Alcom-FT Technical Report Series
ALCOMFT-TR-01-56

Greedy Dynamic Hot–Potato Routing on Arrays*

Ioannis Caragiannis† Christos Kaklamanis‡ Ioannis Vergados§

Abstract

In this paper we consider the problem of routing packets in two–dimensional torus–connected processor arrays. Motivated by recent theoretical work on dynamic routing, we address the dynamic case where packets are continuously injected into the network. We describe three greedy hot–potato routing algorithms and present simulation experiments on tori of several sizes using four well–known greedy (also known as work–conserving) protocols (namely FIFO, LIFO, FTG, and NTG) for the implementation of injection buffers. Our results demonstrate that there exists a greedy hot–potato routing algorithm that is stable for all greedy injection queueing protocols for injection rates very close to 100% *of the network capacity. Furthermore, according to the algorithms we studied, we can claim that the one–pass property is not appropriate for the dynamic case, since the system cannot achieve stability at high injection rates.*

1. Introduction

We study packet routing algorithms on synchronous networks in which, at each unit of time, at most one packet can cross any given communication link. Given a network and a routing algorithm, we wish to evaluate the performance of the algorithm in a dynamic setting where packets are continuously generated and need to be routed to their respective destinations.

The network is modeled as a directed graph where the nodes are the processors and the unidirectional edges are communication links between processors. Each node has an injection buffer and a delivery buffer. When a new packet is generated, it is stored in the injection buffer of its source node; when a packet reaches its destination node, it is stored in the delivery buffer. Routing is performed in discrete, synchronous time steps. During each step, each node examines newly received and already stored packets and forwards packets along outgoing edges with at most one packet leaving per outgoing edge.

Routing algorithms are usually classified according to the following parameters:

Oblivious vs adaptive. In an oblivious routing algorithm, the path a packet follows in the network is determined at the time the packet is injected into the network. In an adaptive routing algorithm, the path is not known a–priori, and depends on other events that happen in the network.

Unbounded vs bounded edge buffers. The routing algorithm may require unbounded buffers in order to temporarily store packets queued at some intermediate node on their way to their destination, may work with bounded buffers (and then provide a mechanism to prevent overflow of the buffers, e.g., by deflecting packets or by dropping packets), or require no buffers at all. In any case, we assume that arbitrarily many packets can be stored at the injection buffer of their source node prior to being injected into the network.

Randomization. A routing algorithm may be randomized (routing decisions depend also on random coin tosses) or deterministic.

1.1. Hot–potato routing

In this paper we consider a form of packet routing known as *hot–potato* (or *deflection*) routing. 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 force some packets to be "deflected" away from their prefered destination. This cannot happen with oblivious store–and–forward routing algorithms, in which each packet chooses a path from its source

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

†Computer Technology Institute and Department of Computer Engineering and Informatics, University of Patras, 26500 Rio, Greece. E–mail: caragian@cti.gr

‡Computer Technology Institute and Department of Computer Engineering and Informatics, University of Patras, 26500 Rio, Greece. E–mail: kakl@cti.gr

§Computer Science Department, Brown University, Providence, RI 02912, USA. Part of this work was done while the author was at the Department of Computer Engineering and Informatics, University of Patras, Greece.

to its destination and proceeds along this path; if it arrives at a node and finds its preferred outgoing edge occupied, it is stored in a queue of packets wishing to use that edge.

Most of the theoretical work on hot–potato routing is devoted to static routing problems. In static (or batch) routing problems, all nodes generate a single packet simultaneously. 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 node of the network, uniformly at random) and permutation routing problems (i.e., batch routing problems where each node is the destination of exactly one packet).

