Randomized Protocols for Low-Congestion Circuit Routing in Multistage Interconnection Networks

Richard Cole* Bruce M. Maggs† Friedhelm Meyer auf der Heide‡ Michael Mitzenmacher§

Andréa W. Richa† Klaus Schröder¶ Ramesh K. Sitaraman‖ Berthold Vöcking‡

Abstract

In this paper we study randomized algorithms for circuit switching on multistage networks related to the butterfly. We devise algorithms that route messages by constructing circuits (or paths) for the messages with small congestion, dilation, and setup time. Our algorithms are based on the idea of having each message choose a route from two possibilities, a technique that has previously proven successful in simpler load balancing settings. As an application of our techniques, we propose a novel design for a data server.

1 Introduction

In this paper, we devise algorithms for routing messages in circuit-switching networks where each message chooses from two possible routes, an idea that has been applied with great success in other load balancing situations [12, 17, 26, 27].

Underlying every parallel computer is a network that delivers messages between processors or between processors and memory modules. Similar networks are found in the switches that route telephone calls and internet traffic. Typically, a message is sent from its input node (source) to its output node (destination) via a path in the network. Methods for routing messages include circuit-switching, store-and-forward routing, and wormhole routing. With circuit switching, each message must first lock down (i.e., reserve) a path (i.e., circuit) in the network from its input node to its output node. The path is then used to transmit the message through the network. In contrast, with store-and-forward routing and wormhole routing paths are not reserved before transmission.

Circuit-switching has enjoyed widespread popularity since its early use in telephony and subsequently in the design of parallel computers. Recent trends in network design emphasize the need for providing quality of service (QoS) guarantees for communication. To provide guarantees as opposed to just best-effort service, network resources must be reserved before communication begins. Consequently, several modern high-speed multimedia switches and ATMs reserve a (virtual) circuit through the network for each communication request [37, 38].

1.1 Circuit routing algorithms and their performance

In a circuit-switched network, a message arrives requesting a path from its source to its destination. A *routing algorithm* determines which of many possible paths is locked down for each message. We measure the performance of a routing algorithm in terms of three parameters: congestion, dilation, and setup time.

Congestion and dilation are properties of the paths locked down for the messages by the routing algorithm. The *congestion* of a set of paths is defined to be the maximum number of paths that pass through any link in the network. Congestion is a measure of the maximum number of paths that must be simultaneously supported by a link of the network, and hence determines the bandwidth that a link should possess. The *dilation* of a set of paths is defined to be the maximum length of a path in the set. Dilation is a measure of maximum distance (in links) that a message must travel to reach its destination. Finally, the *setup time* is the time taken by the routing algorithm to allocate paths through the network. This is the time overhead involved in path selection before the actual message transmissions begin.

The goal of this paper is to devise routing algorithms with small congestion, dilation, and setup time.

1.2 Network and problem definitions

The results in this paper apply to variants of a popular type of multistage interconnection network called the *butterfly network*. Butterfly networks and its variants have been widely used for packet routing in a number of commercial and experimental networks [7, 15, 28, 29]. More recently, several proposed designs for the switching fabric of scalable high-speed ATM networks use the butterfly and its variants for routing virtual circuits [37, 38].

We define an n-input *butterfly network* B_n as follows. An n-input butterfly has $n(\log n + 1)$ nodes arranged in $\log n + 1$ *levels* of n nodes each.[1] An example of an n-input butterfly ($n = 8$) with depth $\log n$ ($\log n = 3$) is show in Figure 1. Each node has a distinct label $\langle w, i\rangle$ where i is the level of the node ($0 \le i \le \log n$)

*Courant Institute, New York University, New York, NY 10012 (email: cole@cs.nyu.edu). Supported by NSF grant CCR-9503309.

†School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213 (email: bmm,aricha@cs.cmu.edu). Supported in part by the Air Force Materiel Command (AFMC) and ARPA under Contract F196828-93-C-0193, by ARPA Contract N00014-95-1-1246, and by an NSF National Young Investigator Award, No. CCR-94-57766, with matching funds provided by NEC Research Institute and Sun Microsystems. The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of AFMC, ARPA, CMU, or the U.S. Government.

‡Department of Mathematics and Computer Science, and Heinz Nixdorf Institute, University of Paderborn, 33095 Paderborn, Germany (email: fmadh,voecking@uni-paderborn.de). Supported in part by DFG-Sonderforschungsbereich 376 and by EU ESPRIT Long Term Research Project 20244 (ALCOM-IT).

§Digital Systems Research Center, Palo Alto, CA 94301 (email: michaelm@pa.dec.com).

¶Department of Mathematics and Computer Science, and Heinz Nixdorf Institute, University of Paderborn, 33095 Paderborn, Germany (email: ellern@uni-paderborn.de). Supported by DFG Graduate College "Parallele Rechnernetze in der Produktionstechnik".

‖Department of Computer Science, University of Massachusetts, Amherst, MA 01003 (ramesh@cs.umass.edu). Supported in part by an NSF CAREER Award No. CCR-97-03017.

[1] Throughout this paper we use $\log n$ to denote $\log_2 n$.

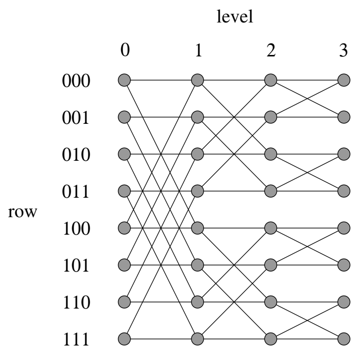

Figure 1: An 8-input butterfly network.

and $w = w_1 w_2 \ldots w_{\log n}$ is a $\log n$-bit binary number that denotes the *row* of the node. All nodes of the form $\langle w, i\rangle$, $0 \le i \le \log n$, are said to belong to row w. Two nodes $\langle w, i\rangle$ and $\langle w', i'\rangle$ are linked by an edge if $i' = i + 1$ and either w and w' are identical or w and w' differ only in the bit in position i'. (The bit positions are numbered 1 through $\log n$.) We call the first type of edge a *straight edge* and the second a *cross edge*. The nodes on level 0 are called the *inputs* of the network, and the nodes on level $\log n$ are called the *outputs*. Sometimes the level 0 node in each row is identified with the level $\log n$ node in the same row. In this case, the butterfly B_n is said to *wrap around.*

We define a *randomly-wired butterfly* RB_n as follows. Network RB_n has the same set of nodes and edges as B_n, except that the cross edges incident on the input nodes of RB_n are permuted randomly according to the following rule. Let $d = \log n$ Each node $\langle w_1 \ldots w_d, 0\rangle$ of RB_n is connected by a cross edge to node $\langle w'_1 \ldots w'_d, 1\rangle$ if and only if $w_1 \ne w'_1$ and $\sigma_{w_1}(w_2 \ldots w_d) = w'_2 \ldots w'_d$, where σ_0 and σ_1 are random permutations of the set of $(\log n - 1)$-bit numbers.

We define a *two-fold butterfly* BB_n as follows. Network BB_n consists of two copies of B_n placed one after the other such that each output node in the first copy is identified with the corresponding input node of the second copy with the same row number. Note that BB_n is a multistage network with n rows and $2\log n + 1$ levels. The nodes in level 0 are called the inputs of BB_n and the nodes in level $2\log n$ are called the outputs of BB_n. Also, observe that a routing algorithm on BB_n can be simulated by making two passes through a butterfly B_n that wraps around.

It is important to contrast the BB_n network with another common variant of the butterfly, the *Beneš network*. An n-node Beneš network consists of two copies of B_n placed "back-to-back" such that each output node of the first copy is identified with the corresponding output node of the second copy.

In this paper, we study a canonical circuit routing problem that is known as the permutation routing problem. In a *permutation routing problem* at most one message originates at each input of the network and at most one message is destined for each output of the network.

We distinguish two kinds of permutation routing problems: static and dynamic. In a *static problem*, all the messages that constitute a permutation routing problem are present at time 0, before the routing begins. The routing algorithm constructs paths for all the messages in a "batch" mode. All the messages are delivered to their respective destinations before the routing of the next batch of messages begins. In contrast, in a *dynamic problem*, messages are injected or deleted one by one. The routing algorithm routes a path for each injected message in an on-line fashion with no knowledge of future message arrivals. We assume that at any time, the messages being routed form a partial permutation; that is, each input and output node correspond to at most one routed message.

1.3 Previous work

There are several different sub-areas of research that relate to our work. We provide a summary of the most relevant.

Routing in Butterfly Networks. There is a vast literature on routing in butterfly networks [20, 21]. Much of the early work focuses on store-and-forward routing [1, 23, 24, 31, 35, 39, 41, 42]. More recently, there has been progress in analyzing wormhole routing algorithms [10, 11, 13, 36]. Since we present no new results in these two routing methods, we focus only on the butterfly circuit-switching literature.

In two early papers, Beizer [8] and Beneš [9] showed that any static permutation routing problem can be routed with congestion 1 and dilation $2\log n$ on an n-input Beneš network. Subsequently, Waksman [43] provided an elegant algorithm that takes $O(n \log n)$ time to determine all the paths, but requires global knowledge of the source and destination of all the messages. Later, Nassimi and Sahni [30] showed how to implement Waksman's algorithm in parallel on the Beneš and related networks in time $O(\log^4 n)$. However, their algorithm is complex and requires the Beneš network to emulate a complete network by executing a series of sorting routines.

