# Globally Adaptive Load-Balanced Routing on k-ary n-cubes

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 that we refer to as *Globally Adaptive Load-balance (GAL)*. Unlike previous adaptive routing algorithms that make routing decisions based on local information (typically channel queue depth), GAL senses congestion globally 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. The use of global information enables GAL to achieve the performance (latency and throughput) of minimal adaptive routing on benign traffic patterns while performing as well as the best obliviously load-balanced routing algorithms (Valiant or GOAL) on adversarial traffic.**

**We compare GAL to four other previously published routing algorithms on six figures of merit - throughput on benign, adversarial and Hot-Spot traffic; average throughput of a sample of 1000 random permutations; latency at low injection load and stability post saturation. GAL is the only algorithm that has top performance in each category.**

## I. Introduction

Torus or $k$-ary $n$-cube interconnection networks [1] provide high bisection bandwidth, relatively low diameter, and high path diversity while keeping channel lengths physically short. These desirable properties have made torus networks popular in many applications including switch and router fabrics [2], for processor-memory interconnect [3], and for I/O interconnect [4].

Torus networks have high *path diversity*, offering many alternative paths between the source of a message and its destination. The routing algorithm employed by a torus network chooses among these alternative paths to select a particular path for each packet that traverses the network. A good routing algorithm provides low latency, balances load to provide high throughput on adversarial traffic patterns, and gracefully handles momentary overloads to provide stability.

Traffic patterns in an interconnection network may be benign or adversarial. A benign traffic pattern, such as uniform random[1], naturally balances load across channels and hence is easy to route. An adversarial traffic pattern, such as tornado[2], on the other hand, loads channels non-uniformly and requires a more sophisticated routing algorithm to achieve good load balance.

Many interconnection network applications require a routing algorithm that provides high throughput on *adversarial* traffic patterns. In an Internet router, for example, there is no backpressure on input channels so the interconnection network used for the router fabric must handle worst-case traffic at line rate, or packets will be dropped. Similarly, many I/O networks must provide guaranteed throughput on all traffic patterns between host and disk nodes. Some multicomputer applications are characterized by random permutation traffic[3]. This arises when operating on an irregular graph structure or on a regular structure that is randomly mapped to the nodes of the machine. To have predictable performance across all permutations, such applications require a network that has high throughput on adversarial patterns.

An ideal routing algorithm routes packets along minimal paths when possible to keep latency low. At the same time, such an algorithm must balance channel load to achieve high throughput. Such load balance often requires non-minimal routing. A key question in the design of a routing algorithm is when to route a packet minimally and when to route non-minimally.

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. GAL adapts globally by maintaining a separate injection queue for each *quadrant*[4] and each *set* of destinations. Each quadrant queue senses congestion in its respective quadrant. As long as the queue for the minimal quadrant remains below a threshold, indicating that there is no congestion in that quadrant, all traffic is routed minimally. If the queue for the minimal quadrant is above threshold, the packet is routed in the quadrant that has the shortest injection queue — balancing load. Once a quadrant has been selected for a packet, that packet routes only in that quadrant, always reducing the distance to the destination. At each step along the path, the path is adapted locally by choosing the next hop in the minimal quadrant with the shortest *channel* queue — which avoids local congestion.

The GAL algorithm comes close to our ideal routing algorithm. At low traffic rates, when there is no congestion,

[1]With uniform random traffic, each source randomly chooses the destination of each packet.

[2]With tornado traffic, each source sends all packets to the node $k/2 - 1$ hops away in one dimension.

[3]With Random permutation traffic each node sends all messages to a single, randomly-selected node. This should not be confused with uniform random traffic in which each message is sent to a different randomly selected node.

[4]A *quadrant* consists of all paths from a given source node that travel in a single direction in each dimension. For example, in a 2-D torus, the $(+,-)$ quadrant consists of all paths that travel in the $+x$ and $-y$ directions.

it routes all traffic along minimal paths — keeping latency at a minimum. Only for adversarial patterns at high traffic rates does the length of the minimal quadrant queue exceed its threshold. Only in these cases does the algorithm route non-minimally, and then 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 high throughput.

Previous work has attempted to address the issue of providing high throughput on adversarial patterns while minimizing latency. Valiant's randomized algorithm [5] (VAL) gives good performance on worst-case traffic, but at the expense of completely destroying locality. Thus VAL gives very poor performance on local traffic and greatly increases latency. The Chaos routing algorithm [6] employs randomization to misroute from a shared queue of packets in each node when the network becomes congested. However, the misrouting decision is very localized and does not address the global load imbalance caused by adversarial traffic patterns. Minimal adaptive routing [7], [8] also suffers from this global load imbalance. The RLB and GOAL algorithms, [9], [10] globally balance load by obliviously routing a certain fraction of traffic along non-minimal paths. Of these algorithms, RLB suffers on worst-case traffic because it cannot adapt locally. GOAL matches the performance of Valiant on adversarial patterns, and significantly outperforms Valiant on local patterns. GOAL, however, underperforms minimal algorithms at low traffic levels and on local traffic patterns because it routes a fixed fraction of traffic non-minimally, even at low loads and for benign traffic patterns.

| | VAL | CHAOS | MIN AD | GOAL | **GAL** |
|---|---|---|---|---|---|
| $\Theta_{\mathrm{benign}}$ | F | A | A | C | **A** |
| $\Theta_{\mathrm{hard}}$ | B | C | C | A | **A** |
| $\Theta_{\mathrm{avg}}$ | D | C | B | A | **A** |
| $\Theta_{\mathrm{HS}}$ | F | A | A | A | **A** |
| **Latency** | F | A | A | C | **A** |
| **Stability** | P | F | P | P | **P** |

TABLE I
REPORT CARD FOR FIVE ROUTING ALGORITHMS

The performance of GAL is summarized and compared to four previous routing algorithms in the report card of Table I. The report card qualitatively compares the performance of the five routing algorithms on four throughput ($\Theta$) measures, latency, and stability. On benign traffic patterns (such as nearest neighbor or uniform random), the GAL injection queues remain below threshold. Thus, GAL routes all benign traffic minimally and thus matches the high throughput ($\Theta_{\mathrm{benign}}$) and low latency (Latency) of minimal algorithms (which all receive an "A" grade). GOAL has somewhat lower ("C") and VAL gives much lower ("F") performance.

On hard traffic patterns (such as tornado) operating near saturation, GAL's injection queues exceed threshold and the global adaptation matches the ideal load balance of GOAL. Thus, both achieve highest throughput ($\Theta_{\mathrm{hard}}$) on these patterns ("A"s). VAL has slightly lower performance ("B"), and minimal algorithms achieve only half of 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 ($\Theta_{\mathrm{avg}}$). All of the adaptive algorithms (all but VAL) handle hot spots ($\Theta_{\mathrm{HS}}$) well. Finally, all algorithms except CHAOS are stable and get a passing grade "P" on stability. We discuss the stability of GAL in Section IV.

