can be reduced in two subcubes Q^0_{n-1} (or simply Q^0) and Q^1_{n-1} (or simply Q^1) of dimension $n-1$. Q^0 (respectively, Q^1) is induced by all the nodes of Q_n whose δth bit is set to 0 (respectively, 1). We assume that a node address in a Q_n can be stored in a fixed number of machine words. Therefore, for two nodes $a, b \in Q_n$, the comparison of a and b, the calculation of their Hamming distance $H(a, b)$ and the detection of the most significant bit set to 1 can be performed in constant time complexity.

An perfect HHC is an undirected graph that has 2^{2^m+m} nodes for any natural number m. It is denoted by HHC_{2^m+m}. Each node of an HHC_{2^m+m} is denoted by a pair of a 2^m-bit sequence, called the subcube ID, and an m-bit sequence, called the processor ID. Two nodes $\boldsymbol{a} = (\sigma_a, \pi_a)$ and $\boldsymbol{b} = (\sigma_b, \pi_b)$ are adjacent if and only if one of the following two conditions holds:

(1) $\sigma_a = \sigma_b$ and $H(\pi_a, \pi_b) = 1$;
(2) $\sigma_a = \sigma_b \oplus 2^{\pi_b}$ and $\pi_a = \pi_b$.

Edges implied by the first condition are called internal edges. The second condition designates external edges. We note that from the first condition, the set of nodes having the same subcube ID σ induces an m-dimensional hypercube, denoted by subcube $Q_m(\sigma)$. A processor ID represents the position of the node inside its subcube. Also the degree of an HHC_{2^m+m} is $m+1$. Finally, an HHC_{2^m+m} is symmetric, $(m+1)$-connected and of diameter 2^{m+1} [4, 6]. Figure 1 represents an HHC_6.

Let us establish or recall several lemmas. First, Lemma 2.1 recalls a basic property of a hypercube.

LEMMA 2.1. *In a Q_n reduced into two subcubes Q^0 and Q^1, for any node a of Q^0(respectively, Q^1) there are n disjoint paths of lengths at most 2 connecting a to n distinct nodes in Q^1(respectively, Q^0).*

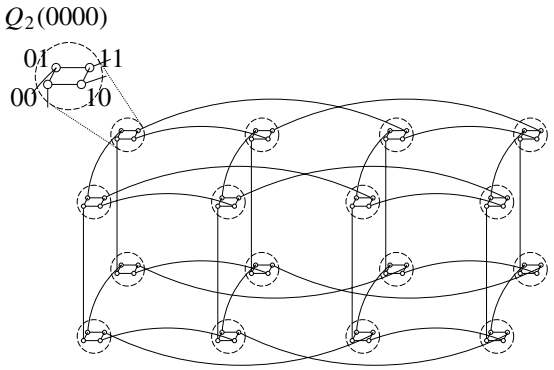

FIGURE 1. $\mathrm{HHC}_6(m = 2)$.

Proof. Assume that Q_n is reduced along a dimension δ and $a \in Q^0$. The $n-1$ paths of lengths 2 are $a \rightarrow a \oplus 2^i \rightarrow a \oplus 2^i \oplus 2^\delta \in Q^1$ with $0 \leq i \leq n-1, i \neq \delta$, and the path of length 1 is $a \rightarrow a \oplus 2^\delta \in Q^1$. □

Secondly, Lemma 2.2 recalls the hypercube node-to-set disjoint-path routing algorithm (Cube-N2S) of [7].

LEMMA 2.2. *In a Q_n, given a node s and a set of k $(\leq n)$ nodes $D = \{d_1, \ldots, d_k\}$, we can find k disjoint paths between s and the nodes of D of lengths at most $n+1$ in $O(kn)$ time complexity.*

Now we define a specific shortest path routing (SPR) algorithm SPR-Gray in a hypercube. SPR-Gray finds a path between two nodes a, b, where $h = H(a, b)$ and $a \oplus b = \sum_{i=0}^{h-1} 2^{\delta_i}$, by successively flipping the bits at the positions of $\delta_0, \ldots, \delta_{h-1}$ according to the order specified by a Gray code. One should note that the nodes on the path from a to b themselves are not following a Gray code, but the dimensions of the flipped bits do. See Example 1 below.

EXAMPLE 1. We give one possible output for a dimension order SPR algorithm and the previously defined SPR-Gray variation:
SPR(00000000, 00111010)

$$00000000 \rightarrow 00000010 \rightarrow 00001010 \rightarrow 00011010 \rightarrow 00111010$$

SPR-Gray(00000000, 00111010)

$$00000000 \rightarrow 00000010 \rightarrow 00001010 \rightarrow 00101010 \rightarrow 00111010$$

For the path returned by SPR, which changes the bits following the dimensions in the ascending order, the bits are changed in the order 1,3,4,5 (binary: 001,011,100,101). The path generated by SPR-Gray changes the bits in the order 1,3,5,4 (binary: 001,011,101,100), which is a subsequence of a Gray code. We call the order specified by a Gray code a Gray code order. In an

SPR in a Q_n, if the bits are flipped according to a Gray code order, we say that the nodes are traversed in Gray code order. An illustration is given in Fig. 2.

Practically, inside a Q_n, SPR-Gray first obtains the differing bit positions $\delta_0, \ldots, \delta_{h-1}$ between the two nodes a, b where $a \oplus b = \sum_{i=0}^{h-1} 2^{\delta_i}$, $h = H(a, b)$. Then it flips successively the bits at the positions $\delta_0, \ldots, \delta_{h-1}$ following the order defined by the function f with $f(\gamma_i) = i$ where $\gamma_0, \ldots, \gamma_{2^g-1}$ is a g-bit Gray code with $g = \lceil \log_2 n \rceil$; that is, if $f(\delta_i) < f(\delta_j)$ holds, the bit at position δ_i is flipped before the bit at position δ_j. Sorting the dimensions $\delta_0, \ldots, \delta_{h-1}$ can be done in $O(n)$ time as well as the corresponding bit flips. Therefore, SPR-Gray has a total time complexity of $O(n)$.

Lemma 2.3 recalls the hypercube fault-tolerant node-to-node routing algorithm (Cube-FTN2N) described by Gu and Peng [9].

LEMMA 2.3. *In a Q_n, given two non-faulty nodes s, d and a set F of at most $n-1$ faulty nodes, we can find a fault-free path from s to d of length at most $n+1$ in $O(n)$ time complexity.*

This algorithm makes at most $\lceil \log_2 |F| \rceil$ hypercube reductions [9]. Gu and Peng assume an arbitrary SPR in Cube-FTN2N but in this paper we use SPR-Gray.

Now we describe an extended hypercube fault-tolerant node-to-set disjoint-path routing algorithm Cube-XFTN2S. In a Q_n, let s be the source node, D be the set of destination nodes and F be the set of faulty nodes such that $|D| + |F| \leq n - 1$. Given a restriction set X with $X \subset N(s)$ and $|X| \leq 1$, Cube-XFTN2S(Q_n, s, D, F, X) returns $|D|$ fault-free disjoint paths from s to D with one of these paths starting with the edge $s \to x$ if $x \in X$. Initially, Cube-XFTN2S is called with non-faulty source and destination nodes, and with $X = \{x\}$. However, since Cube-XFTN2S is recursive, X may become empty and some destination nodes may become faulty. Two cases are distinguished.

Case 1: $|D| = 1$.

Let D be $\{d\}$. If $X = \{x\}$, first reduce Q_n along the dimension $s \oplus x$ into two subcubes Q^0 and Q^1, where $s \in Q^0$ and $x \in Q^1$. Then select the edge $s \to x$. If $d \in Q^1$, connect x to d inside Q^1 with Cube-FTN2N. If $d \in Q^0$, find a fault-free path $d \rightsquigarrow d' \in Q^1$ of at most two edges not including s by enumeration and connect x to d' in Q^1 with Cube-FTN2N.

Otherwise a path going through x has already been created. If $d \in F$ (it means $H(s, d) = 1$), find by enumeration a path $s \rightsquigarrow d$ of length 3 not including a faulty internal node. If $d \notin F$, connect s to d with Cube-FTN2N.

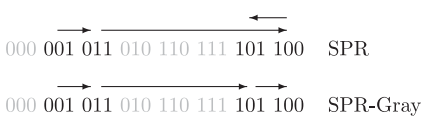

FIGURE 2. A 3-bit Gray code with flipped bit positions by SPR and SPR-Gray.

Case 2: $|D| > 1$.