Although the Beneš network and the BB_n are closely related in structure, whether or not it is possible to route an arbitrary permutation routing problem in an offline fashion with congestion 1 on the BB_n is a long-standing open problem.

In this paper, we devise routing algorithms that *minimize congestion*. A complementary approach aims to maximize throughput. Previous work has studied the model where each link can support at most q paths, and the goal is to maximize the number of messages that lock down paths. Kruskal and Snir [19] showed that if each input in a butterfly network B_n sends a message to a randomly chosen output, and at most one message can use any edge of the network (i.e., $q = 1$), then the expected number of messages that succeed in locking down paths to their destinations is $\Theta(n/\log n)$. Koch [18] generalized the result of Kruskal and Snir by showing that if each edge can support q messages, $q \ge 1$, then the expected fraction of messages that succeed in locking down paths is $\Theta(n/\log^{1/q} n)$. Maggs and Sitaraman [24] generalized the previous two results by showing that, by making two passes through a butterfly, it is possible to route an $\Omega(n/\log^{1/q} n)$ fraction of any permutation (rather than only a random permutation), with high probability.

Use of Randomness. An early example of the use of randomization for circuit-switching in butterfly networks is the work of Valiant [41, 42]. Valiant showed that any permutation routing problem can be transformed into two random problems by first routing a path for each message to a random intermediate destination, and then on to its true destination. This implies that we can route paths for a (static or dynamic) permutation routing problem on a two-fold butterfly BB_n with congestion $\Theta(\log n/\log\log n)$, and dilation $2\log n$. Note that the paths for each message can be set up independently without complete knowledge of the permutation in $O(\log n)$ time. We show how to use randomization to route permutations with substantially smaller congestion and the same dilation.

Ranade [34] observed that a smaller amount of randomness is sufficient to implement Valiant's algorithm. Note that each switch has two input links and two output links. Ranade noted that it is sufficient that each switch in the first $\log n$ levels of BB_n shunts a message from each input link to a random (and distinct) outgoing link. Thus, messages are sent to random but not independent destinations using one random bit per switch. The first $\log n$ levels of such a BB_n constitute a *flip network*. A flip network was subsequently used in [24] in the context of circuit routing. We use flip networks in our routing algorithms in Section 2.

Randomness can be used in constructing the network itself. The use of randomness to design multistage networks dates back to Ikeno[16], and Bassalygo and Pinsker [5]. Networks such as the randomly-wired multibutterfly are known to have good routing and fault tolerance properties [40, 22]. Recent results provide algorithms for routing circuits for any permutation routing problem with congestion 1 in multibutterfly and multi-Beneš networks with set-up time $O(\log n)$ [2, 32]. Unlike these networks, our results in Section 2 apply to commonly-used networks like B_n and BB_n that require neither random wiring nor expanders.

Balls-and-bins problem. Our approach to circuit routing is influenced by recent advances in the classical balls-and-bins problem. It is well known that if n balls are tossed randomly into n bins, the maximum number of balls in any bin will be $\Theta(\log n/\log\log n)$ with high probability. Azar et al [4] consider the following dynamic protocol for throwing n balls into n bins: for each ball pick two bins independently and uniformly at random, and place the ball in the bin with the smaller load at the time of placement. They show that after all balls are placed in bins, the maximum load of any bin is $\Theta(\log\log n)$, with high probability.

Static protocols for the balls-and-bins problem were developed in [17], [12], and [27] and applied to PRAM simulations. They consider variants of the following process. Initially, each ball chooses two random bins. In a round, each ball not yet allocated accesses its two bins. Each bin with at most c accessing balls accepts all of them. The other balls try again in the next round. This protocol guarantees maximum load c. Even for constant c, the protocol allocates all balls, with high probability, using only $O(\log\log n)$ rounds.

We apply similar "two-choice" algorithms to circuit routing. Note that this is a more complex situation. Thinking of each message as a ball and each network edge as a bin, we see that finding a path for each message corresponds to placing each ball in several dependent bins. These dependencies substantially increase the difficulty of the analysis.

Circuit routing in general topology networks. Dynamic circuit-switching has been extensively studied in an on-line competitive framework for arbitrary network topologies. (See [33] for a survey). Results are known for minimizing congestion [3] and for the maximizing throughput [14]. This framework can incorporate more general parameters such as the circuit bandwidth and circuit holding time. However, these results do not yield routing algorithms with congestion smaller than $\Theta(\log n)$ for the regularly-structured muti-stage networks that are the focus of this paper.

1.4 Our results

We introduce two new protocols for circuit-routing: the collision protocol and the minimum protocol. In contrast to Valiant's algorithm, which picks one random path for each message, these protocols choose *two* random (but not independent) paths p and p' for each message M. The *collision protocol* uses a suitably chosen threshold c, and allocates either p or p' to message M, provided the congestion of the allocated path is at most c. In contrast, the *minimum protocol* allocates to M the path with the smaller congestion. As mentioned previously, protocols of this flavor have been utilized and analyzed in simpler settings. We extend these techniques to circuit-routing.

Static Permutation Routing. In Section 2.1, we show the collision algorithm routes any permutation on the two-fold butterfly BB_n with congestion $O(\log\log n/\log\log\log n)$, with high probability, and dilation $2\log n$. The setup time is $O(\log n\log\log n/\log\log\log n)$. Our routing algorithm achieves a substantially smaller congestion bound than Valiant's algorithm. Comparing our result with Waksman's algorithm, which achieves congestion 1 on a Beneš network, we require substantially smaller setup time. Furthermore, we do not require complete knowledge about the permutation being routed and our routing algorithm can be implemented on the network itself. Comparing our result to the algorithm of Nassimi and Sahni [30], our algorithm is much simpler and faster, although their algorithm achieves smaller congestion.

Dynamic Permutation Routing. In Section 2.2, we analyze the minimum algorithm for routing any dynamic permutation routing problem on network BB_n. The congestion is $O(\log\log n)$ with high probability, the dilation is $2\log n$, and the setup time for each new message is $O(\log n)$. Prior to this work, every known algorithm for the dynamic permutation routing problem on the butterfly and related networks required $\Omega(\log n/\log\log n)$ congestion. Our algorithm is optimal in that any routing algorithm on BB_n that considers only a constant number of alternate paths per message must incur $\Omega(\log\log n)$ congestion [4].

Data Server Architecture. As an application of our techniques, in Section 3, we present a proposal for the architecture of a data server. The data server utilizes network RB_n to connect n users to n disks. Each user is associated with a distinct input node and each disk is associated with a distinct output node of RB_n. Objects (typically large, e.g. movies) are distributed among the disks.

A canonical task performed by the data server is the following. Given n requests to objects, one per user, these requests must be satisfied by providing a path from each user to a disk that contains their requested object. The congestion of the paths must be minimized. Besides congestion, another important performance metric is disk contention, which is often a bottleneck. We define *disk contention* to be the maximum number of simultaneous requests that any disk must satisfy. In Section 3, we devise algorithms that achieve both small congestion and small disk contention.

The standard technique of storing the objects by independently and randomly distributing them to the n disks yields congestion and disk contention $\Theta(\log n/\log\log n)$, with high probability. To achieve lower congestion and disk contention, we store two copies of the same object on two disks.

2 Routing in the two-fold butterfly

2.1 Static routing in BB_n

We describe a simple, efficient algorithm for routing permutations on the two-fold butterfly BB_n. Recall that the two-fold butterfly BB_n has n inputs at level 0 and n outputs at level $2d$, where $d = \log n$. Given a permutation π, our routing algorithm connects each input node i to the corresponding output node $\pi(i)$; each pair $(i, \pi(i))$ of input and output nodes is called a *request*. Our randomized algorithm routes paths such that the maximum congestion on an edge is $\Theta(\log\log n/\log\log\log n)$, with high probability. Furthermore, the time required by the algorithm to set up all the paths is at most $\Theta(\log n\log\log n/\log\log\log n)$, with high probability.

The c-collision algorithm. We use the collision protocol described below to perform the routing. The c-collision protocol initially chooses at random two possible paths for each request. Eventually one of these paths will serve as the required connection.

The two random paths for each request are chosen as follows. The nodes on levels $0,\ldots,d/2-1$ and $d+d/2+1,\ldots,2d$ are flipped randomly. In particular, each input and output node maps the *first path* p of a request to its straight edge and its *second path* p' to its cross edge with probability $\frac{1}{2}$,and with probability $\frac{1}{2}$ the order is reversed. Similarly, each node on levels $1,\ldots,d/2-1$ and $d+d/2+1,\ldots,2d-1$ with probability $\frac{1}{2}$ connects its input straight edge with its output straight edge and its input cross edge with its output cross edge, and with probability $\frac{1}{2}$ the connections are reversed. Note that these random choices completely determine the two paths p and p' of each request, because there is exactly one path connecting a node on level $d/2$ with a node on level $d + d/2$ in a BB_n network. For a path p, the other path p' connecting the same input and output nodes is called the *buddy* of p. The random switching ensures that any edge on the levels $1, \ldots, d/2$ and $d+d/2+1, \ldots, 2d$ is traversed by at most one of the randomly-generated paths. However, each edge on the interior levels, i.e., one with "top" node on one of the levels $d/2 + 1, \ldots, d + d/2$, is potentially traversed by several of these paths. We call these edges *collision edges*, and we say that two paths that cross the same collision edge *collide*.

