Experimental Evaluation of Hot–Potato Routing Algorithms on 2–Dimensional Processor Arrays*

Constantinos Bartzis[1**], Ioannis Caragiannis[2], Christos Kaklamanis[2], and Ioannis Vergados[2]

[1] Department of Computer Science and Engineering
University of California, Santa Barbara, USA
E–mail: bar@cs.ucsb.edu.

[2] Computer Technology Institute and
Dept. of Computer Engineering and Informatics,
University of Patras, 26500 Rio, Greece.
E–mail: {caragian,kakl,vergad}@cti.gr

Abstract. In this paper we consider the problem of routing packets in two–dimensional torus–connected processor arrays. We describe the implementation of four hot–potato routing algorithms and study their performance by experimentation. The algorithms are either greedy in the sense that packets try to move towards their destination by adaptively using a shortest path, or have the property that the path traversed by any packet approximates the path traversed by the greedy routing algorithm in the store–and–forward model. Moreover, they are simple and were implemented according to the one– or two–pass scheme.
In our experiments, we consider the static case of the routing problem where we study permutation and random destination input instances as well as the dynamic case of the problem under the stochastic model for the continuous generation of packets. Our experimental results demonstrate that, although the running time of all the algorithms is close to the optimal for static routing problems, heavy traffic may influence differently the performance of each algorithm in the dynamic case.

1 Introduction

We consider a form of packet routing known as *hot–potato* (or *deflection*) routing. The main feature of hot–potato routing is that it does not involve queues at the nodes of the networks. Hot–potato routing algorithms has been observed to work well in practice; parallel computers like the Connection Machine, the Caltech Mosaic C, and the HEP multiprocessor utilize some form of hot–potato routing.

* This work was partially funded by the European Union under IST FET Project ALCOM–FT and RTN Project ARACNE.

** Part of this work was performed while the author was at the Department of Computer Engineering and Informatics, University of Patras, Greece.

Hot–potato routing is also well–suited for all–optical communication networks, where buffering messages is too costly.

The network is modeled as a directed graph where the nodes are the processors and the unidirectional edges are communication links between processors. Each processor has an injection buffer and a delivery buffer. When a new packet is generated, it is stored in the injection buffer of its source processor; when a packet reaches its destination processor, it is stored in the delivery buffer. The routing is performed in discrete, synchronous time steps. During each step, a processor receives zero or one packet along each incoming edge and must send all the packets it received out along outgoing edges with at most one packet leaving per outgoing edge. No buffers are required to hold the packets between the time steps. The topology we consider in this paper is that of the 2–dimensional torus–connected processor array.

In static (or batch) routing problems, all processors generate a single packet simoultaneously. The running time of a routing problem is the number of time steps required to deliver all packets to their destinations. A considerable part of the theoretical work is devoted to routing problems with random destinations (i.e., batch routing problems where each packet selects as destination a processor of the network, uniformly at random) and permutation routing problems (i.e., batch routing problems where each processor is the destination of exactly one packet).

In dynamic routing problems, each node continously generates packets with an injection rate λ. New packets are stored in the injection buffer and wait to be served. When a processor receives less than four packets along its incoming edges, it considers a packet from its injection buffer. When a packet starts moving, it is never buffered at any node until it reaches its destination, where it is stored in the delivery buffer and absorbed.

The basic characteristic of hot–potato routing algorithms is that they do not buffer moving packets. Any packet that arrives at a node other than its destination must immediately be forwarded to another node. This may make some packets to be “deflected” away from their prefered destination. This cannot happen in the store–and–forward model, where each packet chooses a path from its source to its destination and proceeds along this path; if it arrives at a node and finds its prefered outgoing edge occupied, it is stored in a queue of packets wishing to use that edge.