The first hot-potato algorithm was proposed by Baran [4]. Borodin and Hopcroft [11], Prager [25] and Hajek [18] presented algorithms for hypercubes. Hot–potato routing algorithms for 2–dimensional meshes and tori were proposed by Bar–Noy et al. [5], Ben–Aroya et al. [7], Newman and Schuster [24], Kaufman et al. [21], Feige and Raghavan [17], Kaklamanis et al. [20], Borodin et al. [10], and Feige [16]. Some of the above results (e.g., [20]) extend to meshes and tori of higher dimensions. All of them deal with batch routing problems.

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. 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 [16] and Borodin et al. [10] 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 hot–potato routing algorithms is the *one–pass* property introduced by Hajek [18] and by Feige and Raghavan [17]. Packets that enter a node are considered one by one in a predefined order and each of them is assigned to an outgoing edge not previously assigned to another packet.

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 on their performance are far from being tight. Recently, Busch et al. [13, 14] 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.

1.2. Dynamic routing

Motivated by recent theoretical work, we consider the dynamic case where packets are continuously injected into the network. In this setting, the performance of routing algorithms is analyzed under some arrival processes, which describe how new packets are generated. Usually, two types of arrival processes are used.

Adversarial. At every time step, an adversary generates a collection of routing requests of its choice (but subject to constraints derived from the capacity of the network), where each routing request specifies a source node, a destination node, and, possibly, some additional information (such as the path to be followed from source to destination).

Stochastic. Packets with random destinations are continuously generated at each node at a rate λ (i.e., at each time step, each node generates a packet with probability λ, independently from the other nodes).

We focus on stochastic arrival processes. Packets with random destinations are continuously generated at each node with a rate λ (i.e., at each time step, a node generates a packet with probability λ, independently from the other nodes) and stored in the injection buffer. Injection buffers have been implemented as queues; so the network together with the injection buffers is considered as a queueing system [22]. When a node receives less than four packets along its incoming edges, it considers packets from its injection buffer. Once a packet leaves the injection buffer of its source node and starts moving, it is never buffered at any node until it reaches its destination, where it is stored in the delivery buffer and absorbed (leaves the system).

For the implementation of injection buffers as queues, we use the following greedy (also known as work–conserving) queueing protocols:

First–In–First–Out (FIFO). FIFO gives precedence to the packet injected the earliest.

Last–In–First–Out (LIFO). LIFO gives precedence to the packet injected the latest.

Furthest–to–Go (FTG). FTG gives precedence to the packet whose distance from its destination is maximal.

Nearest–to–Go (NTG). NTG gives precedence to the packet whose distance from its destination is minimal.

In the study of dynamic routing algorithms, the following performance measures are of interest:

Queue sizes. The length of queues that build up in the network. Note that for hot–potato routing, queue size refers only to injection buffers, since there are no edge buffers at the network nodes.

Delays. The number of time steps it takes a packet to reach its destination (compared to the length of the shortest path between its source and destination).

Stability. A routing algorithm is stable (under a certain arrival process) if queue lengths and delays remain bounded.

Network throughput. The maximum injection rate for which the system remains stable.

Robustness. Slight changes in the packet arrival process cause only slight changes in the performance of the algorithms (with respect to stability, average delay, and average number of packets in the system).

The topology we consider in this paper is that of the 2–dimensional torus–connected processor array. A theoretical maximum value for the injection rate on the $n \times n$ torus (i.e., the network capacity) is given by $\lambda_{max} = 8/n$ [23]. For higher injection rates, regardless of the algorithm and queueing protocol 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 λ_{max}.

Much work has been devoted to establishing stability results and performance bounds for oblivious routing algorithms in several networks. In the stochastic model for continuous generation of packets, we point out the pioneering work of Leighton [23] who analyzed the behavior of the greedy store–and–forward routing algorithm with FTG queues on the 2–dimensional mesh and torus and proved that it achieves stability for any injection rate less than the capacity of the network. This result was strengthened in [19] where it was proved that the same stability result holds for any greedy queueing protocol.

In the adversarial traffic model, stability results for oblivious routing algorithms and greedy queueing protocols have been proved in [9] and [3]. Aiello et al. [2] present stability results for adaptive routing algorithms.

