

# On the Theory of Interconnection Networks for Parallel Computers <sup>\*</sup>

Eli Upfal

Department of Applied Mathematics\*\*  
The Weizmann Institute of Science  
Rehovot, Israel  
and  
IBM Almaden Research Center  
San Jose, California

**Abstract.** Efficient data transfer between processors is an essential component in any large scale parallel computation. Motivated by the growing interest in parallel computers, a significant amount of theoretical research has been devoted to the area of interconnection networks for parallel computers, most of it to the *packet routing* (or *store-and-forward*) model of communication. We survey some of the major developments in this field, and discuss several new alternative models of communication, such as *wormhole routing*, *virtual cut-through routing*, and *hot-potato routing*.

## 1 Introduction

Information exchange between processors is essential for any efficient parallel or distributed computation. In many large scale applications, communication time dominates the execution time of the whole parallel computation. Thus, the performance of a large scale parallel computer is highly correlated with the efficiency of its communication network and communication algorithm. Indeed, most parallel machines devote a significant portion of their resources to support communication between processors, or between processors and memory modules.

A communication solution or *communication scheme* has of two components:

- A communication network, specifying the connection between the various components (processors, memory modules, and routing switches);
- A communication (or routing) algorithm, computing the route that each message takes on the network, and resolving scheduling conflicts.

What are the main requirements from an efficient communication scheme for a parallel computer?

---

<sup>\*</sup> Invited paper - ICALP 1994.

<sup>\*\*</sup> Work at the Weizmann Institute supported in part by the Norman D. Cohen Professorial Chair of Computer Science.

- **Throughput:** To support parallel computation, the communication scheme must be able to simultaneously transmit a large number of messages in minimum number of parallel steps. The pattern of the communication is usually not known in advance, thus the scheme should be able to adapt to various communication patterns. (However, in most applications the communication patterns are fairly balanced, no processor is a source or destination of too many messages (no *hot spots*).
- **Sparse Topology:** Supporting fast simultaneous communication of a large number of messages is relatively easy if each processor is directly connected to any other processor in the system. Unfortunately, all communication technologies available today put a severe limit on the number of communication channels that can be connected to a single component, and to the total number of channels in the system. Thus, in any realistic system, each processor can be connected directly to a small number of other processors, and most communication must proceed through intermediate components.
- **Scalable and Uniform Topology:** It is highly desirable that the topology of a communication network will be uniform, symmetric, and in particular, scalable.
- **Small Layout:** The total area of chips and boards that make up the network, and the total volume of wires connecting the boards should be minimal.
- **Local Control:** In a large scale parallel system it is inefficient to collect, every communication round, information from all the processors about their communication needs, compute a global routing solution, and distribute the solution to all the individual processors and switching nodes. Instead, routing has to be done locally, using the limited information available to each processor about the global communication task.
- **Fault Tolerance:** It is unrealistic in a large system to stop the computation whenever one component fails. Thus, the communication scheme must be able to tolerate a certain number of faulty nodes and channels without significant degradation in the overall performance.

Most of the theoretical work on interconnection networks has dwelt on a communication method termed *packet routing*. Indeed, in the past packet routing had been the method of choice for virtually all large scale systems. The first part of this paper gives a survey of some of the major theoretical results related to packet routing.

Following progress in hardware and system design, there is a growing trend in multicomputer architecture to experiment with new routing techniques such as *wormhole routing*, *virtual cut-through routing*, and *deflection routing*. This trend has motivated some recent theoretical work. The second part of this paper discusses these new models, reviews several of the recent results, and suggests some intriguing open problems.

In this short paper we can only cover a few of the many interesting results in this research area. The reader is referred to the excellent book of Leighton [23] for an in-depth treatment of this area, to a survey paper by Dally [12] for a more practical point of view, to a survey paper by Pippenger [37] for a discussion

of *circuit switching* which will not be covered here, and to survey papers by Leighton [25], and by Valiant [44].

## 2 Preliminaries

We model a communication network by a graph  $G = (V, E)$ . The vertices of the graph represent processors, memory modules, or routing switches, and the edges (directed or undirected) represent communication channels. We distinguish between two types of networks for parallel computers. In a *direct* network each node consists of a processor, a memory module, and switching device, thus each node is both a computing component and a switching node in the network. In an *indirect* (multistage) network, processors and memory modules are connected only to the input and output nodes (in the first and last stage of the network), internal nodes serve only as routing switches.

Most of the theoretical research has been focused on three type of networks: meshes, hypercubes and butterflies (in section 3.5 we discuss a fourth type of network, the *multibutterfly* that combines the butterfly topology with expander graphs.)