The c-collision algorithm proceeds in rounds to select a path for each request as follows. Initially all paths are *active* and not *selected*. A path p is eligible to be selected if for each edge $e \in p$ the number of active paths traversing e is at most c. If p and its buddy p' are both eligible to be selected, only one is selected arbitrarily. A path p ceases to be active in a round if p is selected or the buddy of p is selected in that round. The algorithm terminates when there are no more active paths.

Each round of the c-collision algorithm can be implemented using a store-and-forward algorithm as a subroutine: in a first pass, for each active path, a packet is sent along the path from level 0 to level $2d$. During this pass, for each edge, the number of packets traversing the edge is counted. Then, in a second pass, all packets are routed backward along their respective paths from level $2d$ to level 0. During this pass the congestion for each active path is computed. Note that, in this model, when computing the setup time the packets and edges of the network can act in parallel, and hence a round may complete in $o(n)$ time.

The c-collision algorithm selects a path p in a round only if p collides with no more than $c - 1$ other active paths on any of the edges in p. This implies that any edge that is included in at least one selected path is included in at most $c - 1$ other selected or active paths. As a consequence, the congestion of all selected paths is at most c. Note that the algorithm as described is not guaranteed to terminate. However, in Theorem 2.1, we show that if c is sufficiently large, the algorithm will terminate with maximum congestion at most c, after a small number of rounds, with high probability. In practice, we may terminate the algorithm after some fixed number of rounds; all requests that still have two active paths at the termination point may choose one arbitrarily, and in this case we fail to guarantee congestion c.

Theorem 2.1 *For any constant $\epsilon > 0$ and c such that $c! = (1 + \epsilon) \cdot \log n$, the probability that the c-collision algorithm on BB_n takes more than $t = \Theta(\log\log n / \log\log\log n)$ rounds to select a path for every request is at most $n^{-c/4+1+o(1)}$. Furthermore, each round can be computed in time $O(\log n)$, with high probability.*

Proof. First, we show that if the algorithm does not terminate after t rounds, we can construct a "witness tree". Next, we show how the witness tree can be pruned to avoid stochastic dependencies. Finally, we show by enumeration that the probability of occurrence of a pruned witness tree is at most $n^{-c/4+1+o(1)}$.

Constructing a witness tree. Fix a permutation π to be routed, and the settings of the randomly-flipped switches on the levels $0, \ldots, d/2 - 1$ and $d + d/2 + 1, \ldots, 2d$. This determines the two paths chosen for each request. Assume that there is a request with paths p and p', and neither path has been selected by round t, where the proper value of t is to be determined later. Then p collides with at least c paths of other requests in round t at some edge e. Let $p_1, \ldots, p_c$ denote the c paths that collide with p in round t at e. The root of the witness tree is the request corresponding to p and the requests corresponding to $p_1, \ldots, p_c$ are its children. The paths $p_1, \ldots, p_c$ and their buddies $p'_1, \ldots, p'_c$ were not selected at round $t - 1$. Applying the argument recursively to $p'_1, \ldots, p'_c$ we can construct a complete c-ary tree of height t. This tree is called the *witness tree*.

Each node v in the witness tree corresponds to a request with two associated paths, one of which collides with one of the paths associated with each sibling and the parent of v (unless v is the root), and the other of which collides with one of the paths associated with each of the children of v (unless v is a leaf). We call the first path the *up path* of v and the other path the *down path* of v. The up path of the root and the down paths of the leaves are defined to be empty paths. Note that by the term "collision represented by node v" we mean the collision of the down path of v with the up paths of the children of v in the witness tree. Finally, to give each tree a unique representation, we assume that children of a node are listed in increasing order from left to right based on the input node number of the corresponding request.

The requests corresponding to the nodes of a witness tree are not necessarily pairwise distinct. Furthermore, the up and down paths of distinct requests may overlap in the randomly-flipped levels, so that a randomly-flipped switch can be included in more than one of these paths. Hence, the collision events represented by a witness tree are not necessarily stochastically independent. Note that, if they were stochastically independent, it would be relatively straightforward to argue the theorem.

Pruning the witness tree. The intuitive reason why the dependencies do not affect the final conclusion is that there are only $O(\log n)$ nodes in the witness tree, hence the dependencies are "rare". In order to handle dependencies, we prune nodes from the witness tree as necessary. This pruning is done by a traversal through the tree visiting the internal nodes in breadth-first-search order starting at the root. When a node v is visited during this traversal, the dependencies between the collision represented by v and the collisions represented by nodes visited before v are checked. If the dependencies significantly affect our calculations, the nodes below v are pruned, and these pruned nodes are excluded from the subsequent traversal.

The detailed pruning rules follow. For a node v visited during the traversal, let $B(v)$ denote the set of nodes visited before v. Furthermore, let $\Gamma(v)$ denote the set of nodes that are children of the nodes in $B(v)$, that are not pruned before v is visited, and that are not in $B(v)$ themselves. For the root r of the witness tree, $B(r)$ and $\Gamma(r)$ are empty since our traversal starts at r. We distinguish two pruning rules:

1. If a path associated with one of v's non-pruned children traverses a randomly-flipped switch that is also traversed by a path associated with a node u from $\Gamma(v)$ then the c subtrees rooted at the children of v are removed from the tree, and the c subtrees rooted at the children of u are also removed from the tree. The node v is called a *pruning node*. The node u that caused the pruning is called the *conflicting node* of v.

2. If a path associated with one of v's non-pruned children traverses a randomly-flipped switch that is also traversed by a path associated with a node u from $B(v)$ then the c subtrees rooted at the children of v are removed from the tree. The nodes v and u are again called pruning and conflicting nodes respectively.

When there is more than one choice for a conflicting node for a certain pruning node we make the choice arbitrarily, so that each pruning node can be associated with exactly one conflicting node. Furthermore, the second pruning rule is considered only if the conditions for the first pruning rule are not met.

Note that the pruning rules ensure that, for every node v visited after the root r, the subgraph induced by $B(v) \cup \Gamma(v)$ is connected; that is, $B(v) \cup \Gamma(v)$ induces a subtree of the full witness tree with root r. Also, when a node v is visited, up to $2c$ subtrees of maximum height $t - 2$ could be pruned from the tree. These subtrees do not include any node from $B(v) \cup \Gamma(v)$. Hence, the subtree induced by this set only grows during the traversal.

We continue the pruning process till either there are no more nodes to visit or there are $\kappa = \lceil c/2 \rceil$ pruning nodes. In the latter case, we apply a final pruning. If v is the κth pruning node, we remove from the tree all nodes not included in $B(v) \cup \Gamma(v)$. This effectively stops the pruning process at the κth pruning node.

The witness tree pruned in this fashion is called the *pruned witness tree*. Let m denote the number of internal nodes in this tree, and $m' \leq \kappa$ denote the number of pruning nodes. Let $v_1, \ldots, v_m$ denote the internal nodes and $w_1, \ldots, w_{m'}$ the pruning nodes in order of visitation, respectively. Furthermore, let u_i denote the conflicting node of w_i, for $1 \leq i \leq m'$. The pruned tree possesses the following properties.

1) Any internal node v represents a collision of the down path of v and the c up paths of the children of v.

2) For any internal node v, the pruning ensures that the up paths of the children of v do not share a randomly-flipped switch with a path associated with a node in $B(v) \cup \Gamma(v)$ except for the down path of v. (As a consequence, all nodes of the tree correspond to distinct requests.)

3) The down path of a pruning node v either collides with a path p that is associated with the conflicting node u, or it collides with a path p such that p or its buddy shares a random switch with a path associated with u. This path p is denoted the *conflicting path* of v.

4) The down path of a pruning node w_i is not the conflicting path of a pruning node w_k with $k < i$. (This can be proved as follows. For contradiction, assume the opposite. Then $w_i = u_k$ and $w_i \in \Gamma(w_k)$. Hence, the subtree below w_i is removed when w_k is visited. This means that w_i has no non-pruned children when w_i is visited and consequently, w_i is not a pruning node.)

5) For each pruning node w_i, the down path p of w_i shares at most $5c$ randomly flipped switches with up and down paths associated with any other node and conflicting paths associated with the pruning nodes $w_1, \ldots, w_i$. (This is because, according to Properties 2 and 4, the down path of w_i is not equivalent to any such up, down, or conflicting path. Furthermore, according to Property 2, the down path of w_i does not share a random switch with any other up or down path, except for the up and down paths of the siblings of w_i, and the up path of w_i. With each of these $2c-1$ paths, the down paths overlaps at most twice in the randomization levels, once in each of the butterflies in BB_n. The same holds for the i conflicting paths associated with $w_1, \ldots, w_i$. Thus, there are at most $4c - 2 + 2\kappa \leq 5c$ overlappings with these paths in the randomization levels.)