If $D \cap N(s) \neq \emptyset$, then select an arbitrary node $d \in D \cap N(s)$. If $d \notin F$, then create the trivial path of length 1 $s \to d$ and recursively apply this algorithm by calling Cube-XFTN2S($Q_n, s, D \setminus \{d\}, F \cup \{d\}, X \setminus \{d\}$). If $d \in F$, then find a node $d' \in N(d) \setminus (F \cup D \cup \{s\})$ and recursively apply this algorithm by calling Cube-XFTN2S($Q_n, s, (D \setminus \{d\}) \cup \{d'\}, F, X$).

If $D \cap N(s) = \emptyset$, reduce Q_n into two subcubes Q^0 and Q^1 along a dimension δ such that $D \cap Q^0 \neq \emptyset$ and $D \cap Q^1 \neq \emptyset$. Assume $s \in Q^0$. Let s_1 be the unique neighbour of s into Q^1 ($s_1 = s \oplus 2^\delta$). Let $F_1 = ((F \cap Q^1) \setminus \{s_1\}) \cup \{s'_i \mid s'_i \in N(s_1) \cap Q^1, s_i (= s'_i \oplus 2^\delta) \in F\}$. We apply recursively this algorithm onto Q^1 to obtain a set $\mathcal{C}_1$ of paths $s_1 \to s'_i \rightsquigarrow d_j$ (Cube-XFTN2S($Q^1, s_1, D \cap Q^1, F_1, \emptyset$)). If $s_1 \in F$, we update all the paths in $\mathcal{C}_1$ to be connected to s by replacing in each path the edge $s_1 \to s'_i$ by the subpath $s \to s_i \to s'_i$. If $s_1 \notin F$, we update all the paths in $\mathcal{C}_1$, except one, to be connected to s by replacing the edge $s_1 \to s'_i$ by the subpath $s \to s_i \to s'_i$ and the remaining path is extended to be connected to s simply using the edge $s \to s_1$. Let $F_0 = (F \cap Q^0) \cup \{s_i \mid (s \to s_i \to s'_i \rightsquigarrow d_j) \in \mathcal{C}_1\}$. Finally, we apply this algorithm recursively onto Q^0 (Cube-XFTN2S($Q^0, s, D \cap Q^0, F_0, (X \cap Q^0) \setminus F_0$)).

Starting from Case 2, Cube-XFTN2S makes at most $|D| - 1$ hypercube reductions before reaching Case 1. In Case 1, Cube-XFTN2S makes at most one hypercube reduction along $s \oplus x$, and Cube-FTN2N makes at most $\lceil \log_2 |F| \rceil$ hypercube reductions. Hence, Cube-XFTN2S performs in total at most $|D| + \lceil \log_2 |F| \rceil$ hypercube reductions.

REMARK 1. SPR-Gray is only called by Cube-FTN2N. Cube-FTN2N is only called in Case 1. If Cube-FTN2N is called, SPR-Gray is applied inside a subcube of dimension at least $n - (|D| + \lceil \log_2 |F| \rceil)$. Precisely, SPR-Gray is applied inside a subcube of dimension that is exactly $n - (|D| + \lceil \log_2 |F| \rceil)$ in the case $X \neq \emptyset$ and $d \in Q^0$ of Case 1. Otherwise, that is, if Cube-FTN2N is not called, a path of length 3 is created in Case 1. Therefore, any path includes at most $\max\{(|D| - 1) + 1 + \lceil \log_2 |F| \rceil + 2, |D| + 2\} + 1 = |D| + \lceil \log_2 |F| \rceil + 3$ nodes traversed not in Gray code order (Fig. 3).

From this discussion, we can state the following lemma.

LEMMA 2.4. *In a Q_n, given a non-faulty source node s, a set of $k (\leq n)$ destination nodes $D = \{d_1, \ldots, d_k\}$, a set F of at most $n-k-1$ faulty nodes and a restriction set X with $X \subset N(s) \setminus F$ and $|X| \leq 1$, we can find k disjoint paths $s \rightsquigarrow d_i (1 \leq i \leq k)$*

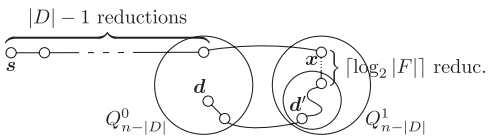

FIGURE 3. A path including $|D| + \lceil \log_2 |F| \rceil + 3$ nodes traversed not in Gray code order.

not including a faulty internal node, one starting with the edge $s \to x$ *if* $\exists x \in X$*, of maximum length* $n+3$ *in* $O(kn)$ *time complexity.*

Proof. We prove this lemma by induction on k.

If $|D| = 1$, if $\exists x \in X$, then we first reduce Q_n along the dimension $s \oplus x$ so that $s \in Q^0$ and $x \in Q^1$. If $d \in Q^1$, then x is connected to d in Q^1 with Cube-FTN2N in at most $(n-1)+1 = n$ edges. Then x is connected to s in one edge and the total path length is thus at most $n+1$. If $d \in Q^0$, by Lemma 2.1 there are n disjoint paths of lengths at most 2 connecting d to distinct nodes in Q^1. Since $|\{s\} \cup F| \leq n-k = n-1$, there is at least one path $d \rightsquigarrow d' \in Q^1$ of these n disjoint paths that does not include s or a faulty node. We see that x is connected to d' in Q^1 with Cube-FTN2N in at most $(n-1)+1 = n$ edges. Finally, s is connected to x in one edge, the total length of this path is thus $1+n+2 = n+3$. Otherwise (i.e. $X = \emptyset$), if $d \in F$, it is caused by a recursive call from Case 2 by introducing a set of faulty nodes $F_1 = ((F \cap Q^1) \setminus \{s_1\}) \cup \{s_i' \mid s_i' \in N(s_1) \cap Q^1, s_i(= s_i' \oplus 2^\delta) \in F\}$, where s_i' is also included in the set $D \cap Q^1$, which is the set of destination nodes in the recursive call Cube-XFTN2S(Q^1, s_1, $D \cap Q^1$, F_1, $\emptyset$). Therefore, if $d \in F$, it means that d is adjacent to the new source node s, which was originally s_1. We can connect s to d with a path of length 3 including non-faulty internal nodes. Since $|F| \leq n-2$ ($F \ni d$), we can find a path $s \to a \to a \oplus d \oplus s \to d$ not including a faulty internal node by enumerating the n disjoint paths for $a \in N(s)$. If $d \notin F$, we connect s to d with Cube-FTN2N with at most $n+1$ edges. Hence, in the case $|D| = 1$ Lemma 2.4 holds.

We assume that Lemma 2.4 holds for any $|D| < k$. Let us show that Lemma 2.4 holds for $|D| = k$. If $\exists d \in (D \cap N(s)) \setminus F$, we create the path of length 1 $s \to d$. We remove d from D and add d into F. Then $|D| + |F| \leq n-1$ still holds. Hence, Lemma 2.4 holds by induction hypothesis. Therefore, we assume $(D \cap N(s)) \setminus F = \emptyset$. If $\exists d \in D \cap N(s) \cap F$, we select a node $d' \in N(d) \setminus (F \cup D \cup \{s\})$. We remove d from D and add d' into D, and hence $|D| + |F| \leq n-1$ still holds. Because $|F| + |D \setminus \{d\}| + |\{s\}| \leq n-1$, we have $|F \cup (D \setminus \{d\}) \cup \{s\}| < |N(d)| = n$. Hence, we can always find d'. Therefore, we assume $D \cap N(s) = \emptyset$. Now we reduce Q_n along a dimension δ into two subcubes Q^0 and Q^1 such that $D \cap Q^0 \neq \emptyset$ and $D \cap Q^1 \neq \emptyset$. Assume $s \in Q^0$ and $s_1 = s \oplus 2^\delta \in Q^1$.

Let $F_1 = ((F \cap Q^1) \setminus \{s_1\}) \cup \{s_i' \mid s_i' \in N(s_1) \cap Q^1, s_i(= s_i' \oplus 2^\delta) \in F\}$. We have $|F_1| \leq |F \cap Q^1| + |\{s_i' \mid s_i' \in N(s_1) \cap Q^1, s_i(= s_i' \oplus 2^\delta) \in F\}| \leq |(F \cap Q^1) \cup (F \cap Q^0)| \leq n-k-1$. Because $|D \cap Q^1| \leq |D| - 1 = k-1$ and $|F_1| \leq n-k-1 = (n-1)-(k-1)-1$, we can find by induction onto Q^1 disjoint paths of lengths at most $(n-1)+3 = n+2$ connecting s_1 to each destination node of Q^1 not including any internal node in F_1.

