Globally Adaptive Load-Balanced Routing on Tori

Arjun Singh, William J Dally, Brian Towles, and Amit K Gupta
Computer Systems Laboratory, Stanford University
{arjuns, billd, agupta, btowles}@cva.stanford.edu

***Abstract*— We introduce a new method of adaptive routing on k-ary n-cubes, *Globally Adaptive Load-Balance (GAL)*. GAL makes global routing decisions using global information. In contrast, most previous adaptive routing algorithms make local routing decisions using local information (typically channel queue depth). GAL senses global congestion using segmented injection queues to decide the directions to route in each dimension. It further load balances the network by routing in the selected directions adaptively. Using global information, GAL achieves the performance (latency and throughput) of minimal adaptive routing on benign traffic patterns and performs as well as the best obliviously load-balanced routing algorithm (GOAL) on adversarial traffic.**

I. Introduction

Interconnection networks are widely used for processor-memory interconnect [7], for I/O interconnect, and as switch and router fabrics [3]. Torus, or k-ary n-cube [2], topologies are very popular in all of these applications. Torus networks have high path diversity, offering many alternative paths between a source and destination node which a routing algorithm must choose from.

In this paper we introduce *Globally Adaptive Load-Balance* (GAL) — a non-minimal, adaptive routing algorithm for torus networks that adapts globally to balance load across minimal and non-minimal paths. At low traffic rates, when there is no congestion, GAL routes all traffic along minimal paths — giving minimum latency. For adversarial patterns, at high traffic rates, GAL senses congestion in the minimal quadrants, and only when this congestion occurs routes non-minimally. Only the minimum amount of traffic needed to maintain the minimal quadrant on the edge of congestion is routed non-minimally. At saturation, the queues for all quadrants are balanced — balancing load across the quadrants and hence providing maximum throughput.

TABLE I
Report Card for four adaptive routing algorithms

	CHAOS	MIN AD	GOAL	**GAL**
Θ_{NN}	4.0 (A)	4.0 (A)	2.33 (C)	**4.0 (A)**
Θ_{TOR}	0.36 (C)	0.33(C)	0.53 (A)	**0.53 (A)**
Θ_{avg}	0.55 (C)	0.63(B)	0.67(A)	**0.73 (A)**
Lat	4.4(A)	4.4(A)	6.2(C)	**4.4 (A)**
Stab	No (F)	Yes (P)	Yes (P)	**Yes (P)**

Table I compares GAL to three previously reported adaptive routing algorithms on three throughput (Θ) measures, low-load latency, and stability. On benign traffic patterns (represented by nearest neighbor), GAL's minimal injection queues remain below threshold. Thus, GAL routes all benign traffic minimally and matches the high throughput (Θ_{NN}) and low load latency (Lat)[1] of minimal algorithms (which all receive an "A" grade) while GOAL gives somewhat lower ("C") performance. On hard traffic patterns (represented by tornado) operating near saturation, GAL's global adaptation matches the ideal load balance of GOAL. Thus, both achieve highest throughput (Θ_{TOR}) on these patterns ("A"s). Minimal algorithms achieve only 62% of this peak throughput ("C"). GAL's ability to route minimally when able, yet to load balance when required gives the best observed performance on average permutations (Θ_{avg}). Finally, all algorithms except CHAOS are stable and get a passing grade "P" on stability. We discuss the stability of GAL in Section V.

II. Previous Adaptive Routing Algorithms

The shortcomings of previous adaptive routing algorithms are illustrated by considering routing nearest-neighbor and tornado traffic on an 8 node ring network. For the benign nearest neighbor (NN) traffic pattern, all traffic from node i is distributed equally amongst its neighboring nodes (half to node $i-1$ and half to node $i+1$). Assuming unit bandwidth channels, each node can inject traffic at an optimal rate of 2 packets per cycle (throughput, $\Theta = 2$) if traffic is simply routed along minimal paths. For the difficult tornado (TOR) traffic pattern, all traffic from node i is sent nearly half-way-around the ring, to node $i+3$. Optimal routing for TOR requires some traffic to be sent non-minimally as shown in Figure 1(a). The ideal throughput for tornado is $\Theta = 8/15$.