The remainder of this paper describes the GAL algorithm in more detail and compares its performance to other routing algorithms. Section II describes the shortcomings of different adaptive schemes. Section III illustrates how GAL overcomes these shortcomings and describes the algorithm in detail. We measure the performance of GAL in Section IV and compare its performance to existing routing algorithms. In Section V, we discuss issues like the effect of reducing the number of injection queues. Finally, Section VI concludes.

## II. PREVIOUS ADAPTIVE ROUTING ALGORITHMS

Existing adaptive routing algorithms for interconnection networks fall into three basic categories[5]: minimal, non-minimal, and globally oblivious, locally adaptive. To understand the tradeoffs inherent in these three design points, we consider routing both a benign and a difficult traffic pattern on an 8 node ring network (Figure 1).

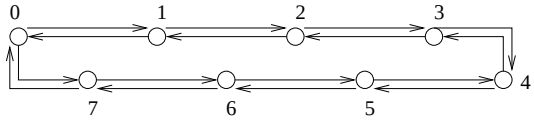



Fig. 1. An 8 node ring (8-ary 1-cube).

For the benign pattern, nearest neighbor (NN), all traffic from node $i$ is distributed equally amongst its neighboring nodes (half to node $i-1$ and half to node $i+1$). By simply routing traffic along shortest paths, throughput [6] on the NN pattern is maximized ($\Theta = 2$).

For the difficult pattern, tornado (TOR), 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 2. The ideal throughput for tornado is $\Theta = 8/15$. No category performs optimally on both of these simple patterns.

### A. Minimal adaptive routing

In minimal adaptive routing algorithms (MADs), packets are only routed along shortest (minimal) paths between source and destination nodes. Current MADs [13], [14], [7] introduce adaptivity by building a packet's path incrementally — at each intermediate node, the packet chooses any channel along a

[5]More detailed taxonomies of adaptive routing algorithms have been made [11], [12], but our categorization focuses on how paths are selected to balance load, which is central to this paper.

[6]The throughput is defined as the reciprocal of the worst-case link load normalized to the network capacity (the maximum load that the bisection of the network can sustain for uniformly distributed traffic and is given by $k/8$ (See [10] for details)).

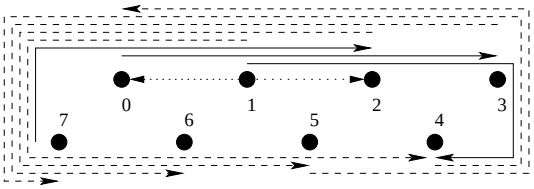



Fig. 2. Optimally routed tornado traffic. Load is shown for a clockwise and a counter clockwise link depicted in dotted lines. The dashed lines contribute a link load of $\frac{3}{8}$ while the solid lines contribute a link load of $\frac{5}{8}$. All links are equally loaded with load = $\frac{15}{8}$.

minimal path to the destination. If there are multiple minimal directions, the next channel is selected based on availability, buffer utilization, or some other channel congestion metric. As reflected in Table I, the ability of MADs to reroute around local congestion results in good average throughput and the use of only minimal paths gives low latency. However, for difficult traffic patterns, restricting routes to be minimal can significantly degrade throughput.

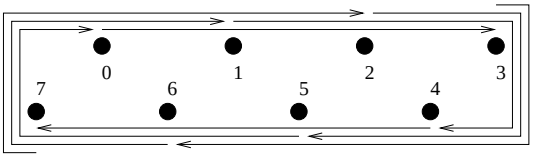



Fig. 3. Minimally routed tornado traffic. Clockwise link load is 3. Counter clockwise link load is 0.

On our example of the ring, MAD performs optimally when routing the benign NN pattern. All traffic is routed minimally and the throughput is $\Theta = 2$. However, for the tornado pattern, minimal routing results in all traffic traversing the clockwise links, leaving the counterclockwise links completely idle (Figure 3). The clockwise link from node $i$ carries traffic from nodes $i$, $i-1$, and $i-2$, thus the throughput is only $\Theta = 1/3$ or approximately $62.3\%$ of ideal.

### B. Non-minimal adaptive routing

Non-minimal adaptive routing algorithms (NMADs) attempt to avoid throughput degradation on difficult traffic patterns by incorporating non-minimal paths. As with MADs, practical NMADs [6], [15], [16] base their adaptive decisions on local congestion information. For example, in Chaotic routing [6] (CHAOS), packets always choose a minimal direction, if available. Otherwise, they are momentarily buffered and eventually misrouted in a non-minimal direction (further from their destination).

Returning to the ring, CHAOS performs optimally on the NN pattern ($\Theta = 2$), but only slightly improves over minimal routing on the TOR pattern ($\Theta = 0.36$). Although packets should ideally choose either the clockwise or counterclockwise direction around the ring to optimally balance the TOR pattern (Figure 2), 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 this excessive misrouting, such as in the BLAM algorithm [15], the fundamental shortcoming of current non-minimal adaptive routing algorithms still exists — they incorporate no concept of global load balance into their routing decisions.

### C. Globally oblivious, locally adaptive routing

Recognizing that adaptivity based on local congestion results in poor global load balance, globally oblivious, locally adaptive routing separates the tasks of local and global balance. Local balance is still achieved by incrementally selecting channels in the packet's path based on congestion. However, global balance is achieved obliviously (independent of network congestion) by spreading packets over different portions of the network.

In the ring example, the GOAL algorithm [10] balances global load by sending a packet non-minimally with probability $d/k$, where $d$ is the minimal distance between the packet's source and destination and $k$ is the radix of the network ($k = 8$ for our ring). Then, for the the TOR pattern, $3/8$ of the traffic is sent non-minimally and the remaining $5/8$ is sent minimally, optimally balancing load. For the NN pattern, however, $1/8$ is sent non-minimally, reducing the throughput to $\Theta = 8/7$ or only $57.2\%$ of optimal. This tradeoff between locality (throughput on benign patterns) and worst-case throughput is inherent to oblivious routing algorithms [17], and, therefore, no oblivious balancing strategy performs optimally on both benign and difficult patterns.

## III. GAL: Globally Adaptive Load-Balanced Routing

### A. A case for Globally Adaptive Load-balance

In order to sense the global load balance information in the previous ring example and adapt to it, let us have $8$ sets of injection queues at each source — each set corresponding to a destination node. Each set has two queues - a minimal queue and a non-minimal one. When a packet is received from the network interface, it is enqueued in the minimal queue of the set corresponding to its destination if the occupancy of that queue is below a certain threshold, $T$, else it is enqueued in the shorter of the two queues. With this scheme, routing NN would always be minimal — i.e. the threshold of the minimal queues will never be overshot and so the throughput will be $\theta = 2$. When we run this scheme on TOR, and increase the injected load, initially all the packets will be sent minimally. However, when $0.33$ load is accepted, the minimal direction will be saturated and the threshold will be overshot. The algorithm will now become non-minimal and will start sending the rest of the traffic the long way around.