All of the disjoint paths $s_1 \to s_i' \rightsquigarrow d_j$ in Q^1 are then connected to s as follows. If $s_1 \in F$, then we replace edges $s_1 \to s_i'$ by the subpaths $s \to s_i (= s_i' \oplus 2^\delta) \to s_i'$. If $s_1 \notin F$, then one path is randomly selected and connected to s by one edge $s \to s_1$, and for all the other paths we replace edges $s_1 \to s_i'$ by the subpaths $s \to s_i \to s_i'$. Because for all of the disjoint paths $s_1 \to s_i' \rightsquigarrow d_j \in Q^1$ we have $s_i' \notin F_1$, it follows that $s_i \notin F$. Hence, the paths $s \to s_i \to s_i' \rightsquigarrow d_j$ do not include an internal faulty node. Because the paths found in Q^1 are disjoint, each s_i' is used by at most one path, hence each path is connected to s via a distinct $s_i \in Q^0$. Therefore, the paths $s \rightsquigarrow d_j$ stay disjoint; they are of length at most $(n+2)-1+2 = n+3$. Let $\mathcal{C}_1$ be the set of the paths $s \to s_i \to s_i' \rightsquigarrow d_j \in Q^1$ obtained.

Let $F_0 = (F \cap Q^0) \cup \{s_i \mid (s \to s_i \to s_i' \rightsquigarrow d_j) \in \mathcal{C}_1\}$. If $s_1 \in F$ we have $|F_0| \leq |F \cap Q^0| + |\{s_i \mid (s \to s_i \to s_i' \rightsquigarrow d_j) \in \mathcal{C}_1\}| \leq (n-k-2) + |D \cap Q^1|$. If $s_1 \notin F$, we have $|F_0| \leq |F \cap Q^0| + |\{s_i \mid (s \to s_i \to s_i' \rightsquigarrow d_j) \in \mathcal{C}_1\}| \leq (n-k-1) + (|D \cap Q^1| - 1)$. Because $|D \cap Q^0| \leq k - |D \cap Q^1|$ and $|F_0| \leq (n-k-2) + |D \cap Q^1|$, we can find by induction onto Q^0 disjoint paths of lengths at most $(n-1)+3 = n+2$ not including an internal faulty node, connecting s to each destination node of Q^0.

In addition, since the neighbours s_i of s used in paths connecting s to destination nodes in Q^1 are considered as faulty (gathered into F_0), and since these nodes s_i are the only internal nodes inside Q^0 of the paths connecting s to destination nodes in Q^1, the paths connecting s to destination nodes in Q^0 are disjoint with the paths connecting s to destination nodes in Q^1.

We express the time complexity of Cube-XFTN2S by induction on n and k. Let $T(k,n)$ be the time complexity of this algorithm in a Q_n with $|D| = k$.

In Case 1, if $X = \{x\}$, the reduction along $s \oplus x$ as well as the selection of the edge $s \to x$ are $O(1)$. If $d \in Q^1$, a path $x \rightsquigarrow d$ in Q^1 can be created in $O(n)$ by Cube-FTN2N. If $d \in Q^0$, the path $d \rightsquigarrow d' \in Q^1$ can be found in $O(n)$ by enumeration, and the path $x \rightsquigarrow d'$ can also be found in $O(n)$ by Cube-FTN2N. Now if $X = \emptyset$ and $d \in F$, a path $s \rightsquigarrow d$ of length 3 not including a faulty internal node can be found in $O(n)$ by enumeration. If $X = \emptyset$ and $d \notin F$, a fault-free path $s \rightsquigarrow d$ can be found in $O(n)$ by Cube-FTN2N. Therefore, in Case 1 we can find a path $s \rightsquigarrow d$ in $O(n)$.

In Case 2, it takes $O(n)$ to check if $D \cap N(s)$ is empty or not. If $\exists d \in D \cap N(s)$, it takes $O(n)$ to check if $d \in F$. If $d \notin F$, we can create a path of length 1 $s \to d$ in $O(1)$. Then the problem is reduced to Cube-XFTN2S(Q_n, s, $D \setminus \{d\}$, $F \cup \{d\}$, $X \setminus \{d\}$). From the induction hypothesis, it takes $T(k-1,n)$ time. If $d \in F$, a node d' can be found in $O(n)$. After the recursive call we select the edge $d \to d'$ in $O(1)$. Now we assume $D \cap N(s) = \emptyset$. We see that Q_n is reduced along a dimension δ in constant time since we assumed that two nodes can be compared in constant time. Creating F_1 requires $O(n)$ time; that is $F \cap Q^1$ is obtained by checking the dimension δ of each node of F, requiring $O(n)$ time. Also s_1 can be excluded from $F \cap Q^1$ in $O(n)$ time. Finally, by computing the Hamming distance from s we can find all the faulty neighbour nodes of s in $O(n)$ time. Hence, the set $\{s_i' \mid s_i' \in N(s_1) \cap Q^1, s_i(= s_i' \oplus 2^\delta) \in F\}$ can be obtained in $O(n)$ time. Connecting the paths inside Q^1 to s is

made in constant time for each path, hence requiring $O(k)$ time in total. Lastly, F_0 is created in $O(n)$ time.

Let $D^0 = D \cap Q^0$ and $D^1 = D \cap Q^1$. We have

$$\begin{aligned} T(1,n) &= O(n), \\ T(k,n) &= T(|D^0|, n-1) + T(|D^1|, n-1) + O(n) \\ &= O(kn) \end{aligned}$$

with $1 \leq |D^0|, |D^1| \leq k-1$ and $|D^0| + |D^1| = k$. If $X = \{x\}$, Case 1 ensures that one of the disjoint paths starts with the edge $s \to x$. Therefore, in a Q_n, Cube-XFTN2S generates k ($\leq n$) disjoint paths not including an internal faulty node, one starting with the edge $s \to x$ if $\exists x \in X$, of lengths at most $n+3$ in $O(kn)$ time complexity. □

Lemma 2.5 recalls that in an HHC_{2^m+m} we can distribute $m+1$ nodes to distinct subcubes with at most two edges.

LEMMA 2.5 ([7]). *In an* HHC_{2^m+m}*, given a set of* $m+1$ *nodes* $D = \{\boldsymbol{d}_1, \boldsymbol{d}_2, \ldots, \boldsymbol{d}_{m+1}\}$*, we can find* $m+1$ *disjoint paths* $\boldsymbol{d}_i \rightsquigarrow \boldsymbol{d}'_i (1 \leq i \leq m+1)$ *of lengths at most* 2 *in* $O(m^3)$ *time complexity such that the subcube of* $\boldsymbol{d}'_i$ *does not include any node in* $D \cup (D' \setminus \{\boldsymbol{d}'_i\})$ *where* $D' = \{\boldsymbol{d}'_1, \boldsymbol{d}'_2, \ldots, \boldsymbol{d}'_{m+1}\}$.

Proof. For an arbitrary $\boldsymbol{d}_i = (\sigma, p_i) \in D$, there are $m+1$ disjoint paths $P^{(i)}_1, \ldots, P^{(i)}_{m+1}$ of lengths at most 2 connecting $\boldsymbol{d}_i$ to $m+1$ distinct subcubes:

$$\begin{cases} \boldsymbol{d}_i = (\sigma, p_i) \to (\sigma \oplus 2^{p_i}, p_i) \in Q_m(\sigma \oplus 2^{p_i}) \\ \boldsymbol{d}_i = (\sigma, p_i) \to (\sigma, p_i \oplus 2^h) \to (\sigma \oplus 2^{p_i \oplus 2^h}, p_i \oplus 2^h) \\ \qquad\qquad \in Q_m(\sigma \oplus 2^{p_i \oplus 2^h}) \ (0 \leq h \leq m-1). \end{cases}$$

Let us show that, for any $\boldsymbol{d}_j$ ($1 \leq j \leq m+1, i \neq j$), the path $P^{(j)}_{w_j} : \boldsymbol{d}_j \to \boldsymbol{d}''_j \to \boldsymbol{d}'_j$ ($1 \leq w_j \leq m+1$) can block at most one of the $m+1$ paths $P^{(i)}_1, \ldots, P^{(i)}_{m+1}$.

Let us consider two paths $P^{(i)}_u : \boldsymbol{d}_i \to \boldsymbol{u}_1 \to \boldsymbol{u}_2 \in Q_m(\sigma_u)$ and $P^{(i)}_v : \boldsymbol{d}_i \to \boldsymbol{v}_1 \to \boldsymbol{v}_2 \in Q_m(\sigma_v)$ with $1 \leq u, v \leq m+1, u \neq v$ (Fig. 4). Because $\boldsymbol{u}_1$ and $\boldsymbol{v}_1$ are two distinct neighbours of the same node $\boldsymbol{d}_i$, we have $H(\boldsymbol{u}_1, \boldsymbol{v}_1) = 2$.