The first hot-potato algorithm was proposed by Baran [2]. Borodin and Hopcroft [7], Prager [17] and Hajek [12] presented algorithms for hypercubes. Hot–potato routing algorithms for 2–dimensional meshes and tori were proposed by Bar–Noy et al. [3], Ben–Aroya et al. [4], Newman and Schuster [16], Kaufman et al. [14], Feige and Raghavan [11], Kaklamanis et al. [13], Borodin et al. [6], and Feige [10]. Some of the above results (e.g., [13]) extend to meshes and tori of higher dimensions. All of them deal with batch routing problems. The only study of the dynamic case we are aware of is that of Broder and Upfal [8].

In spite of the important theoretical work done so far, there are some implementation issues that have to be considered in order to design useful hot–potato algorithms in practice. Some of the algorithms proposed in the bibliography take impractically complicated decisions. Especially in cases of continuous generation of packets and heavy traffic, routing decisions should be simple and fast. There have been several attempts to give a formal definition of the term "simple" for hot–potato routing algorithms. The papers of Feige [10] and Borodin et al. [6] are in this direction. Their results are comparable to earlier ones but their algorithms are more appropriate for practical implementations.

In particular, a desired characteristic that indicates simplicity of these algorithms is the one–pass property introduced by Hajek [12] and by Feige and Raghavan [11]. Packets that enter a node are considered one by one in a predefined order and each of them is assigned to an outgoing link not previously assigned to another packet. An extension of this property is the k-pass, according to which, each input packet is considered k times before any routing decisions are made.

An important class of hot–potato routing algorithms is that of greedy algorithms. In these algorithms, each node forwards each packet closer to its destination whenever possible. Although greedy algorithms have been observed to work well in practice (for static routing problems), the known theoretical results for their performance are far from being tight. Recently, Busch et al. [9] present a greedy algorithm and prove upper bounds of $O(n)$ for their average running time and of $O(n \log n)$ for their worse–case running time on static routing problems.

Especially for meshes and tori, a class of hot–potato routing algorithms that has received much attention is that of algorithms that make packets follow paths that approximate their natural greedy path (i.e., the path utilized by the greedy routing algorithm in the store–and–forward model [15]). Such algorithms were proposed and analyzed in [13].

In this paper we describe the implementation of four hot–potato routing algorithms on the 2–dimensional torus network. The algorithms are either greedy or have the property that approximate the greedy path. Moreover, they are simple in the sense that they can be implemented obeying the one–pass or two–pass properties. We performed extensive simulation studies for static and dynamic routing problems. Our experimental results demonstrate that, although the running time of all the algorithms is close to the optimal for static routing problems, heavy traffic may influence differently the performance of each algorithm in the dynamic case.

2 Description of the algorithms

In this section we describe the four hot–potato routing algorithms we have implemented. In order to simplify the description of the algorithms, we introduce some notation. Consider a step of the execution of a routing algorithm, and let p be a packet at node $v = (i, j)$, whose destination is node $v' = (i', j')$. Node v has four outgoing edges. From the point of view of the packet p, we name those four edges $\mathbf{h}^+(p)$, $\mathbf{h}^-(p)$, $\mathbf{v}^+(p)$, and $\mathbf{v}^-(p)$, so that $\mathbf{h}^+(p)$ is the outgoing edge in the shortest path from node (i, j) to node (i, j') in the horizontal dimension,

$\mathbf{h}^-(p)$ is the outgoing edge in the opposite direction of $\mathbf{h}^+(p)$, $\mathbf{v}^+(p)$ is the outgoing edge in the shortest path from node (i, j) to node (i', j) in the vertical dimension, and $\mathbf{v}^-(p)$ is the outgoing edge in the opposite direction of $\mathbf{v}^+(p)$. We also denote by $\ell_h(p)$ and $\ell_v(p)$ the horizontal and the vertical distance of the packet from its destination, respectively.