A *k-dimensional mesh (or array)* has  $N$  nodes  $\{(x_1, \dots, x_k) \mid 1 \leq x_i \leq N^{1/k}\}$ . Two nodes are connected iff they differ by precisely 1, in exactly 1 coordinate. An *N-node hypercube* is an  $\log_2 N$ -dimensional mesh. Mesh and hypercube networks are usually used as direct networks.

An *N-input/output, degree d butterfly* is a graph with  $\log_d N + 1$  stages, each with  $N$  nodes. Number the stages  $0, \dots, s = \log_d N$ , and the nodes in each stage by  $0, \dots, N$ . Nodes in stage 0 are input nodes, and nodes in stage  $s$  are output nodes. A location in the network is characterized by a pair  $(\ell, x)$ , where  $\ell$  is the stage number, and  $x$  is the node number in the stage. Let  $(x_0, \dots, x_s)$  denote the base  $d$  representation of the number  $x$ . A node  $(\ell, x)$  is connected to  $d$  nodes in stage  $\ell + 1$ . The numbers of these  $d$  nodes are  $(x_0, \dots, x_{\ell-1}, *, x_{\ell+1}, \dots, x_s)$ , where  $*$  =  $0, \dots, d - 1$ . This topology is used for an indirect network. A *wrapped butterfly*, in which the first and the last stages are merged so that node  $(0, x)$  and node  $(s, x)$  become one node, is used as a direct network.

The wrapped butterfly topology is closely related to the hypercube topology. In particular, if we identify as one node each of the  $\log_2 N$  nodes  $(\ell, x)$ ,  $\ell = 1, \dots, \log_2 N$  in a degree 2 wrapped butterfly, we get an  $N$  node hypercube.

Most of the theoretical research focused on the combinatorial problem of routing an  $h$ -relation. Each node (or an input in the case of an indirect network) is a source of up to  $h$ -messages, and each node (or output) is a destination of up to  $h$ -messages. Of particular interest is the case of routing a 1-relation or a permutation.

## 3 Packet Routing

In the *packet routing (or store and forward)* model, messages are conveyed as packets. An entire packet can reside at a routing switch, and a packet is sent

from a queue of one switch to the queue of another switch until it reaches its destination. The entire packet is transmitted from node  $v$  to node  $u$  before any part of the packet can proceed from node  $u$  to node  $w$ . A step is defined as the time it takes one packet to cross one edge, and at most one packet can cross an edge in each step. Theoreticians distinguish between networks in which the size of the queue of each node or edge is bounded, and networks in which the size of the queues is a function of the network size. Another distinction is between networks in which a processor can activate all its outgoing channels simultaneously (multi-port model), and networks in which a processor can activate only one outgoing channel in each step (single-port model). For simplicity, we consider only the multi-port model in this survey (see [8] for results comparing the two models).

The goal is to construct routing algorithms that routes an arbitrary permutation in a minimum number of parallel steps. An obvious lower bound is the diameter of the network, since a packet can traverse at most one edge at each step. Indeed, we are interested in  $O(\log n)$  routing algorithms for the  $N$ -node hypercube and for the  $N$ -input/output butterfly, and in  $O(N^{1/d})$  routing algorithms for the  $d$ -dimensional mesh.

### 3.1 Oblivious Routing

Most parallel machines use some form of simple greedy routing algorithm. Messages follow a shortest path to their destinations, and the path is chosen locally with almost no global information. While these algorithms perform very well in most practice applications, it is easy to construct worst case permutations for which greedy algorithms perform poorly on the mesh, the hypercube or the butterfly network. For example, an  $N$ -packet transpose permutation sends the packet at input  $(x_1, x_2, \dots, x_s)$  (where  $s = \log_d N$ ) to output  $(x_{(s/2)+1}, x_{(s/2)+2}, \dots, x_s, x_1, \dots, x_{s/2})$ . It is not hard to verify that routing an  $N$ -packet transpose permutation on an  $N$ -input/output butterfly network, using a shortest path, greedy algorithm takes  $\Omega(\sqrt{N})$  steps, while the diameter of the butterfly is only  $\log N$ .