First assume that $\boldsymbol{d}_j \in Q_m(\sigma)$. Then $\boldsymbol{d}''_j \in Q_m(\sigma)$ and $\boldsymbol{d}'_j \notin Q_m(\sigma)$ hold. Hence, $\boldsymbol{u}_1$ and $\boldsymbol{v}_1$ cannot be both on $P^{(j)}_{w_j}$ because $H(\boldsymbol{u}_1, \boldsymbol{v}_1) = 2$. In addition, if $\boldsymbol{d}_j = \boldsymbol{u}_1$, then $\boldsymbol{d}'_j \notin Q_m(\sigma_v)$ holds because there is only one external edge $\boldsymbol{v}_1 \to \boldsymbol{v}_2$ between $Q_m(\sigma)$ and $Q_m(\sigma_v)$, and because $\boldsymbol{d}''_j \neq \boldsymbol{v}_1$. Therefore, $P^{(j)}_{w_j}$ cannot block both $P^{(i)}_u$ and $P^{(i)}_v$ if $\boldsymbol{d}_j \in Q_m(\sigma)$.

Now assume $\boldsymbol{d}_j \notin Q_m(\sigma)$ and $\boldsymbol{d}_j \in Q_m(\sigma_u)$, we recall that $H(\sigma, \sigma_u) = H(\sigma, \sigma_v) = 1$, and hence $H(\sigma_u, \sigma_v) = 2$ since $\boldsymbol{u}_1 \neq \boldsymbol{v}_1$. Therefore, there is no external edge between $Q_m(\sigma_u)$ and $Q_m(\sigma_v)$, and hence $\boldsymbol{d}'_j \notin Q_m(\sigma_v)$ since $P^{(j)}_{w_j}$ has only one external edge. As a result $P^{(j)}_{w_j}$ cannot block $P^{(i)}_u$ and $P^{(i)}_v$ at the same time if $\boldsymbol{d}_j \in Q_m(\sigma_u)$.

From this discussion, we can deduce that each path $P^{(j)}_{w_j}$ ($1 \leq w_j \leq m+1$) can block at most one of the $m+1$ paths $P^{(i)}_1, \ldots, P^{(i)}_{m+1}$. Therefore, at least one ($(m+1) - m = 1$) path $P^{(i)}_{w_i}$ ($1 \leq w_i \leq m+1$) remains to connect $\boldsymbol{d}_i$ to a node $\boldsymbol{d}'_i$. We see that $P^{(i)}_{w_i}$ can be found in $O(m^2)$ time complexity by checking all the $m+1$ paths $P^{(i)}_1, \ldots, P^{(i)}_{m+1}$ of lengths at most 2 for $\boldsymbol{d}_i$. Hence, we can connect all nodes $\boldsymbol{d}_i \in D$ to nodes $\boldsymbol{d}'_i$ with disjoint paths of lengths at most 2 in $O(m^3)$ time complexity. □

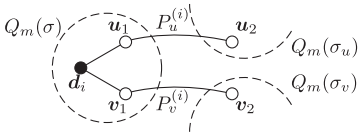

FIGURE 4. Two candidate paths for distribution of $\boldsymbol{d}_i$.

3. HHC NODE-TO-SET DISJOINT-PATH ROUTING ALGORITHM

We describe in this section the HHC node-to-set disjoint-path routing algorithm HHC-iN2S. The main idea of the algorithm is to reduce the node-to-set disjoint-path routing problem in an HHC to the node-to-set disjoint-path routing problem in a hypercube by mapping each subcube of an HHC_{2^m+m} to a single node of a 2^m-dimensional hypercube Q_{2^m}. A path inside an HHC_{2^m+m} is called an HHC-level path, and a path inside a Q_{2^m} is called a cube-level path, made of cube-level nodes, that is subcube IDs of an HHC_{2^m+m}.

Before describing HHC-iN2S we introduce an algorithm CONV which takes as input one cube-level path $P : \sigma_0 \rightsquigarrow \sigma_n$ and two processors IDs $\pi_{\mathrm{beg}}, \pi_{\mathrm{end}}$, and generates an HHC-level path $(\sigma_0, \pi_{\mathrm{beg}}) \rightsquigarrow (\sigma_n, \pi_{\mathrm{end}})$ according to P. For a path $P : \sigma_0 \to \sigma_1 \to \cdots \to \sigma_n$, let π_j ($1 \leq j \leq n$) be an m-bit sequence that satisfies $\sigma_{j-1} \oplus 2^{\pi_j} = \sigma_j$. We construct the corresponding HHC-level path $(\sigma_0, \pi_{\mathrm{beg}}) \rightsquigarrow (\sigma_0, \pi_1) \to (\sigma_1, \pi_1) = (\sigma_0 \oplus 2^{\pi_1}, \pi_1) \rightsquigarrow (\sigma_1, \pi_2) \to \cdots \to (\sigma_{n-1}, \pi_{n-1}) \rightsquigarrow (\sigma_{n-1}, \pi_n) \to (\sigma_n, \pi_n) \rightsquigarrow (\sigma_n, \pi_{end})$, using an SPR algorithm inside every subcube to connect each (σ_j, π_j) to (σ_j, π_{j+1}). The pseudocode is given in Algorithm 1.

REMARK 2. One should note that there is no difference regarding the length of the cube-level paths returned by SPR and SPR-Gray. In case SPR is used, at most m internal edges are required inside each subcube when converting cube-level paths back to HHC-level paths, that is in the worst case $m \times O(2^m)$ internal edges, where $O(2^m)$ represents the cube-level paths' maximum length. On the other hand, SPR-Gray implies at most $2^m - 1$ internal edges in total, visiting the nodes in a Q_m corresponding to the bit positions to be flipped following the occurrence order in the Hamiltonian cycle specified by the Gray code.

In an HHC_{2^m+m}, for any cube-level path P of length l in Q_{2^m}, CONV applies an SPR algorithm inside each visited subcube, thus having a time complexity of $O(lm)$. Now for any cube-level shortest-path P in Q_{2^m} whose nodes are traversed in Gray code

order, the number of internal edges generated by SPR in CONV will be at most 2^m. Since SPR has a linear time complexity in the number of edges it generates, CONV will be of $O(2^m)$ total time complexity.

Algorithm 1 CONV($P = \sigma_0 \to \sigma_1 \to \ldots \to \sigma_n, \pi_{beg}, \pi_{end}$).

Input: A cube-level path $P = \sigma_0 \rightsquigarrow \sigma_n$ and two processors IDs to specify the HHC nodes (σ_0, π_{beg}) and (σ_n, π_{end}).
Output: The HHC-level path $(\sigma_0, \pi_{beg}) \rightsquigarrow (\sigma_n, \pi_{end})$ corresponding to P.
  if $L(P) = 0$ **then**
    $\pi_0(= \pi_{beg}) \to \pi_1 \to \ldots \to \pi_\lambda(= \pi_{end}) :=$ SPR(π_{beg}, π_{end});
    return $(\sigma_0, \pi_0) \to (\sigma_0, \pi_1) \to \ldots \to (\sigma_0, \pi_\lambda)$
  else
    $\pi_{next} := log_2(\sigma_0 \oplus \sigma_1)$;
    $\pi_0(= \pi_{beg}) \to \pi_1 \to \ldots \to \pi_{\lambda'}(= \pi_{next}) :=$ SPR(π_{beg}, π_{next});
    $P' :=$ CONV($\sigma_1 \rightsquigarrow \sigma_n, \pi_{next}, \pi_{end}$);
    return $(\sigma_0, \pi_0) \to (\sigma_0, \pi_1) \to \ldots \to (\sigma_0, \pi_{\lambda'}) \to P'$
  end if

Let $\boldsymbol{s} = (s_0, p_0)$ be the source node and $D = \{\boldsymbol{d}_1 = (s_1, p_1), \ldots, \boldsymbol{d}_k = (s_k, p_k)\}$, $k \leq m+1$ be the set of destination nodes.

Step 0. If $|D \cap Q_m(s_0)| \leq k-2$, then go to Step 1. Otherwise, we assume $\{\boldsymbol{d}_1, \ldots, \boldsymbol{d}_{k-1}\} \subset Q_m(s_0)$ without loss of generality.