Bounding the probability of occurrence of a pruned witness tree. We bound the probability of occurrence of a pruned witness tree via enumeration. Define the *tree shape* to be a description of the topology of the tree including the pruning and the conflicting nodes. Define an *admissible witness tree configuration* to be a tree shape with associated requests, up and down paths, and conflicting paths which eventually, i.e., for some setting of the random switching, matches to a pruned witness tree. In particular, any admissible witness tree configuration has to fulfill the 5 properties above.

Let $\mathcal{T}$ denote the set of tree shapes corresponding to at least one admissible witness tree configuration, and let $\mathcal{K}_T$ denote the set of all admissible witness tree configurations with tree shape $T \in \mathcal{T}$. An admissible configuration K is said to be *active* if the outcome of the random switching corresponds to all paths of the configuration. Hence, each admissible configuration K has a probability to become active, which is just $2^{-\rho(K)}$ with $\rho(K)$ denoting the total number of randomly flipped switches covered by all paths of K. As a consequence, the probability that the c-collision process takes more than t rounds can be bounded by

$$\sum_{T \in \mathcal{T}} \underbrace{\sum_{K \in \mathcal{K}_T} 2^{-\rho(K)}}_{=: \, E(T)} \ .$$

We aim to give an upper bound on $E(T)$, for a fixed tree shape $T \in \mathcal{T}$. $E(T)$ is equal to the *expected number of active witness tree configurations* with tree shape T. Note that the tree shape T only restricts the number of admissible configurations, that is, it defines the set $\mathcal{K}_T$, but does not influence the probability for a given configuration $K \in \mathcal{K}_T$ to become active. This probability depends only on $\rho(K)$, and, hence, on the overlapping of the paths in the randomization levels.

In the following, we utilize Properties 2 and 5 that govern how paths may overlap to compute $E(T)$. Instead of summing over all admissible configurations in $\mathcal{K}_T$ and multiplying each individual configuration with its probability, we consider the nodes of the witness tree one by one and calculate an upper bound on the expected number of configurations for each individual node. In particular, we consider first all the internal tree nodes and then all the pruning nodes; both sets of nodes are considered in the order of visitation.

Define the *configuration of an internal node* v_i to consist of the down path of v_i and the up paths of the children of v_i, for $1 \leq i \leq m$. Furthermore, define the *configuration of a pruning node* w_i to be the down path of w_i and the two paths belonging to the colliding request of w_i, for $1 \leq i \leq m'$. A collection of node configurations is said to be admissible, if they are a subset of an admissible tree configuration. Note that a collection of admissible configurations for all internal and all pruning nodes (in conjunction with the tree shape) completely defines the configuration of the witness tree.

For an internal node v_i and a collection K of configurations for the nodes $v_1, \ldots, v_{i-1}$, let $E_{\text{coll}}(v_i, K)$ denote the expected number of active configurations for v_i under the assumption that the configurations in K are active. Note that K already specifies the request associated with v_i. (For the root v_1 we assume that K specifies only this request.) Let $E_{\text{coll}}(v_i)$ be the maximum over all configurations K of $E_{\text{coll}}(v_i, K)$.

Lemma 2.2 $E_{\text{coll}}(v_i) \leq \log n / c!$.

Proof. We bound the expected number of active configurations for v_i by choosing the down path p of v_i arbitrarily and then deriving an upper bound on the expected number of choices of active up paths $p_1, \ldots, p_c$ of the children of v_i that fulfill Properties 1 and 2.

The expected number of active down paths p is at most one. This is because, there are several different paths in BB_n that connect the two input and output nodes which are given by the configuration K. However, at most two of them are active, and the configuration K determines which of them is the up path and which is the down path of v_i.

Given path p, there are $d = \log n$ possible choices for the collision edge at which the down path collides with $p_1, \ldots, p_c$. Let e denote this edge and ℓ the level of this edge. W.l.o.g., we assume that $d/2 + 1 \leq \ell \leq d$.

We calculate an upper bound on the expected number of active up paths $p_1, \ldots, p_c$ traversing e and fulfilling Property 2. Property 2 ensures that $p_1, \ldots, p_c$ use only unrevealed random switches. Therefore, we assume for the following that all switches are unrevealed. Note that this does not decrease the number of admissible configurations, and, hence, not decrease the expected number of active configurations for $p_1, \ldots, p_c$. The main problem in calculating the number of active configurations for $p_1, \ldots, p_c$ is to handle overlappings among these paths and overlapping between these paths and the down path p in the randomization levels.

The number of nodes on level 0 from which e can be reached is $2^{\ell-1}$. We select an input node for each of the p_i's from these nodes. The number of possible ways to choose these c nodes is $\binom{2^{\ell-1}}{c}$ because the requests associated with the children of a node are ordered according to the ID's of the input nodes. Let $s_1, \ldots, s_c$ denote the source nodes of the paths $p_1, \ldots, p_c$ on level 0 and $d_1 = \pi(s_1), \ldots, d_c = \pi(s_c)$ the destination nodes of these paths on level $2d$.

Next we choose an intermediate destination d_i' for each path p_i on node level $d+\ell$. For every p_i, there are (eventually) several possibilities to choose these intermediate destination. However, independent from the other paths of the configuration of v_i, the number of active destinations is at most one. Hence, the expected number of active intermediate destinations is at most one.

Now assume the intermediate destinations are fixed. Note that this also fixes the path from level $d+\ell$ to level $2d$. It remains to consider the number of active configurations of c paths $p_1', \ldots, p_k'$ such that p_i' connects s_i and d_i' and traverses e. Paths $p_1', \ldots, p_k'$ and p do not overlap in the randomization levels. This can be shown as follows. If two paths share a random switch s then these paths arrive and leave s on different edges. Furthermore, these paths do not overlap at any other switch with distance less than $d+1$ from s. Hence, two paths that traverse edge e cannot have used a random switch with distance less than $d+1$ from the two switches adjacent to e, and consequently, they cannot meet on a random switch on the levels $0, \ldots, d/2-1$ or the levels $d+d/2+1, \ldots, d+\ell$.

The number of different paths connecting s_i with d_i' and traversing e is one. Thus, the number of admissible configuration for the p_i''s is at most one. All paths in the admissible configuration do not share a randomly flipped switch with another path from the configuration of v_i. Hence, the number of unrevealed random switches traversed by each of these paths is $d/2 + (d+\ell) - (d+d/2) = \ell$. Except for the switch on level 0, all of these switches must correspond to the course of the respective path. The probability for this event is $2^{-(\ell-1)}$. As a consequence, the probability that all k paths are active is at most $2^{-c\cdot(\ell-1)}$.

Putting it all together, the expected number of active configurations for v_i is

$$d \cdot \binom{2^{\ell-1}}{c} \cdot 2^{-c\cdot(\ell-1)} \leq \frac{d}{c!} \ ,$$

which completes the proof of Lemma 2.2. ■

Now we give an upper bound on the expected number of the active configurations for the pruning nodes. For a pruning node w_i and a collection K of configurations for all internal nodes and the pruning nodes $w_1, \ldots, w_{i-1}$, let $E_{\text{prune}}(w_i, K)$ denote the expected number of active configurations for w_i under the assumption that all configurations in K are active. Let $E_{\text{prune}}(w_i)$ be the maximum over all configurations K of $E_{\text{prune}}(w_i, K)$.

Lemma 2.3 $E_{\text{prune}}(w_i) \leq 2^{5c+3} \cdot (\log n + 1)/\sqrt{n}$.

Proof. The conflicting path p of pruning node w_i is either associated with the conflicting node u_i or p or its buddy shares a randomly flipped switch with a path associated to u_i. The tree shape specifies u_i, and the configuration K fixes the request associated with u_i. For any consistent setting of the random switches, the number of paths sharing a randomly flipped switch with the two paths belonging to this request is at most $2 \cdot (\log n + 1)$ (inclusive the two paths themselves). Consequently, for any setting of the switches, the number of candidates for the collision request is at most $2 \cdot (\log n + 1)$, and, hence, the number of candidates for the collision path is at most $4 \cdot (\log n + 1)$.

Now suppose the collision path is fixed. The down path of w_i collides with this path. First, we assume that the collision is in level ℓ, with $d/2+1 \leq \ell \leq d$. Let e denote the respective collision edge. There is at most one admissible course for the down path of w_i from its source node on level 0, which is determined by K, to the collision edge e.