Figure 4 shows how this scheme works minimally until an accepted throughput of $0.33$ and then becomes non-minimal until the total throughput is the optimal value of $0.53$. At this point, it sends exactly $5/8$ the traffic minimally and $3/8$ the traffic non-minimally. Also, while the saturation throughput is the same as GOAL for TOR, GAL delivers packets with lower latency prior to saturation. At any load GOAL obliviously sends $3/8$ the traffic non-minimally, thus incurring unnecessarily high latency. Hence, by sensing the global congestion

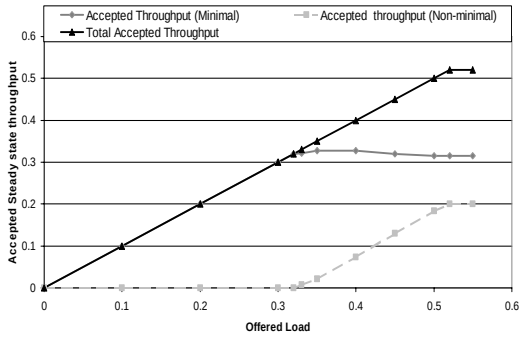



Fig. 4. GAL on tornado traffic on an 8 node ring

adaptively, we achieve the ideal performance on both the hard and benign patterns. We call this scheme Globally Adaptive Load-Balanced[7] (GAL) routing.

### *B. GAL Routing in higher dimension torus networks*

Suppose we have to route a packet from a source node $s = \{s_1, \ldots, s_n\}$ to the destination node $d = \{d_1, \ldots, d_n\}$. Unlike the one dimensional case, where there are just two possible paths for each packet — one short and one long, there are many possible paths for a packet in a multi-dimensional network. As in GOAL, we divide these possible paths based on the directions ($+$ or $-$) in each dimension. Hence, for a 2 dimension network, there are 4 quadrants ($++$, $+-$, $-+$ and $--$) for each source-destination pair. Figure 5 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. In order to segment the injection queues just like we did in the 1 dimension case, we must have $S = k^n$ sets (one per destination) of $2^n$ injection queues each for a $k$-ary $n$-cube in each source node.

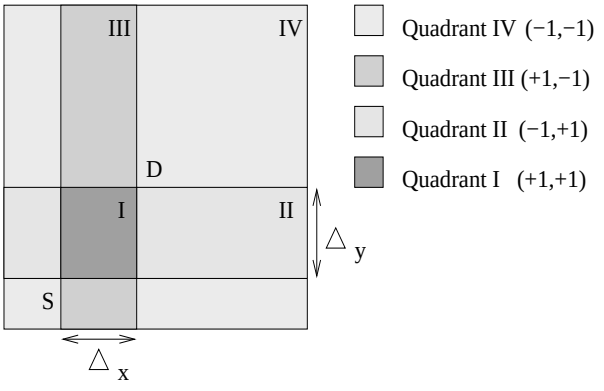



Fig. 5. Quadrants in a $k$-ary 2-cube for a given source S (0,0) and destination D (2,3).

Figure 6 shows the setup for each node for a 2-$D$ network. There is an infinite source queue that models the network interface. There are $S = k^2$ sets, each set comprising 4 injection queues. The injection unit receives a packet from the source queue and enqueues it into the minimal queue of the set corresponding to the destination of the packet if the occupancy of that queue is below a threshold, $T$. Otherwise, it enqueues it into the shortest of all the queues in that set. Once the injection queue is decided, the packet is routed entirely within the corresponding quadrant.

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

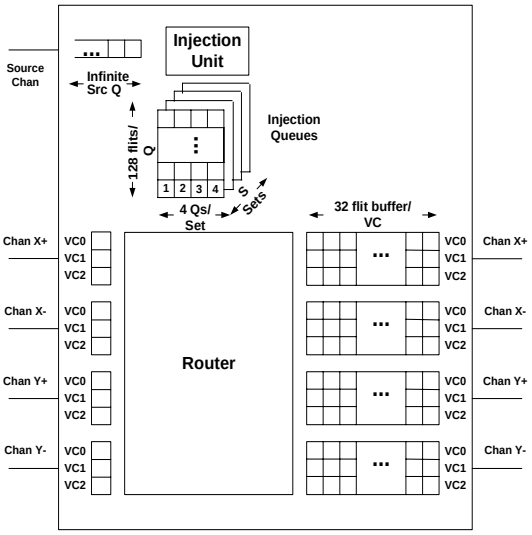



Fig. 6. GAL node

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 output queue to advance the packet.

For example, consider the case above where we are routing from $s = (0,0)$ to $d = (2,3)$ in an 8-ary 2-cube network. Suppose the minimal queue occupancy corresponding to the destination set is below the threshold. Then the algorithm injects the packet in the minimal queue and sends along quadrant I $(+1,+1)$. One possible route of the packet is shown in bold in Figure 7. On the first hop, the productive dimension vector is $p = (1,1)$ that is both the $x$ and $y$ dimensions are productive. Suppose the channel queue in the $x$ dimension is shorter so the packet proceeds to node (1,0). At (1,0) $p$ is still (1,1) so the packet can still be routed in either $x$ or $y$. At this point, suppose the queue in the $y$ dimension is shorter, so the packet advances to node (1,1). At (1,1) $p$ is still (1,1) and this time the route is in $x$ to (2,1). At this point, the packet has reached the destination coordinate in $x$ so $p = (0,1)$. Since the only productive dimension is $y$, the remaining hops are made in the $y$ dimension regardless of queue length. A second possible route is shown in dashed if the injection queue threshold was exceeded and the shortest queue was that corresponding to quadrant II $(-1,+1)$.

### *C. Virtual Channels and Deadlock*

As shown is Figure 6, our implementation of GAL requires 3 virtual channels (VCs) per unidirectional physical channel to

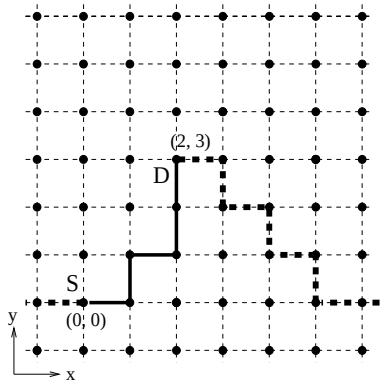



Fig. 7. Example route from S (0,0) to D (2,3) through the minimal quadrant (+1,+1) and a non-minimal quadrant (-1,+1).

achieve deadlock freedom in the network. This is an extension of the scheme proposed in the $*$-channels algorithm [7] for wormhole flow control developed for the non-minimal GOAL algorithm. For a proof of deadlock freedom for the scheme, refer to [10].