If $\boldsymbol{d}_k \in Q_m(s_0)$ and $k < m+1$, we connect $\boldsymbol{s}$ to D inside $Q_m(s_0)$ with Cube-N2S. If $\boldsymbol{d}_k \in Q_m(s_0)$ and $k = m+1$, we connect $\boldsymbol{s}$ to $\{\boldsymbol{d}_1, \ldots, \boldsymbol{d}_{k-1}\}$ inside $Q_m(s_0)$ with Cube-N2S. If $\boldsymbol{d}_k$ is included in one of the $k-1$ paths $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}_i$ $(1 \leq i \leq k-1)$ returned by Cube-N2S, say $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}_j$, then swap the indices of $\boldsymbol{d}_k$ and $\boldsymbol{d}_j$, and discard the subpath $\boldsymbol{d}_k \rightsquigarrow \boldsymbol{d}_j$. Finally, $\boldsymbol{s}$ is connected to $\boldsymbol{d}_k$ with the path $\boldsymbol{s} = (s_0, p_0) \to (s_0 \oplus 2^{p_0}, p_0) \rightsquigarrow (s_0 \oplus 2^{p_0}, p_k) \to (s_0 \oplus 2^{p_0} \oplus 2^{p_k}, p_k) \rightsquigarrow (s_0 \oplus 2^{p_0} \oplus 2^{p_k}, p_0) \to (s_0 \oplus 2^{p_k}, p_0) \rightsquigarrow (s_0 \oplus 2^{p_k}, p_k) \to (s_0, p_k) = \boldsymbol{d}_k$, where $\rightsquigarrow$ represents an SPR. Otherwise, that is $\boldsymbol{d}_k \notin Q_m(s_0)$, we first connect $\boldsymbol{s}$ to $\{\boldsymbol{d}_1, \ldots, \boldsymbol{d}_{k-1}\}$ inside $Q_m(s_0)$ with Cube-N2S. Then we find a cube-level path $P : s_0 \oplus 2^{p_0} \rightsquigarrow s_k \oplus 2^{p_k}$ by using SPR-Gray so that the path does not start with the edge $s_0 \oplus 2^{p_0} \to s_0$. If $s_k \in P$, then we discard the subpath $s_k \rightsquigarrow s_k \oplus 2^{p_k}$ and we apply CONV to P and the processor IDs p_0, p_k. Otherwise we apply CONV to P and the processor IDs p_0, p_k, and we extend the HHC-level path obtained by one external edge $(s_k \oplus 2^{p_k}, p_k) \to \boldsymbol{d}_k$ (Fig. 5).

All paths are now constructed, and the algorithm terminates.

Step 1. We can now assume without loss of generality that $Q_m(s_0) \cap D = \{\boldsymbol{d}_1, \boldsymbol{d}_2, \ldots, \boldsymbol{d}_r\}$, $r < k-1$. Let Z_1 be the set of subcube IDs (s_0 excluded) whose corresponding subcubes contain more than one destination node. Let Z_2 be the set of subcube IDs σ with $H(s_0, \sigma) = 1$ and $Q_m(\sigma)$ linked to $Q_m(s_0)$ with an external edge whose end node in $Q_m(s_0)$ is a destination node. We exclude the subcube IDs of Z_1 from Z_2. Formally

$$Z_1 = \{\sigma \mid |Q_m(\sigma) \cap D| \geq 2\} \setminus \{s_0\},$$
$$Z_2 = \{s_0 \oplus 2^{p_1}, \ldots, s_0 \oplus 2^{p_r}\} \setminus Z_1.$$

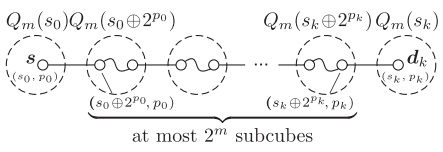

FIGURE 5. A path $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}_k$ generated if $|D \cap Q_m(s_0)| = k-1$ and $\boldsymbol{d}_k \notin Q_m(s_0)$.

For each destination node $\boldsymbol{d}_i = (s_i, p_i)$ with $s_i \in Z_1 \cup Z_2$, we connect $\boldsymbol{d}_i$ to a node $\boldsymbol{d}'_i = (s'_i, p'_i)$ with at most two edges such that the following statements hold:

(1) $\forall \boldsymbol{d}_i = (s_i, p_i)$ with $s_i \in Z_1 \cup Z_2$, $s'_i \notin Z_1 \cup Z_2$ for the corresponding $\boldsymbol{d}'_i = (s'_i, p'_i)$
(each $\boldsymbol{d}_i$ whose subcube ID is in $Z_1 \cup Z_2$ is connected to a node $\boldsymbol{d}'_i$ whose subcube ID is not in $Z_1 \cup Z_2$);
(2) $\forall \boldsymbol{d}_i = (s_i, p_i)$ with $s_i \in Z_1 \cup Z_2$, $D \cap Q_m(s'_i) = \emptyset$ for the corresponding $\boldsymbol{d}'_i = (s'_i, p'_i)$
(each $\boldsymbol{d}_i$ whose subcube ID is in $Z_1 \cup Z_2$ is connected to a node $\boldsymbol{d}'_i$ whose subcube contains no destination node);
(3) $\forall \boldsymbol{d}_i = (s_i, p_i)$, $\boldsymbol{d}_j = (s_j, p_j)$ with $s_i, s_j \in Z_1 \cup Z_2$ and $i \neq j$, $s'_i \neq s'_j$ for the corresponding $\boldsymbol{d}'_i = (s'_i, p'_i)$ and $\boldsymbol{d}'_j = (s'_j, p'_j)$
(each $\boldsymbol{d}_i$ whose subcube ID is in $Z_1 \cup Z_2$ is connected to a node $\boldsymbol{d}'_i$ into a distinct subcube).

Each such node $\boldsymbol{d}'_i$ is called the distributed destination node for $\boldsymbol{d}_i$.

Finally, let Z_3 be the set of subcube IDs whose corresponding subcubes contain either one distributed destination node $\boldsymbol{d}'_i$ or exactly one destination node $\boldsymbol{d}_i$ $(r < i \leq k)$ with $s_i \notin Z_2$ (s_0 is excluded from Z_3). Formally

$$Z_3 = (\{s'_i \mid \forall s_i \in Z_1 \cup Z_2\} \cup \{s_i \mid \forall s_i \notin Z_1 \cup Z_2\}) \setminus \{s_0\}.$$

Step 2. Let X be $\{s_0 \oplus 2^{p_0}\}$ if $k = m+1$ and $s_0 \oplus 2^{p_0} \notin Z_1$. Otherwise $X = \emptyset$. We apply Cube-XFTN2S onto a 2^m-dimensional hypercube Q_{2^m}. The source node for Cube-XFTN2S is s_0, Z_3 is the set of destination nodes, $Z_1 \cup Z_2$ is the set of faulty nodes and X is the restriction set. Let P_i be each of the cube-level paths generated by Cube-XFTN2S where P_i is of the form $s_0 \rightsquigarrow s_i$ or $s_0 \rightsquigarrow s'_i$. If $\exists \boldsymbol{d}'_i \in Q_m(s_0)$, then there is no path for s'_i generated since s_0 $(= s'_i)$ is excluded from Z_3. So let $P_i : s_0$ $(= s'_i)$ of length 0.

Step 3. We convert the paths $P_{r+1}, \ldots, P_k$ obtained in Step 2 back to HHC-level paths using CONV. We recall that a distributed destination node $\boldsymbol{d}'_i$ is connected to $\boldsymbol{d}_i$ with a path of length 2, denoted by $\boldsymbol{d}'_i \to \boldsymbol{d}''_i \to \boldsymbol{d}_i$, or of length 1, denoted by $\boldsymbol{d}'_i \to \boldsymbol{d}_i$. First we handle routing inside $Q_m(s_0)$ separately.

For each path P_i $(r+1 \leq i \leq k)$, if it is of the form $s_0 \rightsquigarrow s'_i$, then it is extended with one edge to $P_i : s_0 \rightsquigarrow s'_i \to s_i$.

If $k = m+1$, $s_0 \oplus 2^{p_0} \in Z_1$ and $\nexists \boldsymbol{d}'_i \in Q_m(s_0)$ with the corresponding $\boldsymbol{d}_i$ inside $Q_m(s_0 \oplus 2^{p_0})$, then let $\boldsymbol{d}_x$ be the destination node of $Q_m(s_0 \oplus 2^{p_0})$ such that $\boldsymbol{d}_x$ or $\boldsymbol{d}''_x$ (if any) is the closest of all the $\boldsymbol{d}_j, \boldsymbol{d}''_j \in Q_m(s_0 \oplus 2^{p_0})$ $(x \neq j)$ nodes to the node $(s_0 \oplus 2^{p_0}, p_0)$. If there exist $\boldsymbol{d}_{j_1}$ and $\boldsymbol{d}''_{j_2}$ in $Q_m(s_0 \oplus 2^{p_0})$ and

Algorithm 2 HHC-iN2S(HHC_{2^m+m}, s, D).