Since simple greedy algorithms perform poorly in the worst case, a challenging question is whether there is a routing scheme that is simple to implement, uses only local information, and performs well on any permutation. A lower bound of Borodin and Hopcroft [6] shows that if we restrict the discussion to deterministic algorithms, then an algorithm that performs well in the worst case can not be too simple. Borodin and Hopcroft defined the class of *oblivious* routing algorithms: a routing algorithm is oblivious if the path of each packet does not depend on the origin or destination of any other packet in the system. They show that any *deterministic* oblivious algorithm for routing on a low degree network performs poorly in the worst case. The following theorem [21] is a slight improvement of the original lower bound in [6]:

**Theorem 1.** *Let  $G$  be an  $N$  node network with maximum degree  $d$ , and let  $\mathcal{A}$  be an oblivious routing algorithm for  $G$ . There is an  $N$ -packet permutation  $\pi(N)$ , such that algorithm  $\mathcal{A}$  routes  $\pi(N)$  on  $G$  in  $\Omega(\sqrt{N}/d)$  parallel steps.*

A recent result [8] has shown that the above bound is optimal, i.e. for any  $N$  and  $d$ , there is an  $N$  node, degree  $d$  network, and a deterministic oblivious algorithm that routes any permutation on that network in  $O(\sqrt{N}/d)$  parallel steps. An oblivious algorithm that matches this bound for the hypercube network has been given in [21].

### 3.2 Randomized Routing

We mentioned before that simple greedy algorithms perform well in most practical applications. These empirical observations are supported by the following result:

**Theorem 2.** *Let  $\pi(N)$  be a permutation chosen uniformly at random from all permutations on  $N$  elements. Routing  $\pi(N)$  on an  $N$ -input/output butterfly network using a unique shortest path for each packet requires, with high probability,  $O(\log N)$  parallel steps.*

Similar results hold for the hypercube and the mesh networks (but since a shortest path is not unique in these networks, the greedy routing algorithm needs to be properly defined).

Valiant [43, 45] was the first to observe that since any communication request can be represented as a concatenation of two random requests, two applications of a routing algorithm that performs well on average, yields a randomized algorithm that performs well on any permutation. Since in practice it is complicated to generate a random permutation, Valiant's result uses the following variant of Theorem 2:

**Theorem 3.** *Assume that each input of an  $N$ -input/output butterfly generates one packet, and sends it to a randomly chosen output of the butterfly. If packets use the unique shortest path, with high probability all packets arrive at their destinations in  $O(\log N)$  parallel steps.*

*Proof.* Let  $P = e_0, \dots, e_{s-1}$  denote an input to output path in the butterfly ( $s = \log_d N$ ), where  $e_i$  is an edge connecting a node in stage  $i$  to a node in stage  $i + 1$ . Let  $x_i$  denote the number of packets traversing edge  $e_i$  in routing the set of  $N$  packets. Packets from no more than  $d^i$  inputs can reach  $e_i$ . Since packets are sent to randomly chosen outputs, each of the  $d^i$  packets traverse edge  $e_i$  with probability  $\frac{1}{d^{i+1}}$ . Thus, the distribution of  $x_i$ , is stochastically bounded by  $B(d^i, \frac{1}{d^{i+1}})$  (Binomial distribution with parameters  $d^i$ , and  $1/d^{i+1}$ ). Since for all  $0 \leq i \leq s-1$ ,  $E[x_i] \leq \frac{1}{d}$ , by Hoeffding inequality [19] the distribution of  $\sum_{i=0}^{s-1} x_i$  is stochastically bounded by  $B(sN, \frac{1}{dN})$ . Applying the Chernoff inequality [9], we prove that

$$\Pr\left\{\sum_{i=0}^{s-1} x_i > C \log N\right\} \leq \frac{1}{N^2},$$

for some constant  $C$  independent of  $N$ . Since there are only  $N$  packets, with high probability no packet is delayed more than  $C \log N$  steps, or all packets reach their destinations in  $O(\log N)$  parallel steps.

**Comment:** Note that the proof of Theorem 3 does not depend on the order of the stages in the butterfly. In particular, the same analysis holds if we reverse the directions of all the edges, and send one packet from each output to a randomly chosen input.

Consider a wrapped butterfly with  $N$  nodes per stage. Assume that each node in stage 0 is a source and destination of one packet. Valiant's randomized routing algorithm has two phases, in each phase we treat the network as an indirect  $N$ -input/output butterfly network (nodes in stage 0 are both input and out nodes).

- **Phase A:** Each packet is sent to a random output, where the random output node is chosen independently and uniformly at random. The packet proceeds to its random node in  $\log N$  transitions through the shortest path on the (indirect) butterfly network.
- **Phase B:** Each packet is sent from its random location (random input node) to its final destination (output node). The packet proceeds to its destination in  $\log N$  transitions through the shortest path on the (indirect) butterfly network.

Theorem 3 gives a bound with high probability on the run-time of the first phase. Furthermore, by symmetry it also gives a bound on the run-time of the second phase. To see this consider two routing scenarios: In the first scenario each output chooses one random input, and a packet is sent from that input to the output node, this scenario corresponds to Phase B of the algorithm. In the second scenario we reverse the directions of the edges, and each “output” node sends one packet to a randomly chosen “input” node. Clearly the distribution of the number of packets traversing a given edge is identical in both scenarios. By the comment following the proof of Theorem 3, the second scenario, and thus also the first scenario or phase B of the algorithm terminates with high probability in  $O(\log N)$  parallel steps.