The only theoretical work on dynamic hot–potato routing we are aware of are the papers of Broder and Upfal [12] and Feige [15]. Broder and Upfal in [12] analyze the performance of a hot–potato algorithm on the 2–dimensional mesh with FIFO queues and prove that it achieves stability for injection rate of 50% of the capacity of the network. However, their algorithm is different in spirit than the algorithms we consider in the current paper, since it drops packets to avoid conflicts and uses extra packets to acknowledge that packets reach their destinations. Feige [15] presents a simple hot–potato routing algorithm that achieves stability even when the injection rate is equal to the network capacity. However, this algorithm uses a time–dependent injection queueing protocol to avoid conflicts between packets. Moreover, many packets may experience significant delays regardless of the injection rate.

In recent experimental work, Bartzis et al. [6] present a black–box mechanism for transforming known static hot–potato routing algorithms so that they work on the dynamic case using FIFO queues. However, inherent characteristics of the algorithms studied as well as the black–box mechanism itself are such that stability at injection rates higher of 75% is infeasible.

1.3. Our results

In this paper we describe three hot–potato routing algorithms (Algorithm A, B, and C) on the 2–dimensional torus network. The algorithms are greedy (i.e., they are not time–dependent, they do not drop packets, and they use no acknowledgement packets) and use randomization in a simple way. Algorithm A takes routing decisions based on an optimization criterion, while Algorithms B and C are one–pass.

We present experimental results obtained by extensive simulation studies on tori of several sizes with FIFO, LIFO, FTG, and NTG queues, for a wide range of packet injection rates.

Our main results can be summarized as follows:

- Algorithm A achieves stability for injection rates higher than 90% for all four queueing protocols.
- For the three algorithms, there is an exponential bound on the tail of the delay distribution, which is extremely sharp for algorithm A.
- Algorithm A gives the best results with FIFO queues and the worst ones with FTG queues, while the performance of algorithms B and C is not affected by the queueing protocol used.
- The one–pass property does not seem to be appropriate for dynamic routing since algorithms B and C cannot achieve stability at injection rates higher than 65%.
- Our bounds are robust in the sense of [15].

The rest of the paper is structured as follows. The three algorithms are described in Section 2. We present our experimental results in Section 3 and discuss possible extensions and directions for future work in Section 4.

2. Description of the algorithms

In this section we describe the three greedy algorithms (namely Algorithms A, B, and C). At each step, each node examines packets that enter the node from its incoming edges and packets from the injection queue, and takes a routing decision, i.e., decides how packets will be forwarded through the outgoing edges of the node in such a way that at most one packet is forwarded through each outgoing edge.

Note that, if a node receives less than four packets from its incoming edges at some step, it selects packets from its injection queue trying to forward exactly four packets through the outgoing edges. Thus, at any step, the node forwards four packets if there are enough packets in its injection queue. This assumption is close to the model used in dynamic store–and–forward routing, as opposed to the model of [6] where the number of packets that are forwarded at an execution step may be less than four even if the injection queue contains many packets. As we will see, at least for algorithm A, this assumption allows for stability at very high injection rates.

In the following, we describe the way routing decisions are made by the algorithms.

Algorithm A. At any step, at most four packets may be at some node. A routing decision will send each packet to a distinct outgoing edge. For any packet, given a routing decision, we define the benefit of the packet to be 1 if the route the packet takes is closer to its destination, 0 otherwise. Clearly, the number of routing decisions equals the number of all possible assignments of distinct outgoing edges to the packets, i.e., it is at most $4! = 24$ for any node of the 2–dimensional torus. The benefit of a routing decision is the sum of the benefits of packets, given this decision. The algorithm computes the benefit of each one of the routing decisions and selects uniformly at random one out of those that give the highest benefit (i.e., it maximizes the number of packets that are forwarded closer to their destination).