## IV. Performance evaluation

| Name | Description |
|---|---|
| VAL | Valiant's algorithm - route to a random node $q$ (phase 1) anywhere in the network, then to the destination (phase 2). Deadlock is avoided using 2 subnetworks (for each phase) of 2 VCs each [5]. |
| CHAOS | The Chaos routing algorithm. Deadlock is avoided using deflection routing [6]. |
| MIN AD | Minimal Adaptive (or the $*$-channels algorithm) - always route in the minimal quadrant, routing adaptively within it. Deadlock is avoided using 3 VCs [7]. |
| GOAL | Globally Oblivious Adaptive Locally [10] - choose a quadrant Q to route in according to a weighted probability distribution, then route within Q adaptively. Deadlock is avoided using 3 VCs [7]. |
| 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. Deadlock is avoided using 3 VCs [7]. |

TABLE II
THE ROUTING ALGORITHMS EVALUATED IN THIS PAPER.

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

### A. Experimental Setup

Measurements in this section have been made on a cycle-accurate network simulator that models the pipeline of each router as described in [18]. Routing is assumed to take one 20 FO4 pipeline stage for each algorithm as supported by gate-level designs. All the assumptions regarding the node model for GAL stated in Section III hold for the experimental setup. Figure 6 shows the experimental setup for a node for GAL. Each node has an infinite source queue to model the network interface and there are $S = 64$ sets of injection queues for an $8 \times 8$ torus. Each set has $4$ injection queues (with $128$ flits each) corresponding to each of the $4$ quadrants. The router is output queued and queues packets into one of the 3 (32 flit deep) VC buffers per physical output channel. Each packet is assumed to be one flit long to separate the routing algorithm study from flow control issues. 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 (the product of the number of VCs and the VC channel buffer depth is kept constant). For the CHAOS algorithm which does not use VCs, we increase the number of buffers in the shared queue of each node. All contention is resolved using age-based arbitration, always giving priority to a packet with an older time-stamp since injection. 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 two topologies, 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.

All simulations were instrumented to measure steady-state performance with a high degree of confidence. The simulator was warmed up under load without taking measurements until none of the queue sizes changed by more than $1\%$ over a period of $100$ cycles. Once the simulator was warmed up, a sample of injected packets were labeled for measurement. This sample size was chosen to ensure that measurements are accurate to within $3\%$ with $99\%$ confidence. The simulation was then run until all labeled packets reached their destinations.

### B. Throughput on Benign and Hard traffic

We first compare the throughput of the five algorithms on the benign and adversarial traffic patterns described in Table III.

| 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 randomly selected destination node. |
| 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. |

TABLE III
TRAFFIC PATTERNS FOR EVALUATION OF ROUTING ALGORITHMS

The first two traffic patterns in Table III, NN and UR, are *benign* in the sense that they naturally balance load and hence give good throughput with minimal routing algorithms. The next three patterns, BC, TP, and TOR, are *adversarial* patterns that cause load imbalance. These patterns have been used in the past to stress and evaluate routing algorithms [9], [19], [6], [11], [7]. Finally, the worst-case pattern is the traffic pattern that gives the lowest throughput. In general, the worst-case pattern may be different for different routing algorithms. The worst-case throughput for VAL is the provably optimal value of $0.5$ as shown in [10]. Unlike oblivious routing algorithms,

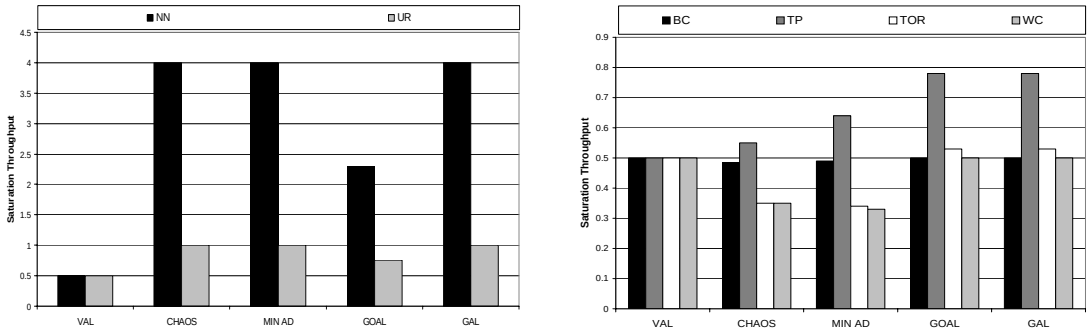



Fig. 8. Comparison of saturation throughput of five algorithms on an 8-ary 2-cube for (a) two benign traffic patterns and (b) four adversarial traffic patterns.

whose worst-case throughput can be analytically calculated [19], there is no known analytical method to determine the worst-case traffic pattern for adaptive routing algorithms. The worst-case reported is the worst throughput we have come across so far for each of the $4$ adaptive algorithms evaluated.

Figure 8 shows the saturation throughput for each algorithm on each traffic pattern. The two benign traffic patterns are shown in Figure 8(a) while the four adversarial patterns are shown in Figure 8(b) with an expanded vertical scale. The figure shows that GAL is the only algorithm that gives best 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.[8] Histograms of saturation throughput over these permutations for the five algorithms are shown in Figure 9. The histogram for VAL is just a delta function at $0.5$ because it gives the same throughput for all traffic patterns. All the other routing algorithms have bell-shaped histograms. The last histogram shows the ideal throughput for each of the random permutations evaluated by solving a maximum concurrent flow problem over the traffic patterns constrained by the unit capacity of every link as described in [20]. The best, average and worst-case throughput in this experiment for each of the algorithms and the ideal case are presented in Figure 10.

The figures show that over the 1,000 permutations, GAL has top performance in the best, average and worst-case throughput — matching the worst-case throughput of GOAL and achieving $96\%$ of the ideal worst-case throughput for this sampling. GAL gives $89\%$ average throughput compared to the ideal average throughput. Part of this gap is because the ideal throughput is evaluated assuming ideal flow control while GAL uses a realistic flow control mechanism.