Using the correspondence between a wrapped butterfly with  $N$  node per stage, and the  $N$ -node hypercube (see Section 2.) we get Valiant's result [43]:

**Theorem 4.** *There is a randomized routing algorithm that with high probability routes an arbitrary permutation on an  $N$ -node hypercube in  $O(\log N)$  parallel steps.*

Valiant's seminal result has been followed by a long chain of works that used variants of the basic technique to prove further results. Valiant and Brebner [45] have shown that similar technique gives  $O(\sqrt{N})$  and  $O(N^{1/3})$  algorithms for routing a permutation on the 2-dimensional and 3-dimensional meshes. Aleliunas [3], and Upfal [41] gave  $O(\log N)$  randomized routing algorithms for routing a full permutation on bounded degree direct networks, i.e. routing a permutation of  $N \log N$  packets on a bounded degree graph (such as the wrapped butterfly or shuffle exchange) with a total of  $N \log N$  routing switches. Pippenger [36], and later Ranade [40] gave randomized, bounded buffers routing algorithms for the butterfly network. Packet routing on *fat trees* was first proposed by Leiserson

[16, 27, 28]. Valiant's type routing algorithm performs as well on a fat-tree as on the butterfly network, and the fat-tree has an optimal area layout.

### 3.3 Time-Randomness Tradeoff for Oblivious Routing

The work on oblivious packet routing may give us some insight into one of the least understood aspect of complexity theory - the contribution of randomness to computation. There are many computation problems for which the best known randomized algorithm performs substantially better than the best known deterministic algorithm. However, oblivious packet routing is one of a very few problems for which there is a proven gap between its randomized and deterministic complexity. Peleg and Upfal [35] have used this fact for a study of randomness as a resource. Their main result is an almost tight tradeoff between the amount of randomness given to an oblivious routing algorithm, and its performance.

**Theorem 5.** *Let  $e^3 \log N \leq T \leq \sqrt{N}/2d$ , and assume that the only source of randomness available to the algorithm is a sequence of independent random bits.*

1. *Any oblivious randomized routing algorithm that routes an arbitrary permutation on an  $N$ -node, degree  $d$  network in expected run-time bounded by  $T$ , must use at least  $\log(N/T) - O(1)$  independent random bits.*
2. *There exists an oblivious randomized algorithm for the  $N$ -node, degree  $d$  butterfly network that uses  $(1 + o(1)) \log(N/T)$  independent random bits, and routes an arbitrary permutation in expected run time bounded by  $T$ .*

Roughly speaking, the above theorem states that with less than  $\frac{1}{2} \log N$  random bits a randomized oblivious algorithm does not perform better than a deterministic oblivious algorithm. Any additional random bit, above the first  $\frac{1}{2} \log N$  bits, improves the run-time by a constant factor, until the run-time reaches the  $O(\log N)$  diameter bound. Similar results have been given in [35] for a general random source, measuring randomness by the entropy of the source.

### 3.4 Information Dispersal

*Information Dispersal* is an elegant probabilistic routing technique that achieves near optimal routing time, uses bounded buffers, and has high fault tolerance properties. This technique was first proposed and analyzed by Rabin [39].

The basic idea in the information dispersal routing technique is to break every packet into  $O(\log N)$  *subpackets*, and to route each of the subpackets independently. Consider the proof of Theorem 4, assume that each node is a source and destination of  $h = O(\log N)$  subpackets instead of one packet, and that each subpacket is routed through a randomly chosen two phase route. The distribution of the number of subpackets crossing a given edge is stochastically bounded by  $B(hN, 1/N)$ , thus with high probability no more than  $O(\log N)$  subpackets traverse each edge. The total run time of the algorithm is therefore bounded by  $O((\log n)^2)$  substeps, where a substep is the time it take a subpacket to traverse and edge.

The efficiency of the information dispersal system depends on two parameters: (a) The ratio between the size of a packet and the size of each subpacket; (b) The relation between a step and a substep. When a packet is partitioned into many subpackets, each subpacket must include a header with an address and routing information. Since with high probability only  $O(\log N)$  subpackets traverse each edge, the size of the queue of each edge can be bounded by the size of  $O(\log N)$  subpackets. If the original packet is large enough so that the total size of the additional  $O(\log N)$  headers is negligible, then the queue of each edge can be bounded by the size of  $O(1)$  packets, thus we can use a network with bounded edge queues. Furthermore, if the time it takes a node to transmit packets is mainly proportional to the total size of the messages and not to the different number of messages it needs to send, then  $O(\log N)$  substeps correspond to one step, and the algorithm routes an arbitrary permutation on the hypercube in  $O(\log N)$  parallel steps.