The course of the down path from level 0 to level ℓ is determined by the randomly flipped switches. Property 5 ensures that at most $5c$ of the switches traversed by the down path are shared with other paths in K. Hence, at least $\ell - 5c$ of the randomly flipped switches determining the course of the path from level 0 to level ℓ are independent of K, and consequently, the probability that the down path of w_i is equivalent to the only admissible path in these levels is $2^{-\ell+5c}$. Summing over all collision levels ℓ, with $d/2+1 \leq \ell \leq d$, yields an upper bound on the probability that the switches along the collision path are set appropriately of $2^{-d/2+5c}$. Since the same bound holds also for collisions when $d+1 \leq \ell \leq 3d/2$, the probability that the down path is equivalent to the only admissible path is at most $2^{-d/2+5c+1}$. As a consequence, the expected number of active configurations for w_i is at most $2^{-d/2+5c+1} \cdot 4 \cdot (\log n + 1) = 2^{5c+3} \cdot (\log n + 1)/\sqrt{n}$. ■

The bound for $E_{\text{coll}}(v_i)$ on the expected number of active configurations for an internal node v_i is independent of the configurations of the internal nodes $v_1, \ldots, v_{i-1}$. Furthermore, the bound for $E_{\text{prune}}(w_i)$ on the expected number of active configurations for a pruning node w_i is independent of the configurations on all internal nodes and the pruning nodes $w_1, \ldots w_{i-1}$. Consequently, these bounds are independent estimations of expected values and can be multiplied in order to get an upper bound on the expected number of all configurations. Since the number of choices for the initial configuration K in $E(v_1, K)$ specifying the request associated with the root is n, we get the following upper bound on the expected number of active witness tree configurations.

$$\begin{aligned}
\sum_{T \in \mathcal{T}} E(T) &\leq \sum_{T \in \mathcal{T}} n \cdot \prod_{i=1}^{m} E_{\text{coll}}(v_i) \prod_{j=1}^{m'} E_{\text{prune}}(w_j) \\
&\overset{(1)}{\leq} n \cdot \sum_{T \in \mathcal{T}} \left(\frac{\log n}{c!}\right)^m \left(\frac{2^{5c+3} \cdot (\log n + 1)}{\sqrt{n}}\right)^{m'} \\
&\overset{(2)}{\leq} n \cdot \sum_{T \in \mathcal{T}} \left(\frac{2^{5c+3} \cdot (\log n + 1)}{\sqrt{n}}\right)^{\kappa} \\
&\overset{(3)}{\leq} n \cdot c^{2\kappa t + \kappa} \cdot \left(\frac{2^{5c+3} \cdot (\log n + 1)}{\sqrt{n}}\right)^{\kappa} \\
&\leq n^{-c/4+1+o(1)} \ ,
\end{aligned}$$

for $\kappa = \lceil c/2 \rceil = \Theta(\log\log n / \log\log\log n)$ and a suitably large $t = \Theta(\log\log n / \log\log\log n)$.

Equation 1 is an immediate consequence of Lemma 2.2 and Lemma 2.3.

Equation 2 is based on the relationship between m and m': The full witness tree includes c disjoint subtrees of height $t-1$. For each of the m' pruning nodes, some nodes from at most two of these subtrees are removed. Consequently, at least $c - 2m'$ of the subtrees remain untouched. Since each of them include at least c^{t-2} internal nodes, we get

$$m \geq (c - 2m') \cdot c^{t-2} \geq (\kappa - m') \cdot c^{t-2} \ .$$

Applying this equation and substituting $c! = (1+\epsilon) \cdot \log n$ yields

$$\left(\frac{\log n}{c!}\right)^m \quad \leq \quad (1+\epsilon)^{-c^{t-2} \cdot (\kappa - m')}$$

$$\leq \left(\frac{2^{5c+3}\cdot(\log n+1)}{\sqrt{n}}\right)^{\kappa-m'},$$

for $t \geq \log_c \log_{1+\epsilon} n + 2 = \Theta(\log\log n/\log\log\log n)$.

Equation 3 results from a bound on the number of different tree shapes. In particular, there are at most

$$\sum_{j=0}^{\kappa}\binom{(c^t-1)/(c-1)}{j} \leq c^{\kappa t}$$

possible choices for the at most κ pruning nodes among the $(c^t - 1)/(c-1)$ internal nodes of the witness tree, and at most

$$\left(\frac{c^{t+1}-1}{c-1}\right)^{m'} \leq c^{\kappa(t+1)}$$

possibilities to choose the m' conflicting nodes among the $(c^{t+1} - 1)/(c+1) \leq c^{t+1}$ nodes of the full witness tree. Since specifying these nodes completely determines the shape of the tree, the total number of different tree shapes is at most $c^{\kappa t} + c^{\kappa(t+1)} \leq c^{2\kappa t+\kappa}$.

We have already shown that $\sum_{T\in\mathcal{T}} E(T)$ is an upper bound on the probability that the c-collision process takes more than t rounds. Hence, this probability is at most $n^{-c/4+1+o(1)}$. It remains to show that determining which paths become inactive each round can be done in time $O(\log n)$, with high probability. Recall that, in our model, this computation is accomplished by sending a packet back and forth along each active path through the network using a store-and-forward algorithm. According to [23], such a computation can be done in time $O(\text{congestion} + \text{dilation})$, with high probability, using only constant size buffers at each edge. Note here that the congestion we wish to bound is the congestion caused using this store-and-forward scheme, not the congestion under the collision algorithm. However, this congestion is easily bounded. Let C denote the congestion of all $2n$ paths.

Lemma 2.4 $C \leq \alpha\cdot\log n/\log\log n$ *, with probability* $n^{-\alpha+O(1)}$.

Proof. The congestion in the randomization levels is 1. Therfore, we only have to consider the collision levels. The probability that a fixed collision edge is traversed by at least C paths is at most $1/C!$. This bound follows analogously to the proof of Lemma 2.2. Hence, the probability that one of the $2\cdot n\cdot\log n$ collision edges has congestion C is at most

$$2\cdot n\cdot\log n\cdot 1/C! \leq n^{-\alpha+O(1)},$$

for $C > \alpha\cdot\log n/\log\log n$. ■

Applying Lemma 2.4 yields that each round can be computed in time $O(\log n)$, with high probability. This completes the proof of Theorem 2.1. ■

2.2 Dynamic routing in BB_n

We now describe a simple algorithm that routes paths dynamically in the network BB_n, where the dynamic model is specified as follows. As before, a *request* is an input-output pair. An oblivious adversary specifies an infinite sequence $\sigma_1, \sigma_2, \ldots$ of requests. The request σ_i must be handled at time step i. If at time i neither the input nor the output of σ_i is already locked, then the algorithm must establish and lock a path in the network between the input and output of σ_i: This is an *arrival*. If a locked path between the input-output pair already exists, then the path is released: This is a *departure*. In all other cases the request may be ignored. That is, the algorithm only connects an input-output pair if neither is already involved in a connection. Without loss of generality we may assume that the sequence of requests includes only valid arrival and departure events. An input-output pair is said to *exist* at each time k between its arrival and departure.

The minimum algorithm. To solve the dynamic routing problem on the two-fold butterfly BB_n, we initialize BB_n as in Section 2.1. Let s_i denote an arrival event. A path for the corresponding request r_i is chosen as follows. For an edge e in the collision levels, define $c(e)$ to be the number of paths that traverse e at time i. The algorithm examines the two paths p and p' that connect the input to the output of r_i. The congestion $c(p)$ of a path p is defined to be $\max_{e\in p}(c(e))$. If $c(p) \leq c(p')$, path p is chosen for request r_i; otherwise, path p' is chosen.

Theorem 2.5 *At any time t, the probability that the congestion is greater than* $\Theta(\log\log n)$ *is at most* $n^{-\Theta(\log\log n)}$.

Proof. The proof is similar to that of Theorem 2.1.

Constructing a witness tree. First, we fix the settings of the randomly-flipped switches. This determines two choices of paths for each request. Assume that there is an edge e with congestion larger than $4c$ at some time t, where $c = \lceil\log\log n\rceil$. Let p denote the last path mapped to edge e on or before time t. When p was mapped to e there were already $4c$ other paths present at this edge. Let $p_1, \ldots, p_{4c}$ denote these paths such that p_i was mapped to e at time step t_i with $t_i < t_{i+1}$. The root of the tree is the request corresponding to p and the requests corresponding to $p_1, \ldots, p_{4c}$ are its children. Now we consider the buddies $p'_1, \ldots, p'_{4c}$ of these paths. Path p'_i traverses an edge with congestion at least $i-1$ at time step t_i, because the congestion of p_i is not larger than the congestion of p'_i at time i, and when p_i was mapped to e there were already $i-1$ other paths present at this edge. As a consequence, we can construct a tree by applying the argument above recursively to $p'_2, \ldots, p'_{4c}$.

The tree constructed above is irregular in that nodes have varying degrees. However, it contains a c-ary tree of height c, which we call the witness tree, with the following properties.

- The node on level 0, i.e., the root, has c children that are internal nodes.
- Each internal node on levels $1, \ldots, c-2$ has two children that are internal nodes and $c-2$ children that are leaves, and each internal node on level $c-1$ has c children that are leaves.

Pruning the witness tree. The pruning is done by a breadth-first traversal of the tree. We use the same definitions for $B(v)$ and $\Gamma(v)$ as in Section 2.1. However, the pruning rules are slightly different. When a node v is visited, the following rules are applied.

1. If a path associated with one of v's non-pruned children traverses a randomly-flipped switch that is also traversed by a path associated with a node u from $B(v)\cup\Gamma(v)$ then all nodes below v are pruned. Node u is denoted the conflicting node of v. Note that the down path of v either shares a collision edge with a path p that is associated with u, or it shares a collision edge with a path p such that p or its buddy shares a random switch with a path associated with u. This path p is denoted the *conflicting path* of v.