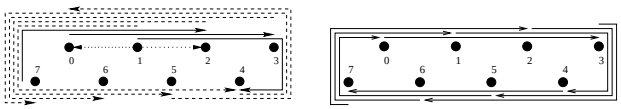

Fig. 1. (a) Optimally routed tornado traffic in an 8 node ring. Load is shown for two links depicted in dotted lines. The dashed (solid) lines contribute a link load of $\frac{3}{8}$ ($\frac{5}{8}$). All links are equally loaded with load = $\frac{15}{8}$. (b) Minimally routed tornado traffic. Clockwise (counter clockwise) link load is 3 (0).

Globally oblivious algorithms such as GOAL [8] optimally load balance difficult traffic patterns such as TOR, achieving optimal throughput on such patterns. Unfortunately, this is done at the expense of benign traffic patterns such as NN. GOAL has high latency and achieves a throughput of only

Manuscript submitted: 17 Feb. 2004. Manuscript accepted: 17 Mar. 2004. Final manuscript received: 29 Mar. 2004.

[1]The latency numbers are in cycles for uniform random traffic at 0.2 injection load on an 8×8 torus.

$\Theta = 8/7$ (57.2% of ideal) on NN. This tradeoff between locality (performance on benign patterns) and load balance (worst-case throughput) is inherent to oblivious routing algorithms [10], and, therefore, no oblivious balancing strategy performs optimally on both benign and difficult patterns.

Minimal adaptive routing algorithms (MADs) [4], [6], give optimal performance on NN traffic, but give poor throughput $\Theta = 1/3$ (62.3% of ideal) on TOR (Figure 1(b)). Achieving good performance on hard patterns like TOR requires routing some traffic non-minimally (the long way around the ring) to balance load.

Most previously reported non-minimal routing algorithms (NMADs) [1], [9] base their adaptive decisions on local congestion information. For example, in Chaotic routing [1] (CHAOS), packets always choose a minimal direction, if available. On the 8-ring, CHAOS performs optimally on NN ($\Theta = 2$), but falls far short of ideal ($\Theta = 0.36$) on TOR. CHAOS can misroute a single packet several times, alternating its path between both the clockwise and counterclockwise directions. While techniques can be employed to limit excessive misrouting [9], the fundamental shortcoming of such non-minimal adaptive routing algorithms still exists — they attempt to solve a global problem by making local decisions.

The DRB algorithm [5] is the only non-minimal algorithm we know of that senses global congestion and expands paths to incorporate non-minimal ones when the latency of minimal paths exceeds a threshold. However, it uses explicit notification packets to enable the source node to sense global congestion, which potentially wastes network bandwidth. In contrast, our work relies on implicit network backpressure to transfer congestion information back to the source nodes.

III. GAL: Globally Adaptive Load-Balanced Routing

A. GAL in a Ring

GAL senses global congestion using, at each source, a set of injection queues for each destination.[2] Each set has two queues - minimal and non-minimal. When a packet is received from the network interface, it is enqueued in the minimal queue for the packet's destination if that queue's length is less than a threshold, T, otherwise it is enqueued in the shorter of the two queues.[3]

GAL always routes NN traffic minimally, since the threshold of the minimal queue is never reached. As illustrated in Figure 2, GAL also routes TOR traffic minimally at low loads. Only when the minimal channels become saturated, at a load of 0.33 does the minimal queue length exceed the threshold. At this point, GAL starts to route some traffic non-minimally. As the load is increased, GAL routes progressively more traffic non-minimally. At saturation, the load is exactly balanced with $5/8$ of the traffic routing minimally and $3/8$ non-minimally. Thus, by making the global routing decision (minimal or non-minimal) using global congestion information (sensed by the injection queues) GAL is able to achieve optimal performance on both benign (NN) and difficult (TOR) traffic patterns, something no previously published routing algorithm has achieved.

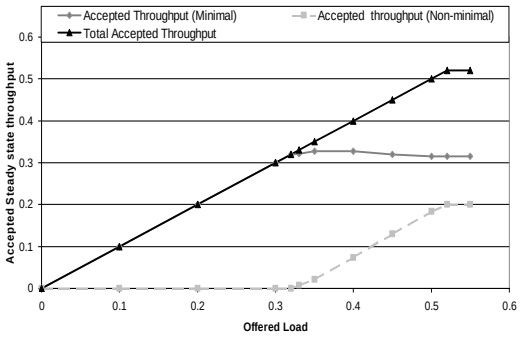

Fig. 2. GAL on tornado traffic on an 8 node ring

B. GAL Routing in Higher Dimension Torus Networks

In a torus with n dimensions, we divide the sets of possible routes from a source s to a destination d into 2^n *quadrants*, one for each combination of directions (one direction per dimension). For a 2 dimension network, there are 4 quadrants ($++$, $+-$, $-+$ and $--$) for each source-destination pair. GAL provides a separate injection queue for each *quadrant*. Figure 3(a) shows the 4 quadrants for the source-destination pair $(0,0),(2,3)$. Quadrant I is the minimal quadrant while the others are non-minimal.