In the following we describe the four algorithms, namely the greedy algorithm, algorithm A1, algorithm KKR, and algorithm A2. The greedy algorithm is the folklore algorithm mentioned in the bibliography. Algorithm KKR was proposed in [13]. Algorithm A1 is a simple algorithm that "approximates the greedy path" while algorithm A2 is a variation of algorithm KKR. To our knowledge, algorithms A1 and A2 have not been studied in previous work.

The greedy algorithm. At each time step of the algorithm, each node examines at most four incoming packets in some order (to be defined below). For each packet p, its horizontal distance from its destination $\ell_h(p)$ is compared to its vertical distance from its destination $\ell_v(p)$. If $\ell_h(p) > \ell_v(p)$, then the outgoing links are inspected in the order $\mathbf{h}^+(p)$, $\mathbf{v}^+(p)$, $\mathbf{v}^-(p)$, $\mathbf{h}^-(p)$, until a link not already assigned to some other packet is found. This link is then assigned to packet p. In case that $\ell_h(p) \leq \ell_v(p)$, the order is respectively $\mathbf{v}^+(p)$, $\mathbf{h}^+(p)$, $\mathbf{h}^-(p)$, $\mathbf{v}^-(p)$. In brief, the greedy algorithm which was implemented tries to move packets towards their destination by adaptively using shortest paths, also trying to minimize the difference between the horizontal and the vertical distance of each packet from its destination.

By its definition, it becomes apparent that the decisions the algorithm makes are purely local, since they depend on the destination of the incoming packets and on the order in which these packets are examined. The greedy algorithm was implemented obeying to the one–pass property. According to the order in which packets are examined, we implemented two different schemes: a fixed predefined scheme which is the same for all nodes, and a randomized scheme where each node selects a random ordering of incoming packets uniformly at random. The experiments presented in section 3 refer to the implentation with the randomized scheme. The results for the fixed scheme are slightly worse.

Note that the path a packet traverses from its origin to its destination may be very irregular. In the following, we consider algorithms that have the property that the path traversed by any packet approximates the path traversed by the greedy routing algorithm in the store–and–forward model.

A simple algorithm that approximates the greedy path. We describe algorithm A1 from the point of view of the motion of the packets.

- Each packet p starts moving along its row, following the shortest path towards its destination column.
- When it reaches its destination column, p attempts to use the link $\mathbf{v}^+(p)$. If the link is already assigned to another packet, the packet fails to turn.
- If p fails to turn into the correct column, it continues moving along its row, and tries to turn at succesive nodes.

– Once p turns, it start moving along its column, following the shortest path towards its destination row.
– When it reaches its destination row, p attempts to use the link $\mathbf{h}^+(p)$. If the link is already assigned to another packet, the packet fails to turn.
– If p fails to turn into the correct row, it again continues moving along its column, and tries to turn at succesive nodes.
– The packet continues to move alternatively along rows and columns, until it reaches its destination.

A simple priority scheme is required here: packets that are moving along a column (or row) and have not reached their destination row (or column) have the highest priority.

Due to the priority scheme, the algorithm cannot be implemented as one–pass. We introduced the following criterion which was implemented as two–pass and also allows fast decisions: at any step, the routing decision made by a node is the one that maximizes the number of packets that follow their prefered outgoing edge.