Fault tolerance is achieved through redundancy in encoding the content of the original message into the  $O(\log N)$  subpackets. Using an error correction code, the content of the original message is encoded in  $O(\log N)$  subpackets, such that the total length of all the subpackets is linear in the size of the original message, and any subset of at least half the subpackets is sufficient for (efficiently) reconstructing the content of the original message. By sending the subpackets through almost disjoint random paths, the algorithm guarantees that even if some nodes or edges become faulty, with high probability the destination node receives sufficient number of subpackets to reconstruct the original message. (See [18, 30, 38] for further use of this method.)

### 3.5 Deterministic Routing - The Multibutterfly

A different approach to packet routing has been introduced in [42]. The multibutterfly communication scheme uses a different topology and a new type of routing algorithm. The routing algorithm is deterministic, and uses bounded buffers. The algorithm routes an arbitrary permutation on a bounded degree multibutterfly topology in logarithmic time.

The topology of the multibutterfly network is based on the notion of expander graphs. Roughly speaking, an expander is a graph in which each set of vertices that is not too large has a large number of neighbors in the graph. The multibutterfly construction uses splitters which are graphs with special expansion properties.

**Definition 6.** An  $(\alpha, \beta, d, m)$ -*splitter* is a bi-partite graph  $G = (A, B, E)$  such that:

1. Each of the two sets of vertices  $A$  and  $B$  have size  $m$ ;
2. The set  $B$  is partitioned into two disjoint, equal size sets,  $B_u$  and  $B_l$  (w.l.o.g. assume that  $m$  is even);
3. The degree of each vertex in the graph is  $d$ ;
4. Any set  $X \subset A$ , such that  $|X| \leq \alpha m$ , has at least  $\beta|X|$  neighbors in each of the two sets  $B_u$  and  $B_l$ .

The set  $A$  is called the input of the splitter, the set  $B_u$  is the upper output set, and the set  $B_l$  is the lower output set.

Using the explicit construction of expander graphs in [29], for any constants  $\alpha$  and  $\beta$  such that  $\alpha\beta < 1/2$ , there exists a constant  $d$  and an explicit construction of an  $(\alpha, \beta, d, m)$ -splitter.

An  $N$ -input/output multibutterfly has  $s = \log_2 N + 1$  stages, each stage has  $N$  nodes. (We assume for simplicity of exposition that  $\log_2 N$  is an integer). Number the stages  $0, \dots, s - 1$ , and the nodes in each stage  $1, \dots, N$ . Partition the  $N$  nodes of stage  $i$  into  $2^i$  blocks,  $B_i^1, \dots, B_i^{2^i}$ , such that block  $B_i^j$  has the nodes  $(j - 1)\frac{N}{2^i} + 1, \dots, j\frac{N}{2^i}$ . To construct the multibutterfly network, use an  $(\alpha, \beta, d, 2^i)$ -splitter to connect each block  $B_i^j$  in stage  $i$  to the two blocks  $B_{i+1}^{2^j-1}$  and  $B_{i+1}^{2^j}$  in stage  $i + 1$ . This construction gives an  $N$ -input/output acyclic network with  $N \log_2 N$  internal switches, each with in and out degree  $d$ . While there are many paths from each input to each output, the sequence of splitters that a packet traverses from a given input to a given output is unique, it is the sequence of  $\log_2 N$  splitters that have a node with the same number as the packet's destination in their input set.

The routing algorithm is simple. Each internal node has a buffer of size one. A packet at a given node has  $d/2$  outgoing edges through which it can proceed to its destination. The packet is sent to the first of these  $d/2$  neighbors that have an empty buffer (see [42] for the exact protocol). Note that although the sequence of splitters that the packet traverses depends only on its destination, the exact path depends on routes of other packets in the network, thus the algorithm is not oblivious.

The high throughput of the network is due to the expansion property of the splitters. It can be shown that a set of packets at a given stage does not move forward in a given step only if it is blocked by a larger set of packets in the next stage. Thus, at any step, a large portion of the packets in the system moves forward. Formulating this intuition leads to the proof of the following theorem:

**Theorem 7.** *There is a deterministic algorithm that routes an arbitrary permutation on an  $N$ -input/output multibutterfly with bounded buffers in  $O(\log N)$  parallel steps.*

A more complicated version of the multibutterfly scheme can route an arbitrary permutation of  $N \log N$  packets on a wrapped multibutterfly with  $N \log N$  nodes [42]. Note that the only previously known  $O(\log N)$  deterministic routing schemes require the use of the AKS sorting network [2, 22].