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

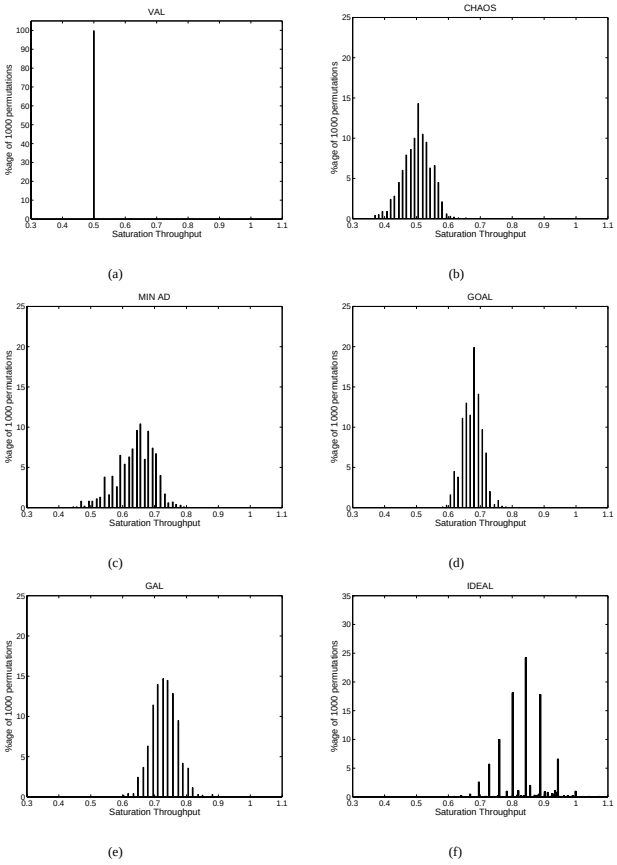



Fig. 9. Histograms for the saturation throughput for $10^3$ random permutations. (a) VAL, (b) CHAOS, (c) MIN AD (d) GOAL (e) GAL (f) IDEAL

### D. Latency

To compare the latency of the five algorithms, we computed latency histograms between a representative source-destination pair in a network loaded with uniform random traffic for each algorithm (as in [9], [10]). In an $8$-ary $2$-cube loaded with uniform traffic, the distance between a source and a destination node can range from $1$ to $8$ hops. In our experiments, we chose to measure the latency incurred by packets from a source to $3$ different destination nodes running uniform random traffic at $0.2$ offered load:

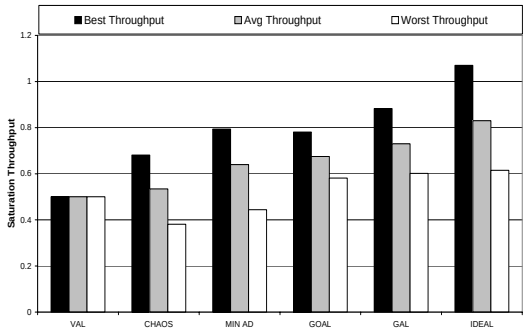



Fig. 10. Best case, average and worst case throughput for 1000 random traffic permutations

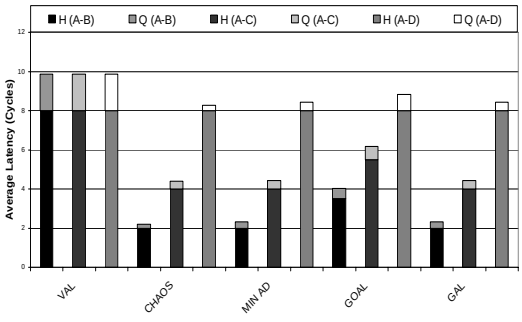



Fig. 12. Average total - hop (H) and queuing (Q) - latency for $10^4$ packets for 3 sets of representative traffic paths at 0.2 load.

- $A$ (0,0) to $B$ (1,1) - path length of 2 representing very local traffic.
- $A$ (0,0) to $C$ (1,3) - path length of 4 representing semi-local traffic.
- $A$ (0,0) to $D$ (4,4) - path length of 8 representing non-local traffic.

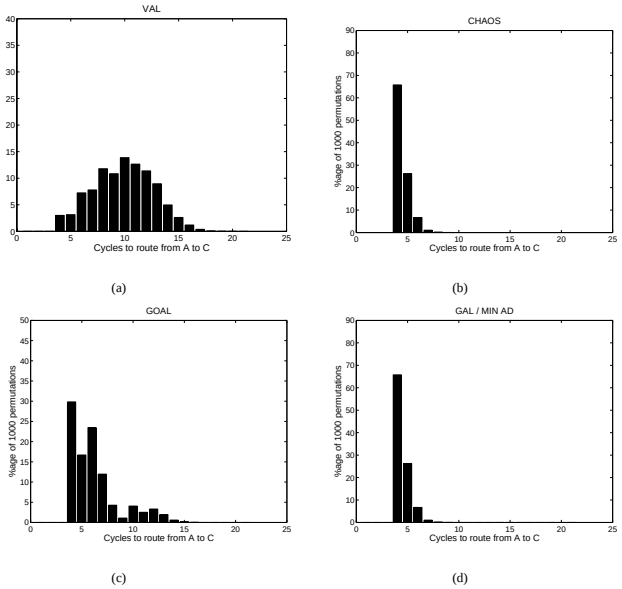



Fig. 11. Histograms for $10^4$ packets routed from node A(0,0) to node C(1,3). (a) VAL, (b) CHAOS, (c) GOAL, (d) GAL and MIN AD.

The histograms for semi-local paths (packets from $A$ to $C$) are presented[9] in Figure 11. Each histogram is computed by measuring the latency of $10^4$ packets for the test pairs. For all experiments, offered load was held constant at 0.2. The experiment was repeated for each of the five routing algorithms.

The latency, $T$, incurred by a packet is the sum of two components, $T = H + Q$, where $H$ is the hop count and $Q$ is the queuing delay. The average value of $H$ is constant while that of $Q$ rises as the offered load is increased. For a minimal algorithm, $H$ is equivalent to the Manhattan distance $D$ from source to destination. For non-minimal algorithms, $H \geq D$.

[9]The other sets of histograms are not presented due to space constraints.

The results for all the three representative paths are presented in Figure 12. At such a low load, the non-minimal algorithms, GAL and CHAOS, behave just like MIN AD and deliver packets with optimal latency. These 3 algorithms have a minimal hop count, $H = 4$, and the queuing delay $Q$ is exponentially distributed with a mean of approximately $0.45$ cycles. The result is a latency histogram with a peak at $H = 4$ that falls off exponentially.

The obliviously balanced GOAL has higher latency (40% more than minimal) and broader distributions than the minimal algorithms because it routes a fraction of the traffic in the non minimal quadrants. Depending on the quadrant chosen the hop count for a given packet is $H = 4$, 6, 10, or 12. With the weighted choice of quadrants the average hop count is $H = 5.5$ and the distribution is the superposition of four exponential distributions, one for each quadrant. This increase in latency compared to the minimal algorithms is one of the costs of providing high worst-case throughput by trying to balance load obliviously.