Input: An HHC_{2^m+m}, a source node $s = (s_0, p_0)$ and a set of k ($\leq m+1$) destination nodes $D = \{d_1 = (s_1, p_1), \ldots, d_k = (s_k, p_k)\}$.
Output: k disjoint paths $s \rightsquigarrow d_i$ ($1 \leq i \leq k$).

if $|D \cap Q_m(s_0)| \geq k-1$ **then** /* *Step 0* */
  /* *Assume* $\{d_1, \ldots, d_{k-1}\} \subset D \cap Q_m(s_0)$ */
  if $d_k \in Q_m(s_0)$ and $k < m+1$ **then**
    $\mathcal{S}_0 :=$ Cube-N2S($Q_m(s_0)$, p_0, $\{p_1, \ldots, p_k\}$);
    $\mathcal{H} := \{(s_0, p_0) \rightsquigarrow (s_0, p_i) \mid \forall (p_0 \rightsquigarrow p_i) \in \mathcal{S}_0, 1 \leq i \leq k\}$
  else if $d_k \in Q_m(s_0)$ and $k = m+1$ **then**
    $\mathcal{S}_0 :=$ Cube-N2S($Q_m(s_0)$, p_0, $\{p_1, \ldots, p_{k-1}\}$);
    $\mathcal{H} := \{(s_0, p_0) \rightsquigarrow (s_0, p_i) \mid \forall (p_0 \rightsquigarrow p_i) \in \mathcal{S}_0, 1 \leq i \leq k-1\}$;
    /* *If* $d_k \in \mathcal{H}$, *additional processing required, but omitted.* */
    $\mathcal{H} := \mathcal{H} \cup \{s \rightarrow (s_0 \oplus 2^{p_0}, p_0) \rightsquigarrow (s_0 \oplus 2^{p_0}, p_k) \rightarrow (s_0 \oplus 2^{p_0} \oplus 2^{p_k}, p_k) \rightsquigarrow (s_0 \oplus 2^{p_0} \oplus 2^{p_k}, p_0) \rightarrow (s_0 \oplus 2^{p_k}, p_0) \rightsquigarrow (s_0 \oplus 2^{p_k}, p_k) \rightarrow d_k\}$
  else /* $d_k \notin Q_m(s_0)$ */
    $\mathcal{S}_0 :=$ Cube-N2S($Q_m(s_0)$, p_0, $\{p_1, \ldots, p_{k-1}\}$);
    $\mathcal{H} := \{(s_0, p_0) \rightsquigarrow (s_0, p_i) \mid \forall (p_0 \rightsquigarrow p_i) \in \mathcal{S}_0, 1 \leq i \leq k-1\}$;
    $P :=$ SPR-Gray($s_0 \oplus 2^{p_0}$, $s_k \oplus 2^{p_k}$); /* *such that* $s_0 \notin P$ */
    if $s_k \in P$ **then**
      Discard the subpath $s_k \rightsquigarrow s_k \oplus 2^{p_k}$ from P;
      $\mathcal{H} := \mathcal{H} \cup \{s \rightarrow \text{CONV}(P, p_0, p_k)\}$
    else
      $\mathcal{H} := \mathcal{H} \cup \{s \rightarrow \text{CONV}(P, p_0, p_k) \rightarrow d_k\}$
    end if
  end if
else
  /* *Assume* $\{d_1, \ldots, d_r\} = D \cap Q_m(s_0)$ *with* $r < k-1$ */
  /* *Step 1: Destination nodes distribution* */
  $Z_1 = \{\sigma \mid |Q_m(\sigma) \cap D| \geq 2\} \setminus \{s_0\}$;
  $Z_2 = \{s_0 \oplus 2^{p_1}, \ldots, s_0 \oplus 2^{p_r}\} \setminus Z_1$;
  $\forall d_i = (s_i, p_i)$ with $s_i \in Z_1 \cup Z_2$, connect d_i to a node $d'_i = (s'_i, p'_i)$ in at most two edges such that $s'_i \notin Z_1 \cup Z_2$, $D \cap Q_m(s'_i) = \emptyset$ and $\forall d_j = (s_j, p_j), s_j \in Z_1 \cup Z_2, i \neq j$ we have $s'_i \neq s'_j$;
  $Z_3 = (\{s'_i \mid \forall s_i \in Z_1 \cup Z_2\} \cup \{s_i \mid \forall s_i \notin Z_1 \cup Z_2\}) \setminus \{s_0\}$;
  /* *Step 2: Routing by Cube-XFTN2S* */
  if $k = m+1$ and $s_0 \oplus 2^{p_0} \notin Z_1$ **then** $X := \{s_0 \oplus 2^{p_0}\}$ **else** $X := \emptyset$ **end if**
  $\mathcal{C}_1 :=$ Cube-XFTN2S(Q_{2^m}, s_0, Z_3, $Z_1 \cup Z_2$, X);
  for all $P \in \mathcal{C}_1$ /* *Assume* $P : s_0 \rightsquigarrow s_i$ *or* $P : s_0 \rightsquigarrow s'_i$ */ **do** $P_i := P$ **end for**
  if $\exists d'_i \in Q_m(s_0)$ **then** $P_i := s_0$ **end if** /* *Path of length zero* */
  /* *Step 3: Path conversion* */
  for all $P_i : s_0 \rightsquigarrow s'_i$ **do** $P_i := P_i \rightarrow s_i$ **end for**
  if $k = m+1$ and $s_0 \oplus 2^{p_0} \in Z_1$ and $\nexists d'_i \in Q_m(s_0)$ with $d_i \in Q_m(s_0 \oplus 2^{p_0})$ **then**
    /* *Assume* $P_x : s_0 \rightsquigarrow s_x$ *is the path such that* d_x *(or* d''_x *if any) is the closest* d_j *or* d''_j *to* $(s_0 \oplus 2^{p_0}, p_0)$ *inside* $Q_m(s_0 \oplus 2^{p_0})$ */
    $P_x := s_0 \rightarrow s_0 \oplus 2^{p_0}$
  end if
  $\mathcal{S}_0 :=$ Cube-N2S($Q_m(s_0)$, p_0, $\{p_1, \ldots, p_r\} \cup \{\pi_i \mid (s_0 \rightarrow s_0 \oplus 2^{\pi_i}) \in P_i, r+1 \leq i \leq k\} \setminus \{p_0\}$);
  $\mathcal{H}_0 := \{(s_0, p_0) \rightsquigarrow (s_0, \pi) \mid \forall (p_0 \rightsquigarrow \pi) \in \mathcal{S}_0\}$;
  $\mathcal{H}_1 := \emptyset$;
  for all $P_i : s_0 \rightarrow s_0 \oplus 2^{\pi_i} \rightsquigarrow s_i$ **do** /* *A special treatment for* P_x *may be required, but omitted.* */
    $\mathcal{H}_1 := \mathcal{H}_1 \cup \{\text{CONV}(P_i, \pi_i, p_i)\}$
  end for
  $\mathcal{H} :=$ { join the paths of $\mathcal{H}_0$ to the corresponding path of $\mathcal{H}_1$ }
end if
return $\mathcal{H}$

they are both closest to the node $(s_0 \oplus 2^{p_0}, p_0)$, we always select d_{j_1} as d_x. Let $P_x : s_0 \rightsquigarrow s_x$ be replaced by $P_x : s_0 \rightarrow s_0 \oplus 2^{p_0}$.

We apply Cube-N2S to disjointly connect s to the set $(D \cap Q_m(s_0)) \cup \{(s_0, \pi_i) \mid (s_0 \rightarrow s_0 \oplus 2^{\pi_i}) \in P_i, r+1 \leq i \leq k\} \setminus \{s\}$ of k or $k-1$ nodes depending on the existence of a path in P_i's starting with the edge $s_0 \rightarrow s_0 \oplus 2^{p_0}$.

If $k = m+1$, $s_0 \oplus 2^{p_0} \in Z_1$, $\nexists d'_i \in Q_m(s_0)$ with the corresponding d_i inside $Q_m(s_0 \oplus 2^{p_0})$ and $d''_x = (s_0 \oplus 2^{p_0}, p''_x)$ is closest to $(s_0 \oplus 2^{p_0}, p_0)$, then for P_x we apply $\text{CONV}(P_x, p_0, p''_x)$ and extend the generated path with the edge $d''_x \rightarrow d_x$. For the other paths $P_i : s_0 \rightarrow s_0 \oplus 2^{\pi_i} \rightsquigarrow s_i$ ($i \neq x$), we apply $\text{CONV}(P_i, \pi_i, p_i)$. Otherwise we apply $\text{CONV}(P_i, \pi_i, p_i)$ for all the paths $P_i : s_0 \rightarrow s_0 \oplus 2^{\pi_i} \rightsquigarrow s_i$ ($r+1 \leq i \leq k$).

All paths are now constructed, and the algorithm terminates. The pseudocode is given in Algorithm 2.