Clearly, this algorithm cannot be implemented obeying the one–pass property. The following two algorithms take routing decisions in an one–pass way.

Algorithm B. The routing decision of a node at any step is constructed in the following way:

- A random order in which packets are examined is computed.
- First, each of the (at most four) packets currently at a node, computes an order in which it will examine the outgoing edges. This order of outgoing edges is chosen uniformly at random among all orders of outgoing edges, in which edges that bring the packet closer to its destination precede the rest of the edges.
- Then, the algorithm examines the packets in a random order and each packet is assigned its most preferable edge that is available.

Algorithm C. The only difference between algorithms B and C is that C uses additional information about the destination node of each packet to take routing decisions.

Again, the order in which each packet examines the outgoing edges is chosen so that outgoing edges that bring the packet closer to its destination precede the rest of the edges. Furthermore, among the outgoing edges that route the packet closer to its destination the one that minimizes the difference between the vertical and the horizontal distance from the packet's destination will be examined first. Among the outgoing edges that route the packet away from its destination, the one that maximizes the difference between the vertical and the horizontal distance from the packet's destination will be examined last.

By their definition, it becomes apparent that the decisions the algorithms make are purely local, since they depend only on the destination of the incoming packets. Furthermore, the routing decisions of algorithms B and C also depend on the order in which packets are examined.

Surprisingly, although there is no strong intuition indicating that algorithm C is better than B, we will see in the next section that algorithm C is significantly better than algorithm B.

3. Experimental results

The 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. About 50 MB of data were gathered, in which the behavior of the algorithms (for a wide range of parameters using the four queueing protocols) was recorded.

In what follows, we use the term packet delay to denote the difference between the time a packet spends in the system and the length of the shortest path from its source to its destination. Note that packets are either delayed waiting in injection buffers or due to deflections. Instead of measuring the queue size of injection buffers, we measure the number of packets in the system, which includes packets waiting in injection buffers and packets being routed.

We have computed network throughput by observing the behavior of our algorithms on tori of size 50×50 after $10,000$ steps of execution. Our results demonstrate that algorithm A achieves stability for very high injection rates, while the one–pass algorithms are unstable even for small injection rates. The network throughput of the three algorithms under the four queueing protocols is depicted in Table 1. Observe that FIFO is the most appropriate queueing protocol for algorithm A while the four protocols have nearly the same performance when a one–pass algorithm is executed.

	50×50			100×100
	Alg. A	Alg. B	Alg. C	Alg. A
FIFO	81.9%	54.3%	61.3%	86.9%
LIFO	81.5%	54.4%	61.3%	86.8%
FTG	81.2%	54.4%	61.3%	86.6%
NTG	81.5%	54.4%	61.3%	86.8%

Table 1. Network throughput of algorithms A, B, and C in tori of size 50×50 and 100×100.

We have not computed the exact network throughput for larger tori but we observed that algorithm A (using any of the four queueing protocols) is stable for injection rates 91% in tori of size 200×200 and 94% in tori of size 500×500. Thus, it seems that network throughput of algorithm A increases with the size of the network. This tempts us to conjecture that algorithm A is stable for any greedy queueing protocol at any injection rate smaller than the capacity of the network as the size of the network goes to infinity.

We performed experiments on the 200×200 torus for small values of injection rate. In Tables 2 and 3, results for injection rates around 50% of the network capacity are depicted. Note that all algorithms are stable at this rate. However, the rate is not far away from the network throughput of algorithms B and C, while it is significantly smaller than the network throughput of A. We observe that there are large gaps in the performance of the three algorithms. Algorithm A achieves almost optimal average delay while the one–pass property maintained by algorithms B and C makes them have poor performance. A somewhat surprising result is that algorithm C is significantly better than algorithm B, not only in terms of network throughput, but also in terms of average delay and average number of packets in the system.