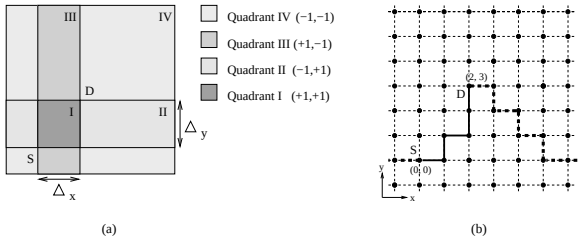

Fig. 3. (a) Quadrants in a k-ary 2-cube for a given source S (0,0) and destination D (2,3). (b) Example routes from S (0,0) to D (2,3) through the minimal quadrant I(+1,+1) and a non-minimal quadrant II(-1,+1).

For a 2-D network, there are $S = k^2$ sets, each set comprising 4 injection queues. When a packet is received from the source queue, its set is determined by its destination. Within that set, one injection queue (and therefore the quadrant to route in) is selected as follows: the queue associated with the quadrant having the smallest distance from the source to the destination whose occupancy is less than a threshold, T, is chosen. If all the queues have surpassed their threshold, then the shortest queue is selected.

Once the quadrant is selected, the packet is routed adaptively within that quadrant. A dimension i is *productive* if, the coordinate of the current node x_i differs from d_i. In other words, it is productive to move in that dimension since the packet is not already at the destination coordinate. At each hop, the router picks the productive dimension with the shortest

[2]In Section V we show that GAL can be implemented with just two sets of injection queues at each source.

[3]In order to stabilize GAL for all traffic patterns, we need to adaptively vary the threshold value which we discuss in Section V

output queue to advance the packet. Two possible routes from $s = (0, 0)$ to $d = (2, 3)$ are shown in bold and dashed as examples of routing in quadrants I and II.

C. Virtual Channels and Deadlock

Our implementation of GAL requires 3 virtual channels (VCs) per unidirectional physical channel to achieve deadlock freedom in the network. This is an extension of the scheme proposed in the $*$-channels algorithm [6] for wormhole flow control developed for the non-minimal GOAL algorithm. For a proof of deadlock freedom for the scheme, refer to [8].

IV. Performance evaluation

In this section we compare the performance of the four routing algorithms described in Table II based on the figures of merit introduced in Section I.

TABLE II

THE ROUTING ALGORITHMS EVALUATED IN THIS PAPER

Name	Description
CHAOS	The Chaos routing algorithm [1].
MIN AD	Minimal Adaptive (or the $*$-channels algorithm) - route in the minimal quadrant, routing adaptively within it [6].
GOAL	Globally Oblivious Adaptive Locally - choose a quadrant Q to route in according to a weighted probability distribution, then route within Q adaptively [8].
GAL	Globally Adaptive Load-Balance - Adaptively choose a quadrant Q to route in at the source node by sensing global congestion, then route within Q adaptively.

A. Experimental Setup

Measurements in this section have been made on a cycle-accurate network simulator in which the routing decision takes one cycle. Each node has an infinite source queue to model the network interface. For GAL, each node also has $S = 64$ sets of injection queues for an 8×8 torus. Each set has 4 injection queues (with 128 flits each) corresponding to each of the 4 quadrants. The threshold value, T, for GAL is kept at 2 flits for each minimal injection queue. The total buffer resources are held constant across all algorithms. All contention is resolved using age-based arbitration. All latency numbers presented are measured since the time of birth of the packets and include the time spent by the packets in the source queues. We have simulated an 8-ary 2-cube and a 16-ary 2-cube, but present only the results for the 8-ary 2-cube topology due to space constraints. The results obtained for the 16-ary 2-cube topology follow the same trends. Finally, each packet is assumed to be one flit long to separate the routing algorithm study from flow control issues.