Leighton and Maggs [26] have shown that the multibutterfly is highly fault-tolerant. The presence of  $t$  worst case faults in an  $N$ -input/output multibutterfly can affect no more than  $O(t)$  inputs and outputs. Any permutation between the remaining  $N - O(t)$  inputs, and  $N - O(t)$  outputs is routed in  $O(\log N)$  parallel steps.

## 4 Alternative Models

While packet routing is very efficient asymptotically, this efficiency is often compromised by the overhead required by the packet routing mechanism; especially in maintaining queues in the intermediate nodes, and breaking long messages into small packets and then reconstructing the original messages when the pieces reach their destination (possibly out of order). Following progress in hardware and system design there is a growing trend in multicomputer architecture to experiment with new routing techniques such as wormhole routing, virtual cut-through routing, and deflection routing. These are simple communication mechanisms that reduce overhead in the local switches, thus lead to faster communication. Designing efficient routing algorithms for each of these routing methods, and understanding the power and limitations of each method is a new challenge for the theory community. To capture the essence of these models, such a study should focus on simple algorithms that can be implemented with minimum overhead.

### 4.1 Wormhole Routing

In *wormhole routing* [4, 10, 11, 12, 34], a message is transmitted as a contiguous stream of bits, physically occupying a sequence of nodes/edges in the network. There are no queues in the intermediate nodes, and a node can only hold a (small) fraction of a message (a *flit*). The routing strategy is simple, keeping the overhead in the intermediate nodes minimal. When the head of a message reaches a node, the node reads the first few bits in the message's header, and uses them to direct the head of the message to the next node on its path. The rest of the message follows the head of the message as it moves, requiring no more intervention by the intermediate node. Thus, a message resembles a worm burrowing through the network, giving rise to the name for this style of routing. Since there are no queues, when a message is routed through the network it occupies a number of switches and edges proportional to its length. If the head of the message cannot proceed because of congestion, the message is blocking a chain of edges and nodes, thus simultaneously delaying many other messages in the network. This fact significantly complicates the analysis of wormhole routing.

Parallel machines that use variants of wormhole routing include the Intel Delta-machine, Intel iPSC/2, Intel Sigma, Symult S14, MIT J-machine, MIT April, and others. Previous research on wormhole routing focused on simulations that study the extent to which a network can be loaded (as a fraction of the available bandwidth) before it gets clogged, and has addressed the issue of *deadlock* [13]. A theoretical study of wormhole routing has been initiated by Felperin, Raghavan and Upfal [14]. They defined a model for this study, and analyzed wormhole routing algorithms based on the idea of *random oblivious delays*. A message waits at its source for a random initial delay before proceeding greedily on a canonical path to its destination. Their analysis shows that the randomness introduced to the system by the random delays significantly improves the performance of simple greedy algorithms on bounded degree networks

such as the two dimensional mesh and the butterfly. However, the proven results are not optimal, and it is an open problem whether an even simpler technique can achieve a better run-time for worst case input.

## 4.2 Virtual Cut-Through Routing

There is some confusion between wormhole routing and the model of *virtual cut-through*. The latter resembles wormhole routing superficially, but differs in one crucial aspect: in virtual cut-through, the buffers at nodes can hold a constant number of *messages*, rather than a constant number of *flits*. Thus, in virtual cut-through, the buffer size at a node is proportional to the worm length, and when the head of a message is blocked in a given node, the entire message can “fold” into the buffer of that node, rather than occupying many nodes and edges and delaying many other messages.

Leighton [24] conducted a probabilistic analysis of virtual cut-through on meshes. A nice feature of Leighton’s result is that it deals with the *greedy algorithm*, which is of great practical interest: there are no priorities assigned to messages, and each message simply keeps moving whenever it can, along the one-bend path to its destination. Worst case bounds for the mesh are given in Makedon and Simvonis [31]. Aiello et al. [1] analyzed a non-oblivious algorithm for the hypercube and other networks of logarithmic degree; however, their algorithm may time-multiplex a link over several worms, so that a worm does not flow contiguously across a link.

## 4.3 Deflection Routing

*Deflection* or *hot-potato routing* is another routing technique that aims at reducing overhead at the intermediate nodes. In deflection routing this is achieved through eliminating queues at intermediate nodes. A packet attempts to travel “towards” its destination. However, two or more packets may arrive at a node and wish to exit it on the same outgoing link. In that event the node forwards one of the packets along the preferred edge. The other packet(s) will be “deflected” away from the preferred direction, exiting the nodes along other link(s). In particular, some packets may temporarily move further away from their destinations. The analysis of deflection routing is considerably more difficult than that of packet routing. There is no a priori bound on how many links a packet would traverse until reaching its destination. A packet may be deflected to unexpected regions of the network, interfering with other packets in these regions. Nevertheless, deflection routing seems to perform very well in practice, parallel machines that use this method include the Tera computer, the HEP multiprocessor, and others.