	45%	50%	55%
Alg. A	0.85	1.06	1.32
Alg. B	15.69	21.26	31.14
Alg. C	8.77	11.40	14.94

Table 2. Average delay on the 200×200 torus with FIFO queues for small injection rates. The average was taken among all packets that reached their destinations within $5,000$ steps of execution.

We now further explore how the queueing protocol used affects the performance of the routing algorithms in terms of average delay and average number of packets in the system. For each of the two one–pass algorithms, we observed no significant difference in their performance when different

	45%	50%	55%
Alg. A	72,665.8	80,800.3	89,181.2
Alg. B	83,467.7	97,268.6	115,888.9
Alg. C	78,380.1	89,309.9	101,255.5

Table 3. Average number of packets in the system on the 200×200 torus with FIFO queues for small injection rates. The average was computed for the steps $2,000 - 5,000$ of execution (note that for the first approximately 500 steps, the system does not reach the stable state).

queueing protocols are used. However, for algorithm A, there are small but interesting differences. The behavior of the algorithm on the 200×200 torus when packets are injected at rate 90% and queues are FIFO, LIFO, FTG, NTG is shown in Table 4.

	Average delay	Average no. of packets
FIFO	14.52	166,523.8
LIFO	14.60	166,600.2
FTG	14.86	167,124.1
NTG	14.61	166,414,7

Table 4. Average delay and average number of packets in the system for Algorithm A on the 200×200 torus at high injection rate (90%).

In Figure 1, the distribution of packet delay for algorithm A is shown. Note that when a packet stays at the injection queue of its source node, this increases the delay of the packet by 1, while a deflection increases the delay of the packet by 2. Thus, delays for small even numbers of steps are more frequent.

We observe that the distribution of packet delay is concentrated at a very small value. Moreover, it seems that the probability that a packet experiences delay t, is exponentially small in t. Compare it to the algorithm of Feige [15] on the $n \times n$ torus, where, for any injection rate, the distribution of packet delays is almost uniform for delays in the range $[0, n/2 - 1]$.

We also performed experiments for injection rates higher than the network throughput. We studied the behavior of algorithms B and C on the 200×200 torus for injection rate 70%. In this case, both algorithms are unstable. In Figure 2, we can see that the number of packets in the system continuously increases. Comparing the two algorithms, we observe that the number of packets in the system does not increase very fast when algorithm C is used.

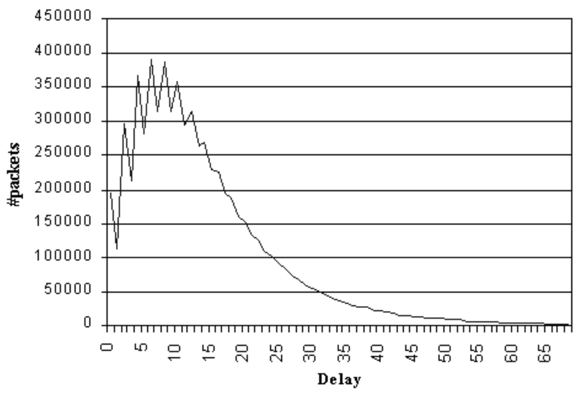

Figure 1. The distribution of packet delay for algorithm A on a 200×200 torus for injection rate 90%.

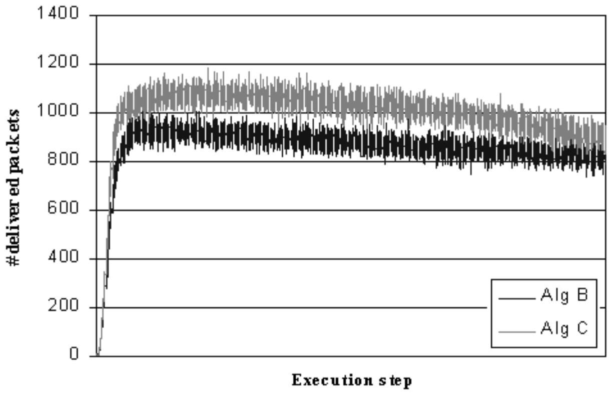

Figure 3. Service rate of algorithms B and C at an unstable case (rate=70%).