B. Throughput on Benign and Hard Traffic

Figure 4 shows the saturation throughput (normalized to throughput on uniformly random traffic) for each algorithm on each traffic pattern of Table III. Two benign traffic patterns are shown in Figure 4(a) while four adversarial patterns are shown in Figure 4(b) with an expanded vertical scale. The figure shows that GAL is the only algorithm that gives best

TABLE III

TRAFFIC PATTERNS FOR EVALUATION OF ROUTING ALGORITHMS

Name	Description
NN	Nearest Neighbor - each node sends to one of its four neighbors with probability 0.25 each.
UR	Uniform Random - each packet is sent to a random destination.
BC	Bit Complement - (x, y) sends to $(k - x - 1, k - y - 1)$.
TP	Transpose - (x, y) sends to (y, x).
TOR	Tornado - (x, y) sends to $(x + \frac{k}{2} - 1, y)$.
WC	Worst-case - the traffic pattern that gives the lowest throughput by achieving the maximum load on a single link.

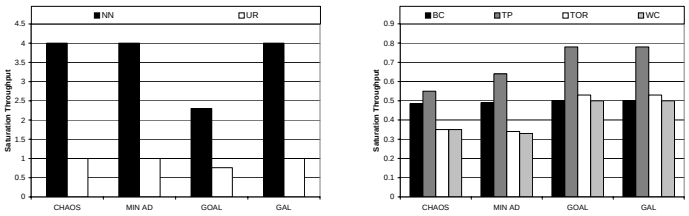

Fig. 4. Saturation throughput of five algorithms on an 8-ary 2-cube for (a) two benign traffic patterns and (b) four adversarial traffic patterns.

performance on both the benign and the adversarial traffic. It becomes a GOAL-like load balancing algorithm and matches the throughput of GOAL on adversarial traffic. At the same time it behaves minimally on benign patterns and matches the performance of MIN AD on them.

C. Throughput on Random Permutations

In order to evaluate the performance of the algorithms for the *average* traffic pattern, we measured the performance of each algorithm on 1,000 random permutations.[4] We compare the performance of each algorithm against the ideal case evaluated by solving a maximum concurrent flow problem over each traffic pattern.

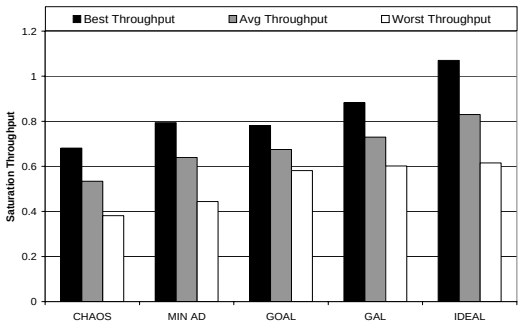

Fig. 5. Throughput on 1000 random traffic permutations

The figure shows that over the 1,000 permutations, GAL has top performance in the best, average and worst-case throughput — exceeding the worst-case throughput of GOAL and achieving 98% of the ideal worst-case throughput for

[4]These 10^3 permutations are selected from the $N! = k^n!$ possible permutations on an N-node k-ary n-cube.

this sampling. GAL gives 89% average throughput compared to ideal. Part of this gap is because the ideal throughput is evaluated assuming ideal flow control while GAL uses a realistic flow control mechanism.

V. Discussion

A. Stability

GAL requires an adaptive threshold T to achieve stability on benign traffic patterns. Without an adaptive threshold, congestion caused by load imbalance cannot be distinguished from congestion caused by saturation. On UR traffic at loads in excess of saturation, for example, the minimal injection queue will exceed T and some traffic will start to be misrouted incorrectly. This misrouted traffic increases channel load, and hence reduces throughput.

Such misrouting of benign traffic patterns can be avoided by adapting the threshold T. T is incremented (decremented) whenever the total number of packets departing from all queues in a set decreases (increases or stays level) over a window of time. This adaptation method senses the drop in throughput caused by incorrectly misrouting a benign pattern at high load and increases the threshold to prevent such misrouting. When the load drops the threshold returns to its original value.