Hajek [17] gave a  $2k + \log N$  deterministic hot-potato algorithm for routing  $k$  packets (with arbitrary destinations) on the  $N$  node hypercube. His algorithm is the first proven *livelock* free algorithm. Feige and Raghavan [15] developed and analyzed hot-potato algorithms that use random oblivious delays. Their

algorithm for the  $N$ -node hypercube routes a set of  $N$  packets with random destination in  $O(\log N)$  parallel steps. On the 2-dimensional mesh their randomized algorithm routed an arbitrary permutation in  $O(\sqrt{N})$  steps. These results have been extended in [20] to an  $O(dN^{1/d})$  randomized algorithm for routing an arbitrary permutation on the  $d$ -dimensional mesh. Bar-Noy, et al. [5] gave deterministic algorithm that route an arbitrary permutation on the 2-dimensional mesh in  $O(N^{1/2+\epsilon})$  steps, for any  $\epsilon > 0$ . They also gave the first non-trivial analysis of a hot-potato algorithm for wormhole routing. Finally, Newman and Schuster [32, 33] have explored the relation between hot-potato and sorting, and packet routing algorithms. Many of the above algorithms do not meet the simplicity requirement, thus the problem of designing efficient, and simple to implement hot-potato algorithms, in particular deterministic algorithms, for routing a permutation is still wide open.

Several researches have tried to strengthen Hajek's result for a general set of destinations, and to extend it to other networks. Borodin and Rabani [7] have shown an  $O(k) + dist$  hot-potato algorithm for routing  $k$  packets in any dimensional mesh (including the hypercube), where  $dist$  is the maximum distance of any packet from its destination. An obvious conjecture is that there exists such an algorithm for any network.

**Acknowledgment:** Many thanks to Allan Borodin and Prabhakar Raghavan for their comments on early versions of this paper.

## References

1. B. Aiello, F.T. Leighton, B. Maggs, and M. Newman. Fast algorithms for bit-serial routing on a hypercube. In *Proc. of the 2nd Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 55–64, 1990.
2. M. Ajtai, J. Komlós, and E. Szemerédi. Sorting in  $c \log n$  parallel steps. *Combinatorica*, 3(1):1–19, 1983.
3. R. Aleliunas. Randomized parallel communication. In *Proc. of the 1st Annual ACM-SIGOPS Symposium on Principles of Distributed Computing*, pages 60–72, 1982.
4. W.C. Athas. Physically compact, high performance multicomputers. In *Sixth MIT Conference on Advanced Research in VLSI*, pages 302–313, MIT Press, 1990.
5. A. Bar-Noy, P. Raghavan, B. Schieber, and H. Tamaki. Fast Deflection routing for packets and worms. In *Proc. of the 11th Annual ACM-SIGOPS Symposium on Principles of Distributed Computing*, 1993.
6. A. Borodin and J. E. Hopcroft. Routing, merging, and sorting on parallel models of computation. *Journal of Computer and System Sciences*, 30:130–145, 1985.
7. A. Borodin and Y. Rabani. Private communication, 1994.
8. A. Borodin, P. Raghavan, B. Scheiber, and E. Upfal. How much can hardware help routing? In *Proc. of the 25th ACM Symp. on Theory of Computing*, pages 573–582, 1993.
9. H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Math. Stat.*, 23:493–509, 1952.
10. W. Dally and C.L. Seitz. Deadlock free message routing in multiprocessor interconnection networks. *IEEE Trans. Computers*, 36:547–553, 1987.