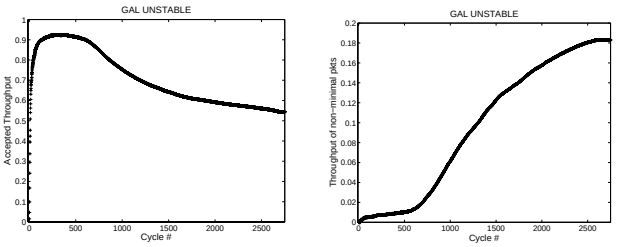
VAL has the highest latency and broadest distribution because it operates in two random phases. Depending on the choice of intermediate node the hop count $H$ can be any even number between 4 and 12. The average hop count is $H = 8$, and the broad distribution is the superposition of exponentially decaying distributions starting at each of these hop counts.

### *E. Stability*

An important property of any routing algorithm is its stability. An algorithm is stable if the accepted throughput remains constant even as the offered load is increased beyond the saturation point. Achieving stability can be challenging for non-minimal, adaptive routing algorithms because it is difficult to differentiate between congestion caused by load imbalance and congestion caused by saturation of a load balanced network. In the former case, rerouting traffic, possibly non-minimally, can alleviate this imbalance and increase throughput. However, in the load balanced, but saturated case, rerouting traffic may introduce imbalance and decrease throughput. GAL solves this instability problem by measuring changes in throughput and adjusting its queue thresholds. Changing these thresholds allows control over the fraction of packets sent non-minimally, which, in turn, is used to maintain stability. Before describing

the mechanisms and rules for these adjustments, we examine a simple example to demonstrate the necessity of controlling the fraction of packets sent non-minimally.

Consider the case of routing UR traffic with GAL, but using a fixed queue threshold. For an offered load beyond saturation, traffic will initially be routed minimally as the minimal queues are empty. Since the network cannot sustain all the traffic, backpressure will cause packets to back up into the minimal source queues. Although minimal routing perfectly balances load for UR traffic, the occupancy of these queues will eventually exceed the fixed threshold and some traffic will be routed non-minimally. At this point, throughput will begin to drop as channel bandwidth is wasted by non-minimal packets (Figure 13). If, instead, packets had not been allowed to route non-minimally, throughput would have remained stable.




(a) (b)

Fig. 13. Performance of GAL on UR traffic at 1.1 offered load when thresholds are fixed. (a) Throughput decreases over time and (b) the fraction of packets routed non-minimally increases.

To control the amount of packets routed non-minimally, GAL dynamically varies each threshold $T$ between a minimum value $T_{\min}$ and a maximum value $T_{\max}$ set by the queue depth. This is done independently for each minimal queue. Then, the threshold value limits the fraction of packets routed non-minimally — increasing $T$ decreases the likelihood that a packet is placed in a non-minimal queue and, when $T = T_{\max}$, all packets are routed minimally. To ensure stability, $T$ is incremented whenever a drop in throughput is observed. Revisiting our stability example, for any $T < T_{\max}$, some fraction packets will be routed non-minimally, reducing throughput and, thus, triggering an increase in $T$. This continues until $T = T_{\max}$ and stability is achieved. Because the traffic pattern may later change so that non-minimal routing is again productive, $T$ is decremented whenever throughput is constant or increasing.

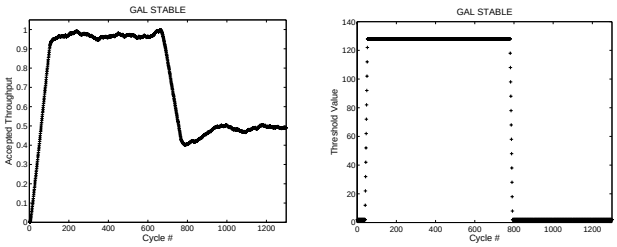
We approximate the throughput for each source-destination pair by instrumenting a running total of the drained packets over a window of $n_1$ cycles, $\overline{D}$, from all the injection queues in the corresponding set at each source node. Then, the change in this drainage, $\Delta D$, is computed over $n_2$ cycles:

$$\overline{D}_t = \sum_{i=0}^{n_1-1} D_{t-i} \qquad \Delta D_t = \overline{D}_t - \overline{D}_{t-n_2},$$

where $D_t$ is the number of departures from the queues in that set at time $t$. The estimated throughput is then used to update the threshold each $n_2$ cycles:

$$T_{i+1} = \begin{cases} \min(T_i + 1, T_{\max}) & \text{if } \Delta D_i < 0, \\ \max(T_i - 1, T_{\min}) & \text{if } \Delta D_i \geq 0. \end{cases}$$

Figure 14 shows an example of threshold adjustment. The test setup is as follows: we impose UR traffic at 1.1 injection load and then switch to TOR at 0.55 injection load on Cycle 600. The parameter values used are $T_{\min} = 2$, $T_{\max} = 128$, $n_1 = 50$ and $n_2 = 20$. Both loads are post saturation, but the accepted throughput is stable. This is because when $\Delta D$ becomes negative, the threshold is incremented. This keeps happening as more packets are injected. Finally, when $T = T_{\max}$, the algorithm routes the UR traffic strictly minimally and the accepted throughput almost reaches the optimal value of $1.0$. After cycle $600$, the traffic changes to TOR at 0.55 load. Still $\Delta D < 0$ and so $T = T_{max}$. Essentially now GAL acts like a minimal algorithm routing TOR traffic and so the accepted throughput drops much below the optimal value of $0.53$. After the lowest point, the throughput starts to level out again ($\Delta D = 0$) and so $T$ is decremented and as a result $\Delta D$ rises as traffic is sent non-minimally. This continues until $T = T_{\min}$ and the accepted throughput is stable for TOR traffic. Thus, GAL with the threshold adjustment scheme is stable for both benign and hard traffic patterns.




(a) (b)

Fig. 14. Performance of GAL with threshold variation post saturation. UR at 1.1 switches to TOR at 0.55 at Cycle 600.

### F. Performance on Hot-Spot traffic

Occasionally a destination node in an interconnection network may become oversubscribed. This may occur in a switch or router due to a transient misconfiguration of routing tables. In a parallel computer such a *hot spot* occurs when several processors simultaneously reference data on the same node.

We evaluate the performance of five algorithms on hot-spot traffic by using the *hot-spot pattern* used in [10]. We first select a background traffic pattern, BC, on which all the algorithms give similar performance. On top of BC, five nodes[10] are selected which are five times more likely to be chosen as destinations than the other nodes. In the resulting pattern, HS, the five hot nodes are oversubscribed and hence it is not an admissible traffic pattern[11].

[10]These five hot-spot nodes are chosen very close to each other to stress the adaptivity of the algorithms.

[11]An *admissible* traffic pattern is one whose destination matrix has its row and column entries summing to at most $1.0$ [10].

