GOAL: A Load-Balanced Adaptive Routing Algorithm for Torus Networks *

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

Abstract

We introduce a load-balanced adaptive routing algorithm for torus networks, GOAL - Globally Oblivious Adaptive Locally - that provides high throughput on adversarial traffic patterns, matching or exceeding fully randomized routing and exceeding the worst-case performance of Chaos [2], RLB [14], and minimal routing [8] by more than 40%. GOAL also preserves locality to provide up to $4.6 \times$ the throughput of fully randomized routing [19] on local traffic. GOAL achieves global load balance by randomly choosing the direction to route in each dimension. Local load balance is then achieved by routing in the selected directions adaptively. We compare the throughput, latency, stability and hot-spot performance of GOAL to six previously published routing algorithms on six specific traffic patterns and 1,000 randomly generated permutations.

1 Introduction

Interconnection networks based on torus or k-ary n-cube topologies [3] are widely used as switch and router fabrics [5], for processor-memory interconnect [13], and for I/O interconnect [12]. Torus networks have high path diversity, offering many alternative paths between a message source and destination. A good routing algorithm chooses between these paths in a manner that exploits locality to provide low latency, balances load to provide high throughput on adversarial traffic patterns, and gracefully handles momentary overloads to provide stability.

Many applications require the interconnection network to provide 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 any traffic pattern, even the worst-case, at the line rate or packets will be dropped. To meet their specifications, 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. This arises when operating on an irregular graph structure or on a regular structure that is randomly mapped to the nodes of the machine. Performance on these applications is limited by the throughput of the network on adversarial patterns.

A routing algorithm must strike a balance between the conflicting goals of providing low latency on local traffic and providing high throughput on adversarial traffic. To achieve high performance on local traffic, minimal routing algorithms - that choose a shortest path for each packet - are favored. Minimal algorithms, however, perform poorly on worst-case traffic due to load imbalance. With a minimal routing algorithm, an adversarial traffic pattern can load some links very heavily while leaving others idle. To improve performance under worst-case traffic, a routing algorithm must balance load by sending some fraction of traffic over non-minimal paths - hence destroying some of the locality.

Previous work has attempted to address the issue of providing high worst-case performance while preserving locality. Valiant's randomized algorithm [19] gives good performance on worst-case traffic, but at the expense of completely destroying locality and hence giving very poor performance on local traffic and greatly increasing latency. The Chaos routing algorithm [2] 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 [8], [9] also suffers from this global load imbalance. The RLB algorithm [14] globally load balances...
traffic, but performs poorly on worst-case patterns because it cannot locally adapt to traffic.

In this paper, we introduce Globally Oblivious Adaptive Locally (GOAL) - a non-minimal, adaptive routing algorithm for torus networks that strikes a balance between the conflicting goals of locality and load balance. GOAL combines the global load balance of oblivious routing with the local balance of adaptive methods. GOAL obliviously chooses the direction of travel in each dimension weighting the short direction more heavily than the long direction in a manner that globally balances channel load while preserving some locality. Once the directions are selected, the packet is adaptively routed to the destination in the resulting quadrant. This adaptive routing avoids the local imbalance created by adversarial patterns.

The GOAL algorithm offers high throughput on worst-case patterns — matching or exceeding the performance of Valiant’s algorithm, and exceeding the worst-case performance of Chaos, RLB, and minimal routing by more than 40%, on known adversarial traffic patterns. At the same time GOAL exploits locality giving 4.6x the throughput of Valiant on local traffic and more than 30% lower zero-load latency than Valiant on uniform traffic. Because it is locally adaptive, GOAL matches the performance of Chaos and minimal adaptive routing on hot-spot traffic. Because it occasionally routes the long way around, GOAL does not match the performance of minimal algorithms on local traffic - achieving only 58% the throughput of minimal algorithms on nearest neighbor traffic. However, we show that any minimal algorithm asymptotically has at best half the worst-case throughput as Valiant’s algorithm.

The remainder of this paper describes the GOAL algorithm in more detail and compares its performance to other routing algorithms. Section 3 derives theoretical bounds on the performance of minimal and non-minimal algorithms. Section 4 describes the GOAL algorithm in detail. We measure the performance of GOAL in Section 5 and compare its performance to existing routing algorithms.

2 Preliminaries

2.1 Definitions and Conventions

We restrict our discussion to multi-dimension torus or k-ary n-cube networks. A k-ary n-cube is a n dimensional torus network with n nodes per dimension giving a total of \( N = k^n \) nodes. Each link is unidirectional, hence there are two links between adjacent nodes — one for each direction. A packet may be divided into fixed size units called flits. Each link has unit capacity, i.e. it can forward one flit per cycle.

In all the algorithms evaluated in this paper (except the Chaos routing algorithm [2]), credit-based virtual channel flow control [4] is employed. To make the algorithms stable, age-based arbitration is used with the oldest packet winning the contested resource. All addition and subtraction on node coordinates is performed modulo \( k \) yielding a result that is in the range \([0, k-1]\).