2. Depending on the conflicting path p we apply a further pruning. For each node $u \in \Gamma(v)$ such that either the input or output node of u coincides with the input or output node of path p, we prune all the nodes below u. The first pruning rule ensures that there is at most one request in $B(v)\cup\Gamma(v)$ incident on each input and output of the network, even though

the requests in $B(v) \cup \Gamma(v)$ exist at possibly non-overlapping times. Thus, at most two nodes, call them u and u', get pruned due to an application of this rule. Nodes u and u' are defined to be the conflicting nodes of v. (For simplicity, we pretend that each pruning node v has two conflicting nodes u and u'; if this is not the case we simply set u and u' to be the same node.) The second pruning rule ensures that Properties 4 and 5 as stated in Section 2.1 hold for the pruned witness tree – specifically, the down path of a pruning node cannot share more than two randomly-flipped switches with a given conflicting path.

We continue the pruning process till either there are no more nodes to visit or there are $\kappa = \lceil c/3 \rceil$ pruning nodes. In the latter case, we apply a final pruning. If v is the κth pruning node, we remove from the tree all nodes not included in $B(v) \cup \Gamma(v)$. The remaining tree is called the *pruned witness tree*.

Bounding the probability of occurrence of a pruned witness tree. The terms *tree shape*, *admissible configuration*, and *active configuration* are defined as in Section 2.1. Let $\mathcal{T}$ denote the set of all tree shapes, and, for $T \in \mathcal{T}$, let $E(T)$ denote the expected number of active witness tree configurations with tree shape T. Let $v_1, \ldots, v_m$ be the m internal nodes of T. Furthermore, for a collection K of configurations for the nodes $v_1, \ldots, v_{i-1}$, let $E_{\text{coll}}(v_i, K)$ denote the expected number of active configurations for v_i under the assumption that K is active, and let $E_{\text{coll}}(v_i)$ denote the maximum over all configurations K of $E_{\text{coll}}(v_i, K)$.

Lemma 2.6 $E_{\text{coll}}(v_i) \leq \log n / c!$.

Proof. The proof is identical to that of Lemma 2.2, since the pruned witness constructed here fulfills Properties 1 and 2 as stated in Section 2.1. ■

Let $w_1, \ldots, w_{m'}$ denote the m' pruning nodes of T, and let u_i and u'_i denote the conflicting nodes associated with w_i. For a collection K of configurations for the nodes $v_1, \ldots, v_m$ and $w_1, \ldots, w_{i-1}$, let $E_{\text{prune}}(w_i, K)$ denote the expected number of active configurations for w_i under the assumption that K is active. Furthermore, let $E_{\text{coll}}(w_i)$ denote the maximum over all configurations K of $E_{\text{coll}}(v_i, K)$.

Lemma 2.7 $E_{\text{prune}}(w_i) \leq 2^{5c+3} \cdot (\log n + 1)/\sqrt{n}$.

Proof. The pruned witness tree described above fulfills Properties 3, 4 and 5 stated in Section 2.1. Hence, the proof of Lemma 2.3, which is based only on these three properties, holds also for this lemma. ■

The probability that the congestion exceeds $4c$ is at most the probability that a pruned witness tree exists. The latter probability is at most

$$\begin{aligned}
\sum_{T \in \mathcal{T}} E(T) &\leq \sum_{T \in \mathcal{T}} n \cdot \prod_{i=1}^{m} E_{\text{coll}}(v_i) \prod_{j=1}^{m'} E_{\text{prune}}(w_j) \\
&\leq n \cdot \sum_{T \in \mathcal{T}} \left(\frac{\log n}{c!} \right)^m \cdot \left(\frac{2^{5c+3} \cdot (\log n + 1)}{\sqrt{n}} \right)^{m'} \\
&\overset{(1)}{\leq} n \cdot \sum_{T \in \mathcal{T}} \left(\frac{2^{5c+3} \cdot (\log n + 1)}{\sqrt{n}} \right)^{\kappa} \\
&\overset{(2)}{\leq} n \cdot c^{5\kappa} \cdot 2^{3c\kappa} \cdot \left(\frac{2^{5c+3} \cdot (\log n + 1)}{\sqrt{n}} \right)^{\kappa} \\
&\leq n^{-c/6+1+o(1)} ,
\end{aligned}$$

where $\kappa = \lceil \frac{c}{3} \rceil = \Theta(\log \log n)$.

Equation 1 follows from the relationship between m and m': Each of the c children of the root of the full witness tree is a root of a subtree with $2^{c-1} - 1$ internal nodes. For each of the m' pruning nodes, nodes from at most 3 of these subtrees are removed. Thus, at least $c - 3m'$ of the subtrees remain untouched. As a consequence,

$$m \geq (c - 3m') \cdot (2^{c-1} - 1) \geq (\kappa - m') \cdot (2^{c-1} - 1) \ .$$

Applying this equation and substituting $c = \lceil \log \log n \rceil$ yields

$$\left(\frac{\log n}{c!} \right)^m \leq 2^{-(2^{c-1}-1) \cdot (\kappa - m')} \leq \left(\frac{2^{5c+3} \cdot (\log n + 1)}{\sqrt{n}} \right)^{\kappa - m'} ,$$

for sufficiently large n.

Equation 2 results from a bound on the number of different tree shapes. In particular, there are at most

$$\sum_{j=0}^{\kappa} \binom{c \cdot 2^{c-1}}{j} \leq 2 \cdot c^{\kappa} \cdot 2^{(c-1) \cdot \kappa}$$

possible ways of chooosing the at most κ pruning nodes from the at most $c \cdot 2^{c-1}$ internal nodes of the witness tree. Furthermore, there are at most

$$\left(c^2 \cdot 2^{c-1} \right)^{2m'} \leq c^{4\kappa} \cdot 2^{(c-1) \cdot 2\kappa}$$

possibilities to choose the $2m'$ conflicting nodes from the at most $c^2 \cdot 2^{c-1}$ nodes of the full witness tree. Multiplying the bounds yields that the total number of different tree shapes is at most $c^{5\kappa} \cdot 2^{3c\kappa}$.

This completes the proof of Theorem 2.5. ■

3 A proposal for a data server

We present an application of our techniques to the data server architecture proposed in the introduction. For each input node i, let o_i be the object requested by the user at input node i of the randomly-wired butterfly RB_n. We assume that $o_i \neq o_j$ for $i \neq j$. Each object is stored on two disks: the first disk is chosen uniformly and randomly from the first $n/2$ disks, while the second disk is chosen uniformly and randomly from the last $n/2$ disks. We call the two instances of object o_i the *copies of* o_i. For an object o_i, let $d_1(o_i)$ and $d_2(o_i)$ be the disks storing the copies of o_i. As in Section 2, we define two paths p and p' starting at input node i: p connects input node i with output node $d_1(o_i)$, and p' connects i with $d_2(o_i)$. Since the copies of object o_i are located in different sub-butterflies, p and p' are edge disjoint paths. Unlike Section 2, we must minimize not only congestion, but also the contention at the output nodes, i.e., the maximum number of requests any disk has to serve.

3.1 Static routing

For the static selection of paths we use a modified version of the collision protocol of Section 2. Initially, all paths are *active* and not *selected*. For a path p connecting input node i and output node $d_k(o_i)$, $k \in \{1, 2\}$, let $\Delta(p)$ be the *destination* of p. A path p is selected if for each edge $e \in p$ the number of active paths plus the number of selected paths traversing e is at most c, *and* the number of active paths plus the number of selected paths with destination $\Delta(p)$ is at most $\bar{c}$. If p and its buddy p' are both eligible to be selected, one is chosen arbitrarily. A path p ceases to be active in a round if p is selected or the buddy of p, p', is selected in that round. The algorithm terminates when there are no more active paths.

Theorem 3.1 *For any $\bar{c} \geq 5$ and $c! \geq 2\log n$, the probability that the collision algorithm on RB_n takes more than $t = \log_{\bar{c}} \log(n/\log n)$ rounds to select a path for every request is at most $n^{-\bar{c}/2+1+o(1)}$.*

Proof. The proof is similar to that of Theorem 2.1 in Section 2.1.

Constructing a witness tree. For each input node i fix its requested object o_i. Fix the random permutations π_0 and π_1 used to define the randomly-wired butterfly RB_n, and fix the random disks $d_1(o_i)$ and $d_2(o_i)$, for $i = 0, \ldots, n-1$. For each request, this determines two paths. We say that two paths p and $\hat{p}$ *edge-collide*, if p and $\hat{p}$ both traverse an edge e. They are said to *disk-collide* if p and $\hat{p}$ have the same destination node on the output level. Two paths that either edge- or disk-collide are said to simply *collide*.

Assume that there is a request with paths p and p', and neither path has been selected by round t, where the value of t is to be determined later. Then p either edge-collided with c other paths $p_1, \ldots, p_c$ in round t or p disk-collided with $\bar{c}$ other paths $p_1, \ldots, p_{\bar{c}}$ in round t. If p is involved in an edge-collision (resp., disk-collision), the root of the witness tree is the request corresponding to p and the requests corresponding to $p_1, \ldots, p_c$ (resp., $p_1, \ldots, p_{\bar{c}}$) are the children. Now $p_1, \ldots, p_c$ (resp., $p_1, \ldots, p_{\bar{c}}$) and their buddies $p'_1, \ldots, p'_c$ (resp., $p'_1, \ldots, p'_{\bar{c}}$) must have been active in round $t-1$. Applying the same argument recursively to $p'_1, \ldots, p'_c$ (resp., $p'_1, \ldots, p'_{\bar{c}}$) we can construct a tree of height t. This tree is called *witness tree*.