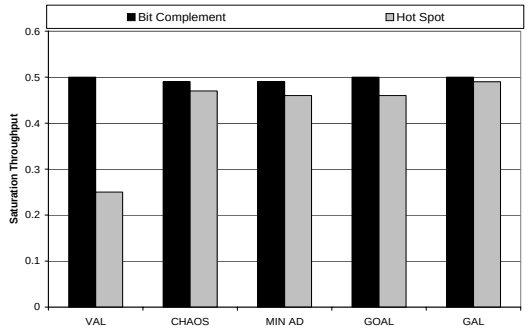



Fig. 15. Saturation Throughput for the Hot-Spot traffic pattern and the background Bit Complement pattern.

Figure 15 shows the performance of each routing algorithm on the hot-spot pattern and the background BC pattern. The adaptive algorithms, CHAOS, MIN AD, and GOAL have similar performance on hot-spot traffic. They clearly outperform the oblivious VAL because adaptivity is required to route around the congestion resulting from hot nodes. GAL once again does the best among all the algorithms, outperforming MIN AD by $6.5\%$. This is because the minimality inherent in MIN AD forces it to route some fraction through the clustered hot nodes. The globally load balancing GAL senses this congestion and routes traffic that would otherwise have to go headlong through the hot nodes with MIN AD, the long way around.

## V. Discussion

### A. Pairs of traffic patterns

In Section IV, we evaluated GAL's performance on permutation and benign traffic patterns. In this subsection, we show how GAL reacts to a combination of interfering traffic permutations. We demonstrate that the optimal way to route a combination of permutations need not be a linear combination of optimally routing them independently (as done by oblivious algorithms). GAL adapts its routing of one traffic permutation sensing the congestion due to the other.

Consider two traffic patterns on an 8-ary 1-cube — TOR and *clockwise* Nearest Neighbor (NNc) in which each node $i$ sends to $i+1$. Lets combine the two traffic patterns such that the new Traffic is $0.1$TOR + $0.9$NNc. Let $\alpha$ be the fraction of TOR packets sent minimally and $\beta$ be the fraction of NNc packets sent minimally. The clockwise link load is $3(0.1\alpha) + 1(0.9\beta) = 0.3\alpha + 0.9\beta$. The counter clockwise link load is $5(0.1(1-\alpha)) + 7(0.9(1-\beta)) = 6.8 - 0.5\alpha - 6.3\beta$. If we were to route TOR independently, the optimal way would be to set $\alpha = 5/8 = 0.625$. Routing NNc optimally requires $\beta = 7/8 = 0.875$ of the traffic to be sent minimally and $1/8$ non-minimally. This is exactly what GOAL would do making the load on each link equal $0.975$ and the throughput $\theta = 1/0.975 = 1.026$. This is not the optimal throughput for the combined traffic.

To get the optimal throughput, we equate the clockwise and counter clockwise link loads that we derived above which gives us the equation $\beta = (6.8 - 0.8\alpha)/7.2$. We have additional constraints that both $\alpha$ and $\beta$ are in the range $[0, 1]$. Substituting $\beta$ in the load equation, we get $\text{load} = 0.2\alpha + 0.85$. To minimize load, we put $\alpha = 0$ which makes $\beta = 0.944$. Hence, $\text{load} = 0.85$ and throughput $\theta = 1.176$. Thus, in order to route this pattern optimally, we need to send *all* TOR traffic non-minimally and $0.944$ of the NNc traffic minimally. GAL does not route this traffic ideally as it cannot send the TOR traffic completely non-minimally. It *has* to send enough traffic minimally so that the threshold is exceeded before it can send non-minimally. However, by adapting to the congestion due to the other traffic pattern, GAL gives better performance than GOAL giving $\theta = 1.05$ with $\alpha = 0.51$ and $\beta = 0.89$.

### B. Subsetting the injection queues

As discussed in Section III, each GAL node has $S$ sets of injection queues. In order to keep the traffic to different destinations from interfering with each other, we keep $S = N$, the number of nodes in the network. However, this is not a scalable system as $N$ grows large. In this subsection, we evaluate the effect of subsetting $S$ to a more manageable number. Note that for permutation traffic, subsetting $S$ does not change any results since in any permutation, each source sends to exactly one destination and so only 1 set is used. For non-permutation traffic, especially for traffic where some traffic needs to be routed minimally while other non-minimally, a naive subsetting could lead to interference between the two resulting in the minimal traffic going non-minimally too. We try 3 different kinds of subsetting of $S$. At the very extreme, we can clump all the $S$ sets into just 1 set, i.e. every packet will be injected into the same set irrespective of its destination. The second method divides the sets into $S = 2$ sets, one set for *near* destinations and the other for *far* destinations. A destination belongs to the *far* set if its farther than the average distance ($k/4$) around the ring in any dimension from the source node. Otherwise, it belongs to the *near* set. The last way of subsetting is to divide $S$ into $4$ sets according to the minimal directions of the destinations. A destination belongs to one of the four sets ($++$,$+-$,$-+$,$--$) depending on its minimal directions from the source in each dimension.

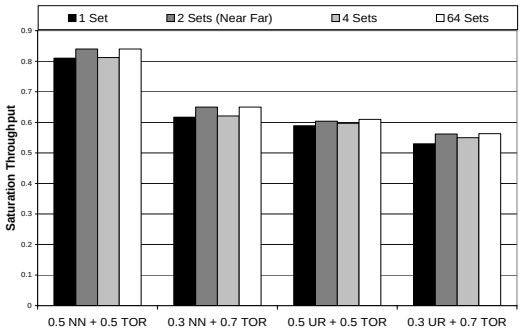



Fig. 16. Sub-setting the injection queue sets.

Figure 16 shows the effect of subsetting compared to the ideal case of $S = N = 64$ for $4$ different mixes of potentially interfering traffic patterns (NN with TOR and UR with TOR). It shows that $S = 2$ almost matches the ideal performance.

This is because for the NN and TOR mix, the sub-setting exactly separates the two traffic patterns into non-interfering queues. $S = 2$ also does very close to ideal wit the UR and TOR mix where it cannot exactly separate the two traffic patterns into different sets. $S = 4$ shows up to $5\%$ degradation in performance while $S = 1$ gives the poorest performance (up to $7\%$ from ideal) because of the interference effect described above. 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 with 64 sets.

### C. Livelock and Packet Ordering

Livelock is a condition whereby a packet keeps circulating within the network without ever reaching its destination. Freedom from such a critical condition must be guaranteed. Minimal algorithms like MIN AD guarantee livelock freedom with fair arbitration since each channel traversed by a packet reduces the distance to the destination. Valiant's algorithm is also deterministically livelock free since it is minimal in each of its phases. The Chaos scheme uses randomization to misroute from a shared queue of packets in each node during congestion. This randomization only ensures that the algorithm is *probabilistically* livelock free. Like GOAL, GAL, while non-minimal, provides deterministic freedom from livelock. Once a quadrant has been selected for a packet, the packet monotonically makes progress in that quadrant, reducing the number of hops to the destination at each step. Since there is no incremental misrouting, all packets reach their destinations after a predetermined, bounded number of hops.