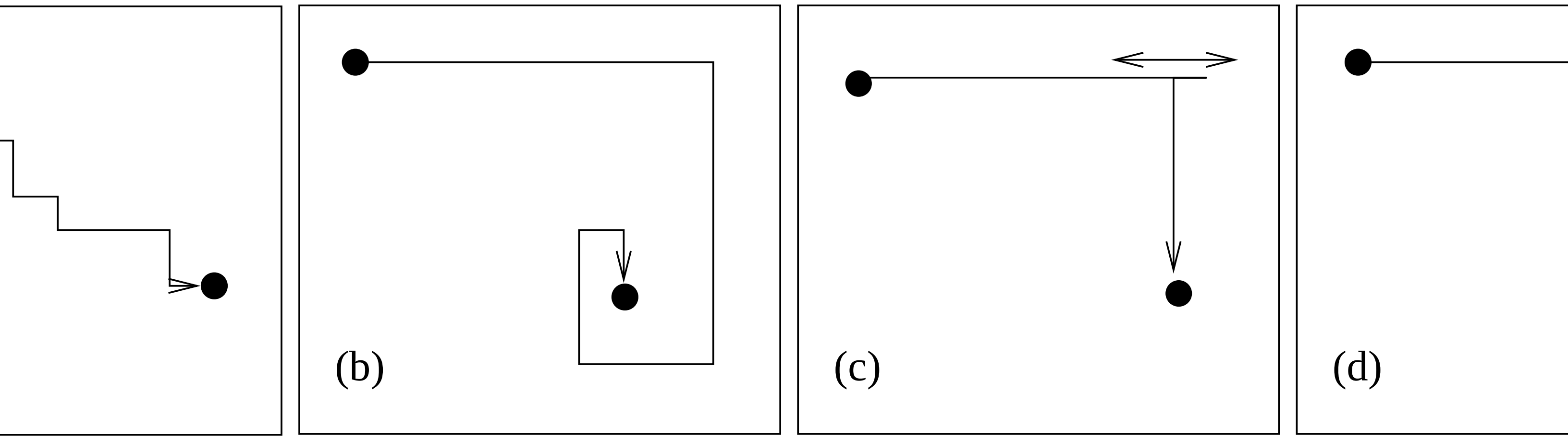

Fig. 1. Typical movement of packets performed by (a) the greedy algorithm, (b) algorithm A1, (c) algorithm KKR, and (d) algorithm A2.

The algorithm KKR. Again, we describe the algorithm from the point of view of the motion of the packets.

– Each packet p starts moving along its row, following the shortest path towards its destination column.
– When it reaches its destination column at node (i, j'), p attempts to use the link $\mathbf{v}^+(p)$. If the link is already assigned to another packet, the packet fails to turn.

– If p fails to turn into the correct column, it continues moving along its row, and tries to switch direction at succesive nodes. At such attempts, if it conflicts with packets that are travelling along the row and have not reached their destination column, then the packet fails to switch direction. If it conflicts with some other packet that is travelling along the row and has already gone past its destination column, then the packet with the higher priority "wins".
– Once p turns into the correct column, it moves vertically until it reaches its destination.

A fixed priority scheme is required to resolve conflicts. Due to this scheme, the algorithm cannot implemented as one–pass. It was implemented as two–pass.

A variation of the algorithm KKR. We observe that the algorithm KKR could be improved according to the following idea: during the time steps that a packet p is moving "back and forth" along its row trying to turn into the correct column, it could also try to turn at some other node following the link $\mathbf{v}^+(p)$. In this way, the vertical distance of the packet from its destination would (possibly) decrease. Algorithm A2 is based on this observation and differs from the original algorithm in only two points:

– If some packet p has failed to turn into its destination column, and also fails to reverse direction at some node, then it attempts to use link $\mathbf{v}^+(p)$. If that link is already assigned to some other packet, then p continues to move in the same direction.
– If p moves along a column other from its destination column, then it first tries to use link $\mathbf{h}^+(p)$. If it is not free then p continues to move vertically in the same direction.

This can be seen as a movement of a packet in its row until it reaches its destination column, and then, greedy movement to the destination processor. Thus, algorithm A2 maintains both properties: it is greedy and also approximates the greedy path.

The typical shape of the paths traversed by the packets during the execution of the algorithms is depicted in figure 1.

3 Experimentation

The four algorithms where implemented in C, and the results presented in the following were conducted by simulation experiments on a Pentium III/500Mhz running Solaris 7.

In the static model, packets are initially stored at the injection buffer and and start moving according to the routing algorithm. Initially, each processor has one packet stored in its injection buffer. We consider 2 routing problems:

– *random destinations*: each packet is assigned as destination a processor, selected among all the processors of the network, uniformly at random.