Traffic pattern \((\alpha \Lambda)\) - \(\Lambda\) is a \(N \times N\) doubly sub-stochastic destination matrix where each entry \(0 \leq \lambda_{i,j} \leq 1\) is equal to the probability that node \(i\) chooses node \(j\) as its destination. All nodes inject the same average number of packets, \(\alpha\), per time step into the network. A node is oversubscribed if the sum of its row or column entries in \(\Lambda\) is greater than 1. A traffic pattern is admissible if none of the nodes in its destination matrix, \(\Lambda\) is oversubscribed.

Permutation matrix \((\Pi)\) - A destination matrix \(\Lambda\) with a single 1 entry in each row and column. All other entries are 0.

Routing algorithm \((R)\) - A routing algorithm maps a source-destination pair to a path through the network from the source to the destination. Oblivious algorithms select the path using only the identity of the source and destination nodes. Adaptive algorithms may also base routing decisions on the state of the network. Both oblivious and adaptive algorithms may use randomization to select among alternative paths.

Minimal algorithms only route packets along a shortest path from source to destination while non-minimal ones may route packets along longer paths.

Channel load \((\gamma_c(R, \Lambda))\) - The expected number of packets that cross channel \(c\) per cycle for the destination matrix \(\Lambda\) and routing algorithm \(R\).

Normalized worst-case channel load \((\gamma_{wcn}(R, \Lambda))\) - The expected number of packets crossing the heaviest loaded channel normalized to the network capacity \(2\),

\[
\gamma_{wcn}(R, \Lambda) = \frac{\max_{C} \gamma_c(R, \Lambda)}{k/8},
\]

where \(C\) is the set of all channels in the network.

Saturation throughput \((\Theta(R, \Lambda))\) - The average number of packets that can be injected by all the nodes in the network per time step so as to saturate the worst-case link to its unit capacity.

\[
\Theta(R, \Lambda) = \frac{1}{\gamma_{wcn}(R, \Lambda)}
\]

\(^2\)The network capacity is the maximum load that the bisection of the network can sustain for uniformly distributed traffic. In a k-ary n-cube subjected to uniform random traffic, there are \(\frac{k^n}{4}\) packets trying to cross the \(4k^{n-1}\) (unidirectionally) links in the bisection. Hence, each link in the bisection must support an average load of \(\frac{k^n/2}{k^{n-1}} = \frac{k}{4}\) packets per cycle.
Worst-case saturation throughput \((\Theta_{wc}(R))\) - The worst-case saturation throughput for routing algorithm \(R\) over the space of all admissible destination matrices,

\[
\Theta_{wc}(R) = \min_{\Lambda} \Theta(R, \Lambda)
\]

### 2.2 Performance Measures

For a given traffic pattern \(\Lambda\) and routing algorithm \(R\) we can describe the performance of an interconnection network with two graphs as shown in Figure 1. The steady-state performance of the network at offered loads below the saturation point is described by Figure 1(a) which shows the average latency per packet plotted against the offered load \(\alpha\). The zero-load latency or hop-count \(H\) is the \(y\)-intercept of this curve and the saturation throughput \(\Theta\) is the \(x\)-coordinate of the asymptote. At offered loads greater than \(\Theta\), average steady-state latency is infinite.

![Figure 1. (a) Notational latency vs. offered load graph. (b) Accepted throughput vs. offered load.](image)

Figure 1(b), which shows a plot of accepted traffic \(\alpha'\) as a function of offered load \(\alpha\), describes network performance after the saturation point, when \(\alpha > \Theta\). We report the minimum accepted traffic over all source-destination pairs to reflect the throughput achieved for the specified traffic matrix \(\Lambda\). Under heavy load, the source-destination pairs with less contention deliver more packets than other pairs. In effect, these pairs get ahead of the other pairs. However, the amount of the desired destination matrix \(\Lambda\) that is delivered is governed by the slowest pair (the one with the least accepted throughput).

At offered loads less than the saturation point all traffic is delivered so \(\alpha'(\alpha) = \alpha\). Beyond the saturation point accepted traffic is flat for a stable routing algorithm. That is, \(\alpha'(\alpha) = \Theta\) for \(\alpha \geq \Theta\). For unstable algorithms, throughput degrades beyond saturation. This occurs for some non-minimal algorithms where, due to congestion, average path length increases with offered traffic. Instability may also occur when global fairness is not maintained and hence the throughput of the slowest source-destination pair is reduced after saturation because more of a critical shared resource is being granted to a faster source-destination pair.