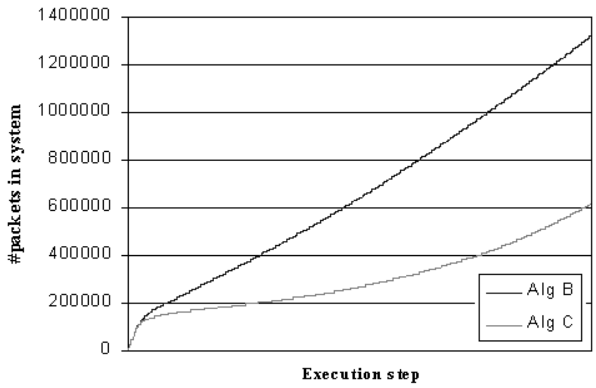

Figure 2. An unstable case (rate=70%) for algorithms B and C.

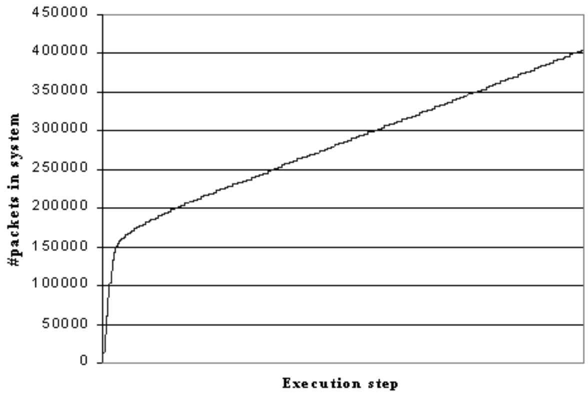

Figure 4. An unstable case (rate=95%) for algorithm A.

Moreover, the increase of the number of packets in the system is not constant neither for algorithm B nor for algorithm C. This means that the service rate of the system (i.e., the rate at which packets are delivered to their destinations) drops with time. This is documented in Figure 3 where the number of packets that are delivered at each step is depicted.

Instead, for algorithm A, the rate at which the number of packets in the system increases is constant, which means that service rate is constant and almost equal to the network throughput even in unstable cases (Figures 4 and 5). Using this observation, we can obtain fairly sharp estimations of network throughput when algorithm A is used on tori with FIFO, LIFO, FTG, NTG queues by studying the behavior of the system (i.e., the number of packets in the system) in an unstable case.

Finally, we mention that our bounds are robust in the sense of [15]. Performing experiments with the three algorithms using FIFO, LIFO, FTG, and NTG queues, we observed that slight changes in the (stochastic) packet arrival process cause only slight changes in the performance of the algorithms (with respect to stability, average delay, and average number of packets in the system).

4. Extensions

We conclude with a brief discussion on some interesting issues revealed by our work.

The experiments presented here can be extended to other greedy hot–potato routing algorithms and queueing protocols as well as to tori of higher dimensions. The optimization criterion used by algorithm A to take a routing decision can be more complicated, e.g., it may take into consideration packets that have been deflected many times, the time packets are in the system or the distance of packets to their

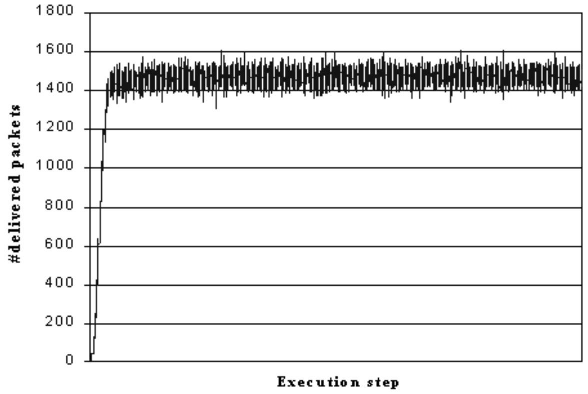

Figure 5. Service rate of algorithm A at an unstable case (rate=95%).

destinations. We plan to examine such variations of algorithm A and include the results in the final version of the paper.