– *random permutations*: the routing problem is selected uniformly at random among all possible permutations.

Each algorithm was executed on 100 inputs for both the cases of random destinations and random permutations on tori of several sizes. The parameter of interest was the running time of the algorithms. Statistics (maximum observed value, standard deviation, and average value) on the running time of the algorithms on routing problems with random destinations in tori 200×200 and 500×500 are depicted in table 1. The distribution of the running time of each algorithm is graphically represented in figure 2. The results from the execution of the algorithms on random permutations are close to those for random destinations.

	200×200			500×500		
	Average	Max.	Std dev.	Average	Max.	Std dev.
Greedy	202.69	207	1.509	505.85	512	2.032
A1	201.39	205	1.214	501.16	506	1.412
KKR	205.19	214	2.881	505.13	511	2.553
A2	204.98	217	3.387	505.2	515	2.785

Table 1. Statistics from the execution of the four algorithms on 100 routing problems with random destinations in tori 200×200 and 500×500. The average and maximum observed running time, as well as the standard deviation of the running time is shown.

The performance of all algorithms is very close to the optimal. We believe that all the four algorithms route (almost all) batch routing problems in time $n + O(\log n)$ on the $n \times n$ torus. Such a strong theoretical result has only been proved for algorithm KKR in [13]. Algorithm A1 has slightly better performance than the other three algorithms. Surprisingly, algorithm A2 does not improve the running time of algorithm KKR.

In the dynamic model, packets with random destinations are continuously generated at each processor with a rate λ (i.e., at each time step, a processor generates a packet with probability λ, independently from the other processors) and stored in the injection buffer. Injection buffers have been implemented as FIFO (first–in–first–out) queues; so the network together with the injection buffers is considered as a queueing system. Once a packet leaves the injection buffer of its origin processor, it starts moving according to the routing algorithm until it reaches its destination, where it is stored in the delivery buffer and absorbed (leaves the system).

In our experiments under the dynamic model, parameters of interest were the delay of packets, the number of packets in the system (i.e., the number of packets in injection buffers and packets being routed) and the network throughput, i.e., the maximum injection rate for which the system is stable.

A theoretical maximum value for the maximum injection rate on the $n \times n$ torus is given by $\lambda_{max} = \frac{8}{n}$. For higher injection rates, no matter which algorithm

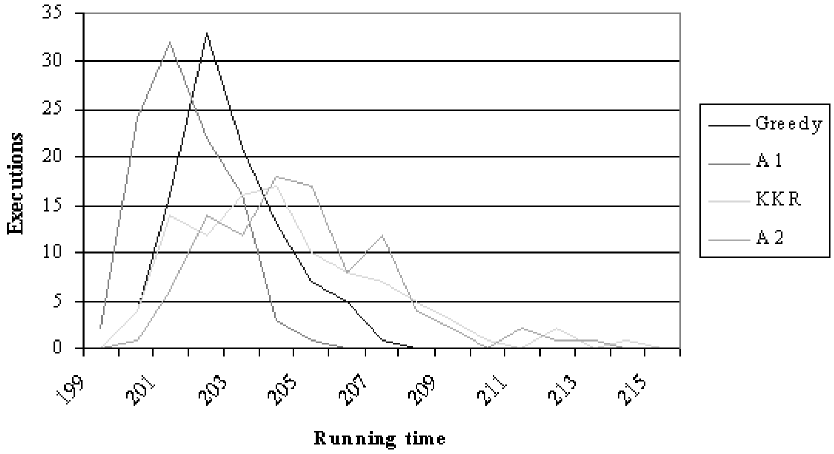

Fig. 2. The distribution of the running time of the four algorithms, when routing packets with random destinations in a 200×200 torus.

is used, packets are congested when they try to pass through the bisection of the torus, and the system becomes unstable (i.e., the number of the packets in the system continuously increases). We alternatively express the injection rate (and the network throughput) as a percentage of its theoretical maximum value. Although we never observe stable behavior of the system for injection rates very close to the theoretical maximum, we observed network throughput up to $72,5\%$.