Each node in the witness tree is a request with two paths, a down path and an up path. Some nodes in the tree corresponding to disk collisions have degree $\bar{c}$, while others corresponding to edge collisions have degree $c \geq \bar{c}$. The rightmost $c - \bar{c}$ children of a node representing an edge collision are called *superfluous nodes*. In order to bound the number of nodes in the witness tree, all subtrees rooted at a child of a superfluous node are removed. (We will *not* refer to this as "pruning' in the sequel.) Note that a superfluous node does not represent a collision.

Pruning the witness tree. As in Section 2.1, the nodes of a witness tree do not necessarily correspond to distinct requests. However, the situation here is less complex because there are no randomly-flipped switches that could be shared by different paths. Thus, it is sufficient to ensure that the requests in the pruned witness tree are distinct.

The pruning is done by a breadth-first traversal of the witness tree. Let $B(v)$ and $\Gamma(v)$ be defined as in Section 2. When a node v is visited, we use the following pruning rules:

1. If a path associated with one of v's non-pruned children is also associated with a node u in $\Gamma(v)$, then the subtrees rooted at the children of v are removed from the tree, and the subtrees rooted at the children of u are also removed from the tree. The node v is called a *pruning node*. The node u is denoted the *conflicting node* of v.

2. If a path associated with one of v's non-pruned children is associated with a node u from $B(v)$ then the subtrees rooted at the children of v are removed from the tree. The node v is called a *pruning node*. The node u is denoted the *conflicting node* of v.

We continue the pruning process till either there are no more nodes to visit or there are $\kappa = \lceil \frac{\bar{c}}{2} \rceil$ pruning nodes. In the latter case, we apply a final pruning. If v is the κth pruning node, we remove from the tree all nodes not included in $B(v) \cup \Gamma(v)$. This effectively stops the pruning process at the κth pruning node. The remaining tree is called the *pruned witness tree*.

Let $v_1, \ldots, v_m$ be the m internal nodes and let $w_1, \ldots, w_{m'}$ be the m' pruning nodes in the order of visitation. Further, let u_i denote the conflicting node of w_i, for $1 \leq i \leq m'$. The pruned witness tree possesses the following properties:

1. Any internal node v represents a collision of the down path of v with the up paths of the children of v. The down path of a pruning node w collides with a path p that is associated with its conflicting node u. The path p is called the *conflicting path* of w.

2. All nodes of the tree correspond to different requests. In particular, pruning node w_i does not represent the same request as a conflicting node u_j, $1 \leq j \leq i$.

Bounding the probability of occurrence of a pruned witness tree. We define the *tree shape* to be a description of the topology of the pruned tree including the degree (c or $\bar{c}$) of the inner nodes, the pruning and the conflicting nodes. An *admissible witness tree configuration* is a tree shape with associated requests, up and down paths, and conflicting paths, which eventually, i.e. for some setting of the random permutations defining the RB_n and the random choices for the $d_1(o_i)$, $d_2(o_i)$, $0 \leq i \leq n-1$, matches a pruned witness tree. In particular each admissible witness tree configuration has to fulfill the two properties stated above. An admissible configuration is *active* if the outcome of the random choices corresponds to all paths in the configuration.

The set of tree shapes corresponding to at least one admissible witness tree configuration is denoted by $\mathcal{T}$. As in Section 2.1, we bound the expected number of active witness tree configurations $E(T)$, for an arbitrary $T \in \mathcal{T}$. Let E_{coll} and E_{prune} be as defined in Section 2.1.

Lemma 3.2 $E_{\text{coll}}(v_i) \leq \max\{\frac{\log n}{c!}, \frac{2^{\bar{c}}}{\bar{c}!}\}$.

Proof. We first bound the expected number of active configurations for v_i representing an edge collision. In this case v_i has c children. Fix the random permutation π_0 and π_1 used to define the randomly-wired butterfly.

The expected number of active down paths for v_i is at most one. Given path p, there are $d = \log n$ possibilities to chose an edge e at which p collides with the up paths $p_1, \ldots, p_c$ of the children of v_i. Let ℓ be the level of e. Since π_0 and π_1 are fixed, there are at most $\binom{2^{\ell-1}}{c}$ possibilities to chose c paths possibly attaining e. Depending on the random choices of the destinations each such path attains e with probability $2^{-(\ell-1)}$. Thus, the expected number of active configurations for v_i is

$$d \cdot \binom{2^{\ell-1}}{c} \cdot 2^{-c(\ell-1)} \leq \frac{d}{c!}$$

Similarly, the expected number of active configurations for v_i representing a disk collision (v_i has $\bar{c}$ children) is bounded by

$$\binom{n}{\bar{c}} \cdot \left(\frac{2}{n}\right)^{\bar{c}} \leq \frac{2^{\bar{c}}}{\bar{c}!},$$

since there are n paths possibly having the same destination as the down path of v_i and each such path actually has this destination with probability $2/n$. ■

Lemma 3.3 $E_{\text{prune}}(w_i) \leq d/n$.

Proof. Let u_i be the conflicting node of w_i. Assume that w_i represents an edge collision. Since u_i is associated with 2 paths, there are $2d$ possibilities to choose the edge e on which the collision takes place. We distinguish two cases. If the up path p of w_i uses a cross edge in level 0, then its buddy p' starts by using a straight edge in level 0. Thus the level 1 node attained by p' is a random node. If p uses a straight edge in level 0, then p' uses a cross edge in level 0, and thus attains a random node in level 1.

In the levels subsequent to level 1, the course of the down path p' of w_i depends only on the random choice of its destination $\Delta(p')$, hence at every level p' attains a random edge and thus the probability for p' to collide with edge e is $1/(2n)$. As a consequence the expected number of active configurations at pruning node w_i is at most $2d \cdot \frac{1}{2n}$

Now, assume that w_i represents a disk collision. Then let $\Delta(p)$ be the destination of the down path p of w_i. The probability for the paths associated to u_i to have destination $\Delta(p)$ is $1/n$. Thus the expected number of active configurations at w_i is at most $2/n \leq d/n$, if $d \geq 2$.

■

As in Section 2.1 we proceed by bounding the expected number of active witness tree configurations.

$$\begin{aligned}
\sum_{T\in\mathcal{T}} E(T) &\leq \sum_{T\in\mathcal{T}} n \cdot \prod_{i=1}^{m} E_{\mathrm{coll}}(v_i) \cdot \prod_{j=1}^{m'} E_{\mathrm{prune}}(w_i) \\
&\leq n \cdot \sum_{T\in\mathcal{T}} \max\left\{\frac{\log n}{c!}, \frac{2^{\bar{c}}}{\bar{c}!}\right\}^{m} \left(\frac{d}{n}\right)^{m'} \\
&\overset{(1)}{\leq} n \cdot \sum_{T\in\mathcal{T}} \left(\frac{d}{n}\right)^{\kappa} \\
&\overset{(2)}{\leq} n \cdot \bar{c}^{2\kappa t} c\, 2^{\left(\bar{c}^{t}\right)} \cdot \left(\frac{d}{n}\right)^{\kappa} \\
&\overset{(3)}{\leq} n^{-\bar{c}/2+1+\mathrm{o}(1)} .
\end{aligned}$$

Equation 1 follows from a bound relating m and m'. Equation 2 is obtained by bounding the number of trees in $\mathcal{T}$, taking into account the fact each internal tree node can either represent an edge collision or a disk collision. Both of these bounds are derived in a fashion similar to their counterparts in Section 2.1. Equation 3 follows from the fact that $t = \log_{\bar{c}} \log(n/d)$ and $\max\{\frac{\log n}{c!}, \frac{2^{\bar{c}}}{\bar{c}!}\} \leq 1/2$.

■

3.2 Dynamic routing

Model description. The model is similar to the adversary model used in Section 2.2. An oblivious adversary constructs an infinite sequence of events, where each event is either an input requesting an object or an input releasing an object. At any given time each object is accessed by at most one input, and each input accesses at most one object.

The minimum algorithm. Let a request (i, o) arrive at time t, where i is an input node and o is the object requested by i. For an edge e, define $c(e)$ to be the number of paths that traverse e at time t, and $\hat{c}(i)$ to be the number of paths with destination i at time t. (We leave the t implicit as the meaning will be clear.) The congestion $c(p)$ of a path p is defined to be $\max\{\max_{e\in p}(c(e)), \hat{c}(\Delta(p))\}$, where $\Delta(p)$ is the destination of p. The algorithm examines the two paths p and p' that connect input node i with the two output nodes $d_1(o)$ and $d_2(o)$ that store object o. The request is fulfilled by p if $c(p) \leq c(p')$, otherwise the request is fulfilled by p'.

Theorem 3.4 *At any time t, the probability that the congestion exceeds* $\Theta(\log\log n)$ *is at most* $n^{-\Theta(\log\log n)}$.

Proof. As in Section 3.1, we construct a witness tree obeying the modifications made in Section 2.2 to the witness tree construction of Section 2.1. We prune the tree using the rules in Section 3.1 (modified as in Section 2.2) using at most $\kappa = \lceil c/3 \rceil$ pruning nodes. The proofs of the following lemmas are similar to the proofs of the corresponding lemmas in Section 3.1.