B. Subsetting the Injection Queues

GAL can be implemented with a much smaller number of queues without sacrificing significant performance. Rather than provide a separate set of queues for each destination, we group destinations together, providing a separate set of queues for each group of destinations. In Figure 6 we compare the performance of three groupings to the full 64 sets of queues: 1 set — all destinations are grouped together, 2 sets — far destinations (farther than $k/4$ in any dimension), and near destinations, 4 sets — grouped by minimal direction. We evaluate performance on mixes of traffic patterns (since a permutation would get ideal performance with a single set of queues). The figure shows that two sets almost matches the ideal performance. For the NN/TOR mix, near/far partitioning exactly separates the two traffic patterns. Even for the UR/TOR mix, where near/far cannot exactly separate the two traffic patterns, two sets still does quite well. All results are identical for permutation traffic and for the evenly balanced NN and UR patterns. Hence, with only 2 sets, GAL achieves performance matching that of 64 sets.

VI. Conclusion

Globally-Adaptive Load-Balanced Routing (GAL) is a new routing algorithm for k-ary n-cube networks that adapts globally by sensing global congestion using injection queues at the source node. At low loads and on benign traffic patterns, GAL routes all traffic minimally and thus matches the low latency and high throughput of minimal routing algorithms on such 'friendly' traffic. On adversarial traffic patterns, GAL routes minimally at low loads and then switches to non-minimal routing as congestion is detected by the injection queues.

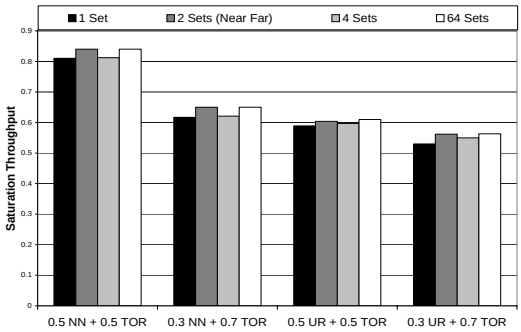

Fig. 6. Sub-setting the injection queue sets.

At saturation, GAL matches the throughput of load-balanced oblivious routing. GAL combines the best features of minimal algorithms (low latency at low load) and obliviously load balanced algorithms (high saturation throughput on adversarial patterns).

Unlike oblivious algorithms, adaptive routing algorithms are characterized by both a steady-state and a transient response. The transient response of GAL depends not only on the routing algorithm, but also on the details of per-channel flow control. In particular, short per-node queues that give *stiff* backpressure give more rapid response to traffic transients. Fully characterizing the transient response of GAL and other adaptive routing algorithms remains an open question.

Acknowledgment

We would like to thank Steve Scott of Cray Inc. for suggesting an improvement to our algorithm.

References

[1] K. Bolding, M. L. Fulgham, and L. Snyder, "The case for chaotic adaptive routing," *IEEE Transactions on Computers*, vol. 46, no. 12, pp. 1281–1291, 1997.

[2] W. J. Dally, "Performance analysis of k-ary n-cube interconnection networks," *IEEE Transactions on Computers*, vol. 39, no. 6, 1990.

[3] W. J. Dally, P. Carvey, and L. Dennison, "Architecture of the Avici terabit switch/router," in *Proceedings of Hot Interconnects Symposium VI*, August 1998, pp. 41–50.

[4] J. Duato, "A new theory of deadlock-free adaptive routing in wormhole networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 12, pp. 1320–1331, 1993.

[5] D. Franco, I. Garces, and E. Luque, "A new method to make communication latency uniform: Distributed routing balancing," in *ACM International Conference on Supercomputing (ICS)*, Greece, 1999.

[6] L. Gravano, G. Pifarre, G. Pifarre, P. Berman, and J. Sanz, "Adaptive deadlock- and livelock-free routing with all minimal paths in torus networks." *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, no. 12, pp. 1233–1252, Dec. 1994.

[7] S. Scott and G. Thorson, "The cray t3e network: adaptive routing in a high performance 3d torus," in *Proceedings of Hot Interconnects Symposium IV*, Aug. 1996.

[8] A. Singh, W. J. Dally, A. K. Gupta, and B. Towles, "GOAL: A load-balanced adaptive routing algorithm for torus networks," in *Proc. 30th Annual International Symposium on Computer Architecture ISCA'03*, San Diego, California, 2003.

[9] M. S. Thottethodi, A. R. Lebeck, and S. S. Mukherjee, "BLAM: A high-performance routing algorithm for virtual cut-through networks," in *International Parallel and Distributed Processing Symposium (IPDPS)*, Nice, France, April 2003.

[10] B. Towles and W. J. Dally, "Throughput-centric routing algorithm design," in *Proceedings of the ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, San Diego, California, June 2003.