Note that our work is limited to the study of the behavior of routing algorithms against stochastic adversaries. Constructing non–stochastic adversaries and evaluating the performance of algorithms against them seems to be a challenge for algorithmic engineering with applications that will cover many more topics than packet routing.

Definitely, theoretically proving stability results and performance bounds for greedy hot–potato routing algorithms in the dynamic case is a very interesting (although hard) open problem.

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] W. Aiello, E. Kushilevitz, R. Ostrovsky, A. Rosen. Adaptive packet routing for bursty adversarial traffic. In *Proc. of the 30th Annual ACM Symposium on Theory of Computing (STOC 98)*, pp. 359–368, 1998.

[3] M. Andrews, B. Awerbuch, A. Fernandez, J. Kleinberg, T. Leighton, Z. Liu. Universal Stability Results for Greedy Contention Resolution Protocols. In *Proc. of the 37th Annual IEEE Symposium on Foundations of Computer Science (FOCS 96)*, pp. 380–389, 1996.

[4] P. Baran. On Distributed Communication Networks. *IEEE Transactions on Communications*, pp. 1–9, 1964.

[5] 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.

[6] C. Bartzis, I. Caragiannis, C. Kaklamanis, and I. Vergados. Experimental Evaluation of Hot–Potato Routing Algorithms on 2–Dimensional Processor Arrays. In *Proc. of EUROPAR 2000*, to appear.

[7] I. Ben–Aroya, T. Eilam, and Schuster. Greedy Hot–Potato Routing on the Two–Dimensional Mesh. *Distributed Computing*, 9(1):3–19, 1995.

[8] 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.

[9] A. Borodin, J. Kleinberg, P. Raghavan, M. Sudan, D. Williamson. Adversarial Queueing Theory. In *Proc. of the 28th Annual ACM Symposium on Theory of Computing (STOC 96)*, pp. 376–385, 1996.

[10] 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.

[11] A. Borodin and J. Hopcroft. Routing, Merging, and Sorting on Parallel Models of Computation. *Journal of Computer and System Sciences*, 30:130–145, 1985.

[12] 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.

[13] 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.

[14] C. Busch, M. Herlihy, and R. Wattenhofer. Hard–Potato Routing. In *Proc. of the 32nd Annual ACM Symposium on Theory of Computing (STOC '00)*, 2000, to appear.

[15] U. Feige. Nonmonotonic Phenomena in Packet Routing. In *Proc. of the 31st ACM Annual Symposium on Theory of Computing (STOC 99)*, pp. 583–591, 1999.

[16] U. Feige. Observations on Hot–Potato Routing. In *Proc. of the 3rd Israeli Symposium on Theory of Computing and Systems*, pp. 30–39, 1995.

[17] 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.

[18] B. Hajek. Bounds on Evacuation Time for Deflection Routing. *Distributed Computing*, 5:1–6, 1991.

[19] N. Kahale, F.T. Leighton. Greedy Dynamic Routing on Arrays. In *Proc. of the 6th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA 95)*, 1995.

[20] 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.

[21] 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.

[22] L. Kleinrock. *Queueing Systems, Volume 1: Theory*. John Wiley & Sons, 1975.

[23] 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.

[24] I. Newman and A. Schuster. Hot–Potato Algorithms for Permutation Routing. *IEEE Transactions on Parallel and Distributed Systems*, 6(11): 1168–1176, 1995.

[25] R. Prager. An Algorithm for Routing in Hypercube Networks. Master's thesis, University of Toronto, 1986.

[26] 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.