11. W.J. Dally. Fine grain message passing concurrent computers. In *Third Conference on Hypercube Concurrent Computers and Applications*, pages 2–12. ACM Press, 1988.
12. W.J. Dally. Virtual channel flow control. In *Seventeenth Annual International Symposium on Computer Architecture*, pages 60–68. ACM Press, 1990.
13. J. Duato. On the design of deadlock free adaptive routing algorithms for multicomputers: theoretical aspects. In *Second European Conference on Distributed Memory Computing*, pages 234–243. Springer Verlag LNCS 487, 1991.
14. S. Felperin, P. Raghavan, and E. Upfal. A theory of wormhole routing in parallel computers. *Proceedings of the 33rd Annual IEEE Conference on Foundations of Computer Science*, 1992, pages 563–572.
15. U. Feige and P. Raghavan. Exact analysis of Hot Potato routing. In *33rd Annual Symposium on Foundations of Computer Science*, pages 553–562, 1992.
16. R. I. Greenberg and C. E. Leiserson. Randomized routing on fat-trees. *Advances in Computing Research*, 5:345–374, 1989.
17. B. Hajek. Bounds on evacuation time for deflection routing. *Distributed Computing*, 5:1–6, 1991.
18. J. Hastad, F.T. Leighton, and M. Newman. Fast computation using faulty hypercube. In *Proc. of the 21th Annual ACM Symp. on Theory of Computing*, 1989, pages 251–263.
19. W. Hoeffding. On the distribution of the number of successes in independent trails. *Ann. of Math. Stat.* 27:713–721, 1958.
20. C. Kaklamanis, D. Krizanc, and S. Rao. Improved hot potato routing for processor array. In *Proc. of the 5rd Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 273–282, 1993.
21. C. Kaklamanis, D. Krizanc, and T. Tsantilas. Tight bounds for oblivious routing in the hypercube. In *Proc. of the 3rd Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 31–36, 1991.
22. F.T. Leighton. Tight bounds on the complexity of parallel sorting. *IEEE Transactions on Computers*, C-34:344–354, 1985.
23. F.T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. Morgan Kaufmann Publishers, San Mateo, California 1992.
24. F.T. Leighton. Average case analysis of greedy routing algorithms on arrays. In *Second Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 2–10. 1990.
25. F.T. Leighton. Methods for Message Routing in Parallel Machines. In *Proc. of the 24th Annual ACM Symp. on Theory of Computing*, 1992, pages 77–96.
26. F.T. Leighton and B. Maggs. Expanders might be practical: Fast algorithms for routing around faults in multibutterflies. In *Proc. of the 30th Annual Symposium on Foundations of Computer Science*, pages 384–389, 1989.
27. C. E. Leiserson. Fat-trees: universal networks for hardware-efficient supercomputing. *IEEE Transactions on Computers*, C-34(10):892–901, 1985.
28. C. E. Leiserson. VLSI theory and parallel supercomputing. In R.F. Rashid, editor, *Carnegie Mellon University School of Computer Science 25th Ann. Symp.* Addison-Wesley, 1991. Pages 29–44.
29. A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–277, 1988.
30. Y.-D. Lyuu. Fast fault-tolerance parallel communication and on-line maintenance using information disposal. In *Second Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 378–387, 1990.

31. F. Makedon and A. Simvonis. On bit-serial packet routing for the mesh and the torus. In *Third Symposium on Frontiers of Massively Parallel Computation*, pages 294–302. IEEE Computer Society Press, 1990.
32. I. Newman and A. Schuster. Hot potato algorithms for permutation routing. Technical Report, PCL Report # 9201, CS dept., Technion, November 1992.
33. I. Newman and A. Schuster. Hot-potato worm routing via store-and forward packet routing. In *Proc. 2nd Israeli Symp. on Theory of Computing and Systems*, 1993.
34. M. Noakes and W.J. Dally. System design of the j machine. In *Sixth MIT Conference on Advanced Research in VLSI*, pages 179–194. MIT Press, 1990.
35. D. Peleg and E. Upfal. Time-randomness trade-off for oblivious routing. *SIAM Journal on Computing*, 19:256–266, 1990.
36. N. Pippenger. Parallel communication with limited buffers. In *25th Annual Symposium on Foundations of Computer Science*, pages 127–136, 1984.
37. N. Pippenger. Communication networks. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science, Vol. A: Algorithms and Complexity*. MIT Press, Cambridge, 1990, pages 805–834.
38. F. Preparata. Holographic dispersal and recovery of information. *IEEE Trans. Information Theory*, IT-35(5):1123–1124, 1989.
39. M. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the ACM*, 36(2):335–348, 1989.
40. A. Ranade. How to emulate shared memory. In *Proceedings of the 28th Annual IEEE Symposium on Foundations of Computer Science*, pages 185–194, 1987.
41. E. Upfal. Efficient schemes for parallel communication. *Journal of the ACM*, 31:507–517, 1984. (Preliminary version in PODC 82.)
42. E. Upfal. An  $O(\log N)$  deterministic packet routing scheme. *Journal of the ACM*, pages 55–70, 1992. (Preliminary version in STOC 1989.)
43. L.G. Valiant. A scheme for fast parallel communication. *SIAM Journal on Computing*, 11(2):350–361, 1982.
44. L.G. Valiant. General purpose parallel architectures. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science, Vol. A: Algorithms and Complexity*. MIT Press, Cambridge, 1990, pages 943–972.
45. L. G. Valiant and G. J. Brebner. Universal schemes for parallel communication. In *Proc. of the 13th Annual ACM Symposium on Theory of Computing*, pages 263–277, 1981.