The use of an adaptive routing algorithm can and will cause out of order delivery of packets. If an application requires in-order packet delivery, a possible solution is to reorder packets at the destination node using the well known sliding window protocol [21].

## VI. Conclusion

In this paper we have introduced Globally Adaptive Load-Balanced Routing (GAL), a new algorithm for routing in $k$-ary $n$-cube networks that adapts globally by sensing global congestion using queues at the source node. By sensing global congestion, GAL adapts to route minimally when it can, and non-minimally only when the minimal channels become congested. Previous adaptive algorithms that sense channel congestion, rather than global congestion, are not able to adapt in this manner. GAL also adapts locally, sensing channel congestion using channel queues.

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 seamlessly to non-minimal routing as congestion is detected by the injection queues. At saturation, GAL matches the throughput of optimally load-balanced oblivious routing. This combines the best features of minimal algorithms (low latency at low load) and obliviously load balanced algorithms (high saturation throughput).

The report card of Table IV summarizes the performance of GAL and compares it to previous algorithms. This report card augments the qualitative grade from Table I with a quantitative measure of performance. To simplify comparison we have normalized all latency and throughput measures to VAL. GAL is the only algorithm to get straight "A"s having the highest performance in every category. It matches the throughput of minimal algorithms on benign traffic patterns (such as NN) and the throughput of obliviously load-balanced algorithms on hard patterns (such as TOR). It has the highest average throughput across a sample of 1,000 random permutations, and the highest throughput on hot-spot traffic.

| | VAL | CHAOS | MIN AD | GOAL | **GAL** |
|---|---|---|---|---|---|
| $\Theta_{\mathbf{NN}}$ | 1.0 (F) | 8.0 (A) | 8.0 (A) | 4.7 (C) | **8.0 (A)** |
| $\Theta_{\mathbf{TOR}}$ | 1.0 (B) | 0.72 (C) | 0.66(C) | 1.06 (A) | **1.06 (A)** |
| $\Theta_{\mathbf{avg}}$ | 1.0 (D) | 1.1 (C) | 1.3 (B) | 1.35(A) | **1.44 (A)** |
| $\Theta_{\mathbf{HS}}$ | 1.0 (F) | 1.88(A) | 1.84(A) | 1.84(A) | **1.96 (A)** |
| **Lat** | 1.0 (F) | 0.45(A) | 0.45(A) | 0.63(C) | **0.45 (A)** |
| **Stab** | Yes (P) | No (F) | Yes (P) | Yes (P) | **Yes (P)** |

TABLE IV
RESULT SUMMARY OF FIVE ROUTING ALGORITHMS ON AN 8-ARY 2-CUBE

To make GAL stable on minimal traffic patterns requires an adaptive queue threshold. At saturation, all queues, including the minimal queue, exceed their thresholds and the algorithm begins to route non-minimally. The router senses the drop in throughput due to this non-minimal routing and increases the threshold until routing returns to minimal. When traffic changes to a pattern that requires non-minimal routing variation in throughput adjusts the threshold back down. With an adaptive threshold, GAL receives a passing grade "P" for stability in Table IV.

While GAL outperforms all known routing algorithms on every test to which we have subjected it, it is not an optimal algorithm. As shown in Figure 10 the *ideal* throughput for a set of 1,000 random permutations, as determined by solving a max-flow problem for each permutation, significantly outperforms GAL. Closing this gap further between GAL and ideal is an open research issue.

GAL's use of injection queues to sense global congestion has applications beyond selecting the quadrant in which to route each packet. This measure of global congestion could, for example, be used to drive a data or thread migration policy. Also, in cases when a packet can be delivered to any one of a set of destinations (e.g., the packet can be sent to any server that provides a particular service), the measure of global congestion can be used to decide which server to select.

Adaptive routing algorithms, whether global (like GAL) or local (like MIN AD), because they adapt to changes in network traffic, are characterized by both a steady-state and a transient response. In contrast, oblivious algorithms (like GOAL) are entirely characterized by their steady-state 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.

## REFERENCES


[1] W. J. Dally, "Performance analysis of k-ary n-cube interconnection networks," *IEEE Transactions on Computers*, vol. 39, no. 6, pp. 775–785, 1990.

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

[3] 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.

[4] G. Pfister, *An Introduction to the InfiniBand Arechitecture (http://www.infinibadta.org)*. IEEE Press, 2001.

[5] L. G. Valiant, "A scheme for fast parallel communication," *SIAM Journal on Computing*, vol. 11, no. 2, pp. 350–361, 1982.

[6] 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.

[7] 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.

[8] D. Linder and J. Harden, "An adaptive and fault tolerant wormhole routing strategy for k-ary n-cubes." *IEEE Transaction on Computers*, vol. 40, no. 1, pp. 2–12, Jan. 1991.

[9] A. Singh, W. J. Dally, B. Towles, and A. K. Gupta, "Locality-preserving randomized routing on torus networks," in *Proc. 12th Annual ACM Symposium on Parallel Algorithms and Architectures SPAA'02*, Winnipeg, Canada, 2002.

[10] 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.

[11] J. Duato, S. Yalamanchili, and L. Ni, *Interconnection Networks An Engineering Approach*. IEEE Press, 1997.

[12] P. Gaughan and S. Yalamanchili, "Adaptive routing protocols for hypercube interconnection networks," *IEEE Computer*, vol. 26, no. 5, pp. 12–23, May 1993.

[13] S. Mukherjee, P. Bannon, S. Lang, A. Spink, and D. Webb, "The alpha 21364 network architecture," in *Proc. Hot Interconnects 9*, Stanford, California, August 2001, pp. 113–117.

[14] A. A. Chien and J. H. Kim, "Planar-adaptive routing: Low-cost adaptive networks for multiprocessors," *Journal of the ACM*, vol. 42, no. 1, pp. 91–123, January 1995.

[15] 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.

[16] W. J. Dally and H. Aoki, "Adaptive routing using virtual channels," *IEEE Trans. on Parallel and Distributed Systems*, vol. 4, no. 4, pp. 466–475, April 1993.

[17] 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, pp. 194–205.

[18] L.-S. Peh and W. J. Dally, "A delay model for router micro-architectures,," *IEEE Micro*, vol. 21, no. 1, pp. 26–34, 2001.

[19] B. Towles and W. J. Dally, "Worst-case traffic for oblivious routing functions." in *Proc. 12th Annual ACM Symposium on Parallel Algorithms and Architectures SPAA'02*, Winnipeg, Canada, 2002.

[20] F. Shahrokhi and D. W. Matula, "The maximum concurrent flow problem," *Journal of the ACM (JACM)*, vol. 37, no. 2, pp. 318–334, April 1990.

[21] A. S. Tanenbaum, *Computer Networks, 3rd ed.* Prentice Hall, 1996, pages 202-219.