	Average delay			Average size of injection buffers		
	$\lambda = 0.005$	$\lambda = 0.01$	$\lambda = 0.02$	$\lambda = 0.005$	$\lambda = 0.01$	$\lambda = 0.02$
Greedy	0.76	2.547	10.932	189	1,026	9,088
A1	0.336	0.943	4.664	156	386	3,835
KKR	0.475	1.304	6.456	78	543	5,383
A2	0.444	0.88	4.444	46	486	3,720

Table 2. Average delay of packets and average total size of injection buffers. The average on the delay was taken among all packets that reached their destinations within $2,500$ steps of execution. The average size of injection buffers was computed for the steps of execution 1000–2500 (note that for the first approximately 200 steps of execution the system has not been stabilized).

We performed experiments on the 200×200 torus for injection rates smaller than 50%. In this case, for the four algorithms we study, the network is stable. The results (average packet delay and average number of packets in the injection buffers) from the execution of the algorihms for injection rates $\lambda = 0.005, 0.01$, and 0.02, are depicted in table 2. We observe that, even for small injection rate, packets experience significant delays when routed with the greedy algorithm, while the average number of the packets in the network is large. Especially the average (total) size of injection buffers when the greedy algorithm is used is more than twice the average size of injection buffers when any of the other three algorithms is used. This could make us assume that the greedy algorithm has the smallest network throughput, but this is not true, as we will see below.

We also performed experiments for injection rates higher than the network throughput. We studied the behavior of the four routing algorithms for $\lambda = 0.03$ in the torus 200×200 (see figure 3). Algorithm A2 has the best performance, as the number of packets in the system increases with the lowest rate. Algorithm A1 has the worse performance as the number of packets in the system increases dramatically.

We attempted to obtain a rough estimation of the network throughput using the average rate with which the average number of packets in the system increases (by the curves of figure 3) and the assumption that, even when the system is unstable, the service rate equals the network throughput. Surprisingly, we obtained sharp approximations to the network throughput of all algorithms except for algorithm A1; the approximation of the network throughput for this algorithm was very pessimistic (less than 40% of the theoretical maximum rate). This means that the three algorithms still have high service rate, even in unstable cases.

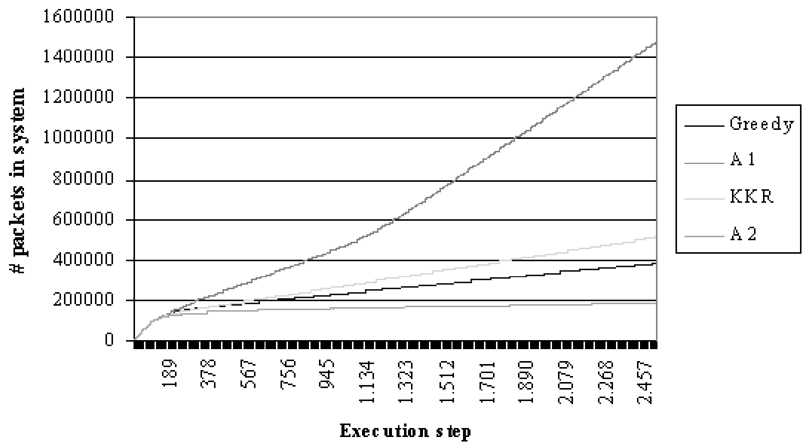

Fig. 3. An unstable case ($\lambda = 0.03$) for the four algorithms. The number of the packets in system in the first 2,500 steps of execution is shown. Notice the knee of curve A1 close to the 1150th step. We did not observe similar behavior for the other three algorithms, even in 100,000 steps of execution.