4. CORRECTNESS AND COMPLEXITIES

We shall prove in this section the correctness of HHC-iN2S, evaluate the maximum path length and assess its time complexity.

In the case $|D \cap Q_m(s_0)| \geq k-1$, the routing problem is solved using Cube-N2S inside $Q_m(s_0)$, generating paths of lengths at most $m+1$. However, an additional path $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}_k$ can be generated outside $Q_m(s_0)$. The maximum path length is attained in the case $\boldsymbol{d}_k \notin Q_m(s_0)$ and it is $(1+2^m)+2^m = 2^{m+1}+1$, that is 2^m+1 external edges and 2^m internal edges to cycle through an m-bit Gray code. The corresponding cube-level path is created in $O(2^m)$ time complexity as well as the application of the CONV algorithm since we have used SPR-Gray, and hence that path is constructed in $O(2^m)$ total time complexity. It is disjoint with the other paths inside $Q_m(s_0)$ since all its internal nodes are outside $Q_m(s_0)$. If $\boldsymbol{d}_k \in Q_m(s_0)$, checking if $\boldsymbol{d}_k$ is included in one of the paths generated by Cube-N2S inside $Q_m(s_0)$ requires $O(m^2)$ time complexity. From this discussion, we can deduce that the time complexity of Step 0 is $O(2^m)$.

To apply in Step 2 Cube-XFTN2S, we need that in Q_{2^m} the number of destination nodes plus the number of faulty nodes be less than or equal to 2^m-1. We assumed $|D \cap Q_m(s_0)| = r, r < k-1$, and hence the number of destination nodes for Cube-XFTN2S is $|Z_3| = k-r$. The number of faulty nodes is given by $|Z_1|+|Z_2| \leq \lfloor (k-r)/2 \rfloor + r$. We have $|Z_3|+|Z_1|+|Z_2| \leq (k-r)+\lfloor (k-r)/2 \rfloor + r = r+\lfloor 3(k-r)/2 \rfloor$. It is easy to check that $r+\lfloor 3(k-r)/2 \rfloor \leq 2^m-1$ holds for $m \geq 3$ (note that $|Z_3|+|Z_1|+|Z_2|$ is maximized for $r=0$). If $m=1$, the corresponding HHC is a cycle, and hence it is trivial to solve the node-to-set disjoint-path routing problem ($|D| \leq 2$). If $m=2$, we solve the routing problem as in [7].

By Lemmas 2.2, 2.4 and 2.5 we understand that the HHC-level paths connecting $\boldsymbol{s}$ to D are disjoint.

Now let us focus on the maximum path length generated by HHC-iN2S (Fig. 6). We recall that when applying Cube-XFTN2S in Q_{2^m}, Z_3 is the set of destination nodes and $Z_1 \cup Z_2$ is the set of faulty nodes. Cube-XFTN2S returns paths that include at most 2^m+4 nodes, with each path having a subsequence of at least $(2^m+4) - (|Z_3| + \lceil \log_2 |Z_1 \cup Z_2| \rceil + 3) = 2^m - |Z_3| - \lceil \log_2 |Z_1 \cup Z_2| \rceil + 1 \ (= \nu)$ nodes traversed in Gray code order (Remark 1), in $O(|Z_3|2^m)$ time complexity. Routing inside $Q_m(s_0)$ requires at most $m+1$ internal edges (Cube-N2S) plus one external edge. For the at most 2^m+3 remaining subcubes, at least ν of them are traversed in Gray code order. This means that $\nu-1$ subcube routings generate at most 2^m-1 internal edges (Remark 2). Therefore, at most $(2^m+3)-(\nu-1) \ (=\mu)$ subcubes will need at most m internal edges for subcube routing. In the final subcube among ν subcubes that are traversed in Gray code order, routing requires at most m internal edges. Hence, it is also counted in μ. Finally, we count the internal edge $\boldsymbol{d}_i'' \rightarrow \boldsymbol{d}_i$. In addition, Cube-XFTN2S generates a cube-level path $s_0 \rightsquigarrow s_i'$ of length at most 2^m+3, which corresponds to at most 2^m+3 external edges. Also, taking the external edge $\boldsymbol{d}_i' \rightarrow \boldsymbol{d}_i''$ (or $s_i' \rightarrow s_i$) into consideration, there are at most 2^m+4 external edges. Hence, we obtain the following maximum path length:

$$
\begin{aligned}
&(m+1)+\mu m+(2^m-1)+1+(2^m+4)\\
&\quad \leq 2^{m+1}+m((k-r)+\lceil \log_2(\lfloor (k-r)/2 \rfloor + r)\rceil + 4)+5\\
&\quad \leq 2^{m+1}+m(k+\lceil \log_2 \lfloor k/2 \rfloor \rceil + 4)+5\\
&\qquad \text{(maximized for } r=0)\\
&\quad \leq 2^{m+1}+m((m+1)+\lceil \log_2 \lfloor (m+1)/2 \rfloor \rceil + 4)+5\\
&\qquad \text{(maximized for } k=m+1)\\
&\quad = 2^{m+1}+m^2+m(\lceil \log_2 m \rceil + 4)+5.
\end{aligned}
$$

Regarding the time complexity of HHC-iN2S, Step 1 may have in the worst case to distribute all the destination nodes $\boldsymbol{d}_i$ to the corresponding nodes $\boldsymbol{d}_i'$. By Lemma 2.5, these paths can be found in $O(m^3)$ time. By Lemma 2.4, Cube-XFTN2S(Q_n, s, D, F, X) requires $O(|D|n)$ time complexity. Hence, Step 2 has a time complexity of $O(k2^m)$ since it includes Cube-XFTN2S(Q_{2^m}, s_0, Z_3, $Z_1 \cup Z_2$, X) where $|Z_3| \leq k$. Step 3 converts each path of Step 2 to HHC-level paths by applying CONV. Since CONV takes $O(lm)$ time complexity, where l is the length of the cube-level path and m the dimension of subcubes, Step 3 requires $O(m2^m)$ time complexity to convert one path whose length is $O(2^m)$. Therefore, Step 3 has a total time complexity of $O(km2^m)$, which is thus the dominant time complexity of the algorithm.

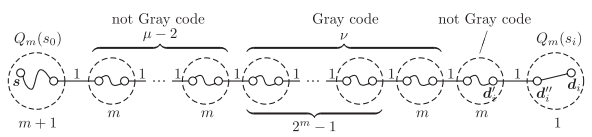

FIGURE 6. A path of maximum length.

REMARK 3. If instead of SPR-Gray we would have used SPR, each subcube routing would have used at most m internal edges. Hence, the maximum path length would have been

$$
\begin{aligned}
&(m+1)+m(2^m+3)+1+(2^m+4)\\
&\quad = m2^m+2^m+4m+6.
\end{aligned}
$$

HHC-N2S proposed by Bossard *et al.* [7] obtained a slightly shorter maximum path length of $m2^m+2^m+2m+2$. Consequently, the improvement proposed in HHC-iN2S resulting in the maximum path length $2^{m+1}+m^2+m(\lceil \log_2 m \rceil + 4)+5$ has a definite impact, significantly reducing the maximum path length from $O(m2^m)$ to $O(2^m)$.

REMARK 4. One should note that in the case $m=1$, the node-to-set disjoint-path routing problem is solved by enumeration, producing paths of lengths at most 6. In the case $m=2$, the problem is solved by HHC-N2S resulting in a maximum path length of 20. These maximum path lengths are shorter than the

TABLE 1. A routing example inside an HHC_{11}.

Q_8 path	HHC_{11} path	Q_8 path	HHC_{11} path	Q_8 path	HHC_{11} path	Q_8 path	HHC_{11} path
00000000	(00000000, 000)	00000000	(00000000, 000)	00000000	(00000000, 000)	00000000	(00000000, 000)
00000001	(00000001, 000)		(00000000, 100)		(00000000, 010)		(00000000, 001)
	(00000001, 010)	00010000	(00010000, 100)	00000100	(00000100, 010)		(00000000, 011)
	(00000001, 110)		(00010000, 110)		(00000100, 000)		(00000000, 111)
01000001	(01000001, 110)	01010000	(01010000, 110)	00000101	(00000101, 000)	10000000	(10000000, 111)
	(01000001, 100) $\boldsymbol{d}'_1$		(01010000, 111)		(00000101, 001)		(10000000, 110)
01010001	(01010001, 100) $\boldsymbol{d}_1$	11010000	(11010000, 111)		(00000101, 011)		(10000000, 100)
			(11010000, 110)	00001101	(00001101, 011) $\boldsymbol{d}_3$		(10000000, 000)
			(11010000, 100)			10000001	(10000001, 000)
			(11010000, 000)				(10000001, 010)
		11010001	(11010001, 000)				(10000001, 110)
			(11010001, 001)			11000001	(11000001, 110)
			(11010001, 011)				(11000001, 100)
			(11010001, 111) $\boldsymbol{d}'_2$				(11000001, 000) $\boldsymbol{d}_4$
		01010001	(01010001, 111) $\boldsymbol{d}_2$				
P_1		P_2		P_3		P_4	

maximum path lengths computed in the general case with $m = 1$ and $m = 2$:

$$2^{m+1} + m^2 + m(\lceil \log_2 m \rceil + 4) + 5 = \begin{cases} 14 & (m = 1), \\ 27 & (m = 2). \end{cases}$$

Therefore, the following theorem holds for any m.