Lemma 3.5 $E_{\mathrm{coll}}(v_i) \leq \log n/c!$.

Lemma 3.6 $E_{\mathrm{prune}}(w_i) \leq d \cdot \frac{1}{n}$.

Finally, we bound the probability that the congestion exceeds $4c$, where $c = \Theta(\log\log n)$, by bounding the probability that a pruned witness tree exists.

$$\begin{aligned}
\sum_{T\in\mathcal{T}} E(T) &\leq \sum_{T\in\mathcal{T}} n \cdot \prod_{i=1}^{m} E_{\mathrm{coll}}(v_i) \cdot \prod_{j=1}^{m'} E_{\mathrm{prune}}(w_i) \\
&\leq n \cdot \sum_{T\in\mathcal{T}} \left(\frac{\log n}{c!}\right)^{m} \cdot \left(\frac{d}{n}\right)^{m'} \\
&\overset{(1)}{\leq} n \cdot \sum_{T\in\mathcal{T}} \left(\frac{d}{n}\right)^{\kappa} \\
&\overset{(2)}{\leq} n \cdot c^{4\kappa} \cdot 2^{2c\kappa} \cdot \left(\frac{d}{n}\right)^{\kappa} \\
&\leq n^{-c/3+1+\mathrm{o}(1)} ,
\end{aligned}$$

where Equations 1 and 2 are justified as in Section 2.2. ■

References

[1] R. Aleliunas. Randomized parallel communication. In *Proceedings of the ACM SIGACT–SIGOPS Symposium on Principles of Distributed Computing*, pages 60–72, August 1982.

[2] S. Arora, T. Leighton, and B. Maggs. On-line algorithms for path selection in a non-blocking network. *SIAM Journal on Computing*, 25(3), pages 600–625, June 1996.

[3] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-line machine scheduling with applications to load balancing and virtual circuit routing. In Proc. 25th Annual ACM Symposium on Theory of Computing, pages 240–249, November 1994.

[4] Y. Azar, A. Broder, A. Karlin, and E. Upfal. Balanced allocations. In *Proceedings of the 26th Annual ACM Symposium on the Theory of Computing*, pages 593–602, May 1994.

[5] L. A. Bassalygo and M. S. Pinsker. Complexity of an optimum nonblocking switching network without reconnections. *Problems of Information Transmission*, 9:64–66, 1974.

[6] K. Batcher. Sorting networks and their applications. In *Proceedings of the AFIPS Spring Joint Computing Conference*, volume 32, pages 307–314, 1968.

[7] Butterfly$^{\mathrm{TM}}$ Parallel Processor Overview. BBN Report No. 6148, Version 1, Cambridge, MA, March 1986.

[8] B. Beizer. The analysis and synthesis of signal switching networks. In *Proceedings of the Symposium on Mathematical Theory of Automata*, pages 563–576, Brooklyn, NY, 1962. Brooklyn Polytechnic Institute.

[9] V. E. Beneš. Optimal rearrangeable multistage connecting networks. *Bell System Technical Journal*, 43:1641–1656, July 1964.

[10] R. Cole, B. Maggs, and R. Sitaraman. On the benefit of supporting virtual channels in wormhole routers. In *Proceedings of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 131–141, June 1996.

[11] R. Cypher, F. Meyer auf der Heide, C. Scheideler, and B. Vöcking. Universal algorithms for store-and-forward and wormhole routing. In *Proceedings of the 28th Annual ACM Symposium on the Theory of Computing*, May 1996.

[12] M. Dietzfelbinger and F. Meyer auf der Heide. Simple, efficient shared memory simulations. In *Proceedings of the 5th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 110–119, June 1993.

[13] S. Felperin, P. Raghavan, E. Upfal. A theory of wormhole routing in parallel computers. In *Proceedings 33rd IEEE Symposium on Foundations of Computer Science*, pages 563-572, 1992.

[14] J.A. Garay, and I. Gopal. Call preemption in communication networks. In Proc. INFOCOM '92, Vol 44, pages 1043–1050, Florence, Italy, 1992.

[15] A. Gottlieb. An overview of the NYU Ultracomputer Project. In J. J. Dongarra, editor, *Experimental Parallel Computing Architectures*, pages 25–95. Elsevier Science Publishers, B. V., Amsterdam, The Netherlands, 1987.

[16] Ikeno. A limit on crosspoint number. *IRE Trans. on Info. Theory.* 5, pages 187–196, 1959.

[17] R. Karp, M. Luby, and F. Meyer auf der Heide. Efficient PRAM simulation on a distributed memory machine. *Algorithmica* 16, pages 245–281, 1996.

[18] R. R. Koch. Increasing the size of a network by a constant factor can increase performance by more than a constant factor. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, pages 221–230. IEEE Computer Society Press, October 1988.

[19] C. P. Kruskal and M. Snir. The performance of multistage interconnection networks for multiprocessors. *IEEE Transactions on Computers*, C–32(12):1091–1098, December 1983.

[20] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays • Trees • Hypercubes*. Morgan Kaufmann, San Mateo, CA, 1992.

[21] T. Leighton. Methods for message routing in parallel machines. In *Proceedings of the 24th Annual ACM Symposium on the Theory of Computing*, pages 77–96, May 1992.

[22] F. T. Leighton and B. M. Maggs. Fast algorithms for routing around faults in multibutterflies and randomly-wired splitter networks. *IEEE Transactions on Computers*, C–41(5):578–587, May, 1992.

[23] F. T. Leighton, B. M. Maggs, A. G. Ranade, and S. B. Rao. Randomized routing and sorting on fixed-connection networks. *Journal of Algorithms*, 17(1):157–205, July 1994.

[24] B. M. Maggs and R. K. Sitaraman. Simple algorithms for routing on butterfly networks with bounded queues. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pages 150–161, May 1992.

[25] F. Meyer auf der Heide and B. Vöcking. A packet routing protocol for arbitrary networks. In *Proceedings of the 12th Symposium on Theoretical Aspects of Computer Science*, pages 291–302, March 1995.

[26] M. Mitzenmacher, "Load Balancing and Density Dependent Jump Markov Processes", *Proc. of the* 37^{th} *IEEE Symp. on Foundations of Computer Science*, pages 213–222, 1996.

[27] F. Meyer auf der Heide, C. Scheideler, and V. Stemann. Exploiting storage redundancy to speed up randomized shared memory simulations. In *Theoret. Comput. Sci.* 162, pages 245–281, March 1996.

[28] T. Nakata, S. Matsushita, N. Tanabe, N. Kajihara, H. Onozuka, Y. Asano, and N. Koike. Parallel programming on Cenju: A multiprocessor system for modular circuit simulation. *NEC Research & Development*, 32(3):421–429, July 1991.

[29] G. F. Pfister, W. C. Brantley, D. A. George, S. L. Harvey, W. J. Kleinfelder, K. P. McAuliffe, E. A. Melton, V. A. Norton, and J. Weiss. An introduction to the IBM Research Parallel Processor Prototype (RP3). In J. J. Dongarra, editor, *Experimental Parallel Computing Architectures*, pages 123–140. Elsevier Science Publishers, B. V., Amsterdam, The Netherlands, 1987.

[30] D. Nassimi and S. Sahni. Parallel algorithms to set up the Beneš permutation network. *IEEE Transactions on Computers*, C–31(2):148–154, Feb, 1982.

[31] N. Pippenger. Parallel communication with limited buffers. In *Proceedings of the 25th Annual Symposium on Foundations of Computer Science*, pages 127–136. IEEE Computer Society Press, October 1984.

[32] N. Pippenger. Self-Routing Superconcentrators. *Journal of Computer and System Sciences*, 52(1):53–60, Feb, 1996.

[33] S. Plotkin. Competitive routing in ATM networks. *IEEE Journal on Selected Areas in Communication*, pages 1128–1136, August 1995.

[34] A. G. Ranade. Constrained randomization for parallel communication. Technical Report YALEU/DCS/TR-511, Department of Computer Science, Yale University, New Haven, CT, 1987.

[35] A. G. Ranade. How to emulate shared memory. *J. Comp. Syst. Scis. 42*, pages 307-326, 1991.

[36] A. Ranade, S. Schleimer, and D. S. Wilkerson. Nearly tight bounds for wormhole routing. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, 1994.

[37] R. Rooholamini, V. Cherkassky, and M. Garver. Finding the right ATM switch for the market. *IEEE Computer*, pages 16–28, April 1994.

[38] J. Turner and N. Yamanaka. Architectural Choices in Large Scale ATM Switches. Technical Report WUCS–97–21. Department of Computer Science, Washington University St. Louis, MO. 1997.

[39] E. Upfal. Efficient schemes for parallel communication. *Journal of the ACM*, 31(3):507–517, July 1984.

[40] E. Upfal. An $O(\log N)$ deterministic packet routing scheme. *Journal of the ACM*, 39(1), pages 55–70, Jan 1992.

[41] L. G. Valiant. A scheme for fast parallel communication. *SIAM Journal on Computing*, 11(2):350–361, May 1982.

[42] L. G. Valiant and G. J. Brebner. Universal schemes for parallel communication. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing*, pages 263–277, May 1981.

[43] A. Waksman. A permutation network. *Journal of the ACM*, 15(1):159–163, January 1968.