In our experiments on the 200×200 torus, the network throughput observed was about 0,0276 (69%) for the greedy algorithm, 0.0254 (63.5%) for algorithm A1 0.0265 (66.25%) for algorithm KKR, and 0.029 (72.5%) for algorithm A2.

Whether the network throughput can be improved using different implementations of the injection buffer will be part of our future work.

References

1. A. Acampora and S. Shah. Multihop Lightwave Networks: a Comparison of Store and Forward and Hot–Potato Routing. In *IEEE INFOCOM*, pp. 10–19, 1991.
2. P. Baran. On Distributed Communication Networks. *IEEE Transactions on Communications*, pp. 1–9, 1964.
3. A. Bar–Noy, P. Raghavan, B. Shieber, and H. Tamaki. Fast Deflection Routing for Packets and Worms. In *Proc. of the 12th Annual ACM Symposium on Principles of Distributed Computing*, pp. 75–86, 1993.
4. I. Ben–Aroya, T. Eilam, and Schuster. Greedy Hot–Potato Routing on the Two–Dimensional Mesh. *Distributed Computing*, 9(1):3–19, 1995.
5. A. Ben–Dor, S. Halevi, and A. Schuster. Potential Function Analysis of Greedy Hot–Potato Routing. In *Proc. of the 13th Annual ACM Symposium on Principles of Distributed Computing*, pp. 225–234, 1994.
6. A. Borodin, Y. Rabani, and B. Schieber. Deterministic Many–to–Many Hot–Potato Routing. *IEEE Transactions on Parallel and Distributed Systems*, 8(6): 587–596, 1997.
7. A. Borodin and J. Hopcroft. Routing, Merging, and Sorting on Parallel Models of Computation. *Journal of Computer and System Sciences*, 30:130–145, 1985.
8. A. Broder and E. Upfal. Dynamic Deflection Routing on Arrays. In *Proc. of the 28th Annual ACM Symposium on the Theory of Computing*, pp. 348–358, 1996.
9. C. Busch, M. Herlihy, and R. Wattenhofer. Randomized Greedy Hot–Potato Routing. In *Proc. of the 11th Annual ACM/SIAM Symposium on Discrete Algorithms (SODA '00)*, pp. 458–466, 2000.
10. U. Feige. Observations on Hot–Potato Routing. In *Proc. of the 3rd Israeli Symposium on Theory of Computing and Systems*, pp. 30–39, 1995.
11. U. Feige and P. Raghavan. Exact Analysis of Hot–Potato Routing. In *Proc. of the 33rd Annual IEEE Symposium on Foundations of Computer Science*, pp. 553–562, 1992.
12. B. Hajek. Bounds on Evacuation Time for Deflection Routing. *Distributed Computing*, 5:1–6, 1991.
13. C. Kaklamanis, D. Krizanc, and S. Rao. Hot–Potato Routing on Processor Arrays. In *Proc. of the 5th Annual ACM Symposium on Parallel Algorithms and Architectures*, pp. 273–282, 1993.
14. M. Kaufmann, H. Lauer, and H. Schroder. Fast Deterministic Hot–Potato Routing on Meshes. In *Proc. of the 5th International Symposium on Algorithms and Computation*, LNCS 834, Springer–Verlag, pp. 333-341, 1994.
15. F.T. Leighton. Average Case Analysis of Greedy Routing Algorithm on Arrays. In *Proc. of the 2nd Annual ACM Symposium on Parallel Algorithms and Architectures*, pp. 2–10, 1990.
16. I. Newman and A. Schuster. Hot–Potato Algorithms for Permutation Routing. *IEEE Transactions on Parallel and Distributed Systems*, 6(11): 1168–1176, 1995.
17. R. Prager. An Algorithm for Routing in Hypercube Networks. Master's thesis, University of Toronto, 1986.
18. P. Spirakis and V. Triantafillou. Pure Greedy Hot–potato Routing in the 2–D Mesh with Random Destinations. *Parallel Processing Letters*, 7(3): 249–258, 1997.