THEOREM 4.1. *Inside an* HHC_{2^m+m}, *given a node* $\boldsymbol{s}$ *and a set of* $k(\leq m + 1)$ *nodes* $D = \{\boldsymbol{d}_1, \ldots, \boldsymbol{d}_k\}$, *we can find* k *disjoint paths* $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}_i (1 \leq i \leq k)$ *of lengths at most* $2^{m+1} + m^2 + m(\lceil \log_2 m \rceil + 4) + 5$ *in* $O(km2^m)$ *time complexity.*

As an example, we solve a node-to-set disjoint-path routing problem in an $HHC_{11}(m = 3)$ using HHC-iN2S. The source node is $\boldsymbol{s} = (00000000, 000)$ and the destination nodes are $D = \{\boldsymbol{d}_1 = (01010001, 100), \boldsymbol{d}_2 = (01010001, 111), \boldsymbol{d}_3 = (00001101, 011), \boldsymbol{d}_4 = (11000001, 000)\}$. We note that $\boldsymbol{d}_1$ and $\boldsymbol{d}_2$ are inside the same subcube and will thus be first distributed to the nodes $\boldsymbol{d}'_1$ and $\boldsymbol{d}'_2$, respectively. The four generated paths are fully described in Table 1, where P_i $(1 \leq i \leq 4)$ represents a cube-level path.

5. EMPIRICAL EVALUATION

In this section, we conduct two computer experiments to evaluate HHC-iN2S. We implemented HHC-iN2S using the Scheme functional programming language under the development environment DrScheme 4.2.5 [10] running on a 32-bit system.

We evaluated first the time needed for the algorithm to solve node-to-set disjoint-path routing problems, and then we focused on the maximum length of the generated paths. Concretely for both experiments we solved 10 000 routing problems inside an HHC_{2^m+m} for each value of m ranging from 3 to 9. The number of destination nodes is always set to $m + 1$. All nodes, including the source, are randomly generated.

Figure 7 represents for each m the average time needed to solve one routing problem. The theoretical time complexity is also represented. Figures 8 and 9 represent, respectively, for each m the average and the maximum maximal path lengths generated; that is, we compute, respectively, the average and the maximum of the 10 000 maximum path lengths obtained for the current value of m. The corresponding average and maximum maximal path lengths of the algorithm HHC-N2S are also represented to illustrate the gain by this improved algorithm

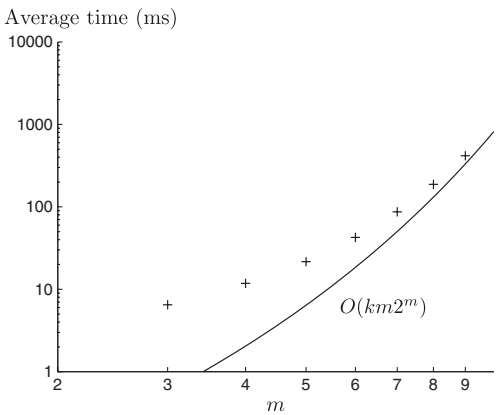

FIGURE 7. Average execution time ($k = m + 1$).

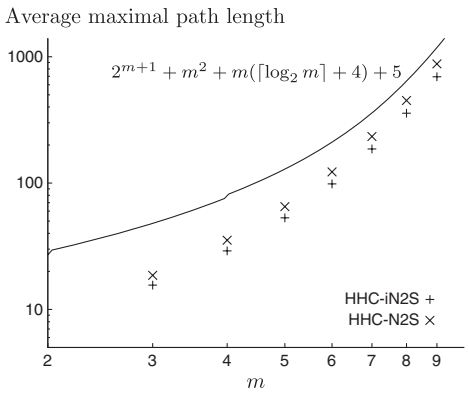

FIGURE 8. Average maximal path length.

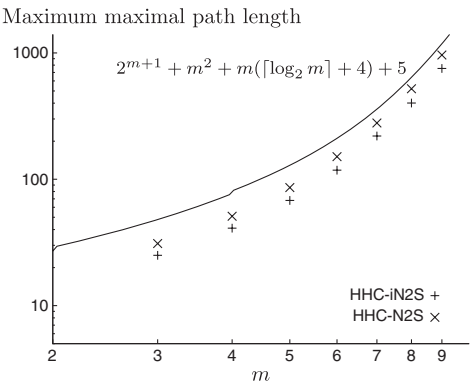

FIGURE 9. Maximum maximal path length.

HHC-iN2S. The average gain measured for the average maximal path length ranges from 16 to 21%, and from 19 to 23% for the maximum maximal path length.

6. CONCLUSION

In this paper, we proposed a new algorithm HHC-iN2S improving the maximum path length established by HHC-N2S in [7], reducing it from $O(m2^m)$ to $O(2^m)$. Inside an HHC_{2^m+m}, given a source node and a set of k ($\leq m+1$) destination nodes, HHC-iN2S finds disjoint paths between the source node and all the destination nodes of lengths at most $2^{m+1} + m^2 + m(\lceil \log_2 m \rceil + 4) + 5$ in $O(km2^m)$ time complexity. The algorithm was implemented to conduct several experiments so as to challenge the theoretical results and perform a comparison between HHC-N2S and HHC-iN2S. We measured an average reduction of ∼20% of the maximum path length when using HHC-iN2S.

Showing that the theoretical maximum path length is not attainable and subsequently reducing it are the aims included in future works. Also, applying a similar approach to solve the same problem in a metacube, which is a generalization of an HHC, is an interesting future work.

FUNDING

This work was supported by the Fund for Promoting Research on Symbiotic Information Technology of Ministry of Education, Culture, Sports, Science and Technology (MEXT) Japan; and a Grant-in-Aid for Scientific Research (C) of the Japan Society for the Promotion of Science (JSPS) [22500041].

REFERENCES

[1] Kaneko, K. and Peng, S. (2008) Node-to-Set Disjoint Paths Routing in Dual-Cube. *Proc. 9th Int. Symp. Parallel Architectures, Algorithms, and Networks*, Sydney, Australia, May 7–9, pp. 77–82. IEEE Computer Society, Los Alamitos, CA.

[2] Li, Y., Peng, S. and Chu, W. (2010) Metacube—a versatile family of interconnection networks for extremely large-scale supercomputers. *J. Supercomput.*, **53**, 329–351.

[3] Suzuki, Y. and Kaneko, K. (2003) An algorithm for node-disjoint paths in pancake graphs. *IEICE Trans. Inf. Syst.*, **E86-D**, 610–615.

[4] Malluhi, Q.M. and Bayoumi, M.A. (1994) The hierarchical hypercube: a new interconnection topology for massively parallel systems. *IEEE Trans. Parallel Distrib. Syst.*, **5**, 17–30.

[5] Wu, J. and Sun, X.-H. (1994) Optimal cube-connected cube multicomputers. *J. Microcomput. Appl.*, **17**, 135–146.

[6] Wu, R.-Y., Chen, G.-H., Kuo, Y.-L. and Chang, G.J. (2007) Node-disjoint paths in hierarchical hypercube networks. *Inf. Sci.*, **177**, 4200–4207.

[7] Bossard, A., Kaneko, K. and Peng, S. (2011) Node-to-Set Disjoint-Path Routing in Perfect Hierarchical Hypercubes. *Proc. 11th Int. Conf. Computational Science*, Singapore, June 1–3. Elsevier, Amsterdam.

[8] Gu, Q.-P. and Peng, S. (1997) Node-to-set disjoint paths with optimal length in star graphs. *IEICE Trans. Inf. Syst.*, **E80-D**, 425–433.

[9] Gu, Q.-P. and Peng, S. (1996) An efficient algorithm for node-to-node routing in hypercubes with faulty clusters. *Comput. J.*, **39**, 14–19.

[10] Findler, R.B., Clements, J., Flanagan, C., Flatt, M., Krishnamurthi, S., Steckler, P. and Felleisen, M. (2002) DrScheme: a programming environment for scheme. *J. Funct. Program.*, **12**, 159–182.