![Figure 2. Minimally routed pattern](image)

To illustrate why we compute \(\alpha'\) as the minimum across all source-destination pairs, consider an 8-ary 1-cube network routing three flows as shown in Figure 2. There are three source nodes - 0, 1 and 5 - sending packets to destination nodes 3, 2 and 6 respectively. All source nodes inject the same load, \(\alpha\) into the network. As we increase the injected load for each node from zero up to the saturation point, \(\alpha = 0.5\), none of the links in the network are saturated and the average accepted load \(\alpha'\) and the minimum accepted load \(\alpha''\) across the flows are both the same, i.e., \(\alpha'' = \alpha' = \alpha\). Suppose we now increase \(\alpha\) to 1.0. Link 1 \(\rightarrow\) 2 becomes saturated and allocates half of its capacity to each of the two flows 0 \(\rightarrow\) 3 and 1 \(\rightarrow\) 2. However, link 5 \(\rightarrow\) 6 offers its full capacity to flow 5 \(\rightarrow\) 6. The accepted loads for nodes 2, 3 and 6 are therefore 0.5, 0.5 and 1.0 respectively. Hence, \(\alpha'' = 2/3 = 0.67\) while the \(\alpha' = 0.5\). The minimum number \(\alpha'\) reflects the amount of the original destination matrix \(\Lambda\) that is being delivered, the extra traffic on 5 \(\rightarrow\) 6 represents additional traffic beyond the \(\alpha'\Lambda\) that is not part of the specified destination matrix.

### 3 Performance Bounds

In this section, we make some claims regarding the performance of different types of routing algorithms.

**Claim 1.** No routing algorithm can guarantee a throughput greater than half the network capacity on \(k\)-ary \(n\)-cubes.

**Proof.** We prove this claim by showing that for a particular traffic permutation, no routing algorithm, \(R_s\) can achieve a saturation throughput greater than 0.5. Consider the permutation \(\Pi_{dia}\), in which every source sends a packet half way across the ring in each dimension. In a \(k\)-ary 2-cube, every source \((i, j)\) sends to \((i + k/2, j + k/2)\). In such a permutation, there are a total of \(k^n\) packets, each packet being \(nk/2\) hops away from its destination. This implies that all the packets need to traverse at least \((k^n)nk/2\) links in the network. However, there are \((2n)k^n\) links in a \(k\)-ary \(n\)-cube network. Therefore, no matter what routing algorithm, \(R_s\) is employed,
Claim 1 therefore, gives us an upper bound on the worst-case performance of any routing algorithm.

Claim 2. Valiant’s algorithm gives optimal worst-case throughput but performs identically on every admissible destination matrix on k-ary n-cubes.

Proof. Let us first consider any permutation matrix, \( \Pi_s \). Valiant’s algorithm, \( R_{val} \), involves two completely random phases. A source node first sends a packet to a completely random intermediate node \( q \) and then from \( q \) to the actual destination. On average, a packet traverses \( k/4 \) hops per dimension in each of the two phases. So, in all, the \( k^n \) packets of \( \Pi_s \) traverse \( 2n(k/4) \) links simultaneously. There are \( (2n)k^n \) links in the network that uniformly support this traffic. Since, all the channels are identically loaded for \( R_{val} \),

\[
\forall c, \quad \gamma_c(R_{val}, \Pi_s) = \frac{k^n2n(k/4)}{2nk^n} = \frac{k}{4} \quad (1)
\]

By the result of Birkhoff [1], any admissible destination matrix, \( \Lambda_s \), can be written as a weighted combination of permutation matrices:

\[
\Lambda_s = \sum_i \phi_i \Pi_i, \quad \text{s.t. } \sum_i \phi_i = 1 \quad (2)
\]

Hence,

\[
\forall c, \quad \gamma_c(R_{val}, \Lambda_s) = \gamma_c(R_{val}, \sum_i \phi_i \Pi_i) \quad (3)
\]

As shown in [17], due to the linearity of channel loading in oblivious routing algorithms, such as \( R_{val} \), we can rewrite (3) as:

\[
\forall c, \quad \gamma_c(R_{val}, \Lambda_s) = \sum_i \gamma_c(R_{val}, \phi_i \Pi_i) = \sum_i \phi_i \gamma(R_{val}, \Pi_i) = \sum_i \phi_i \frac{k}{4} = \frac{k}{4} \quad \text{from } (1), (2)
\]

Since all channels are identically loaded, we get \( \gamma_{wc}(R_{val}, \Lambda_s) = \frac{k/4}{k/4} = 2 \). Therefore,

\[
\Theta_{wc}(R_{val}) = \Theta(R_{val}, \Lambda_s) = 0.5
\]

Claim 2 shows that Valiant’s algorithm gives optimal worst-case saturation throughput. However, it destroys the locality of every traffic pattern reducing it to the worst case. Valiant also doubles the average path length of any packet thereby resulting in very high latency even at low offered load.

Next, let us consider the class of minimal routing algorithms.

Claim 3. No minimal routing algorithm can guarantee a throughput asymptotically greater than 25% the network capacity on k-ary n-cubes.

Proof. Consider the tornado traffic permutation, \( \Pi_{tor} \), in which every source sends a packet to a destination that is one less than half-way across the ring in a single dimension. For e.g., in an 8-ary 2-cube, every source \( (i, j) \) sends to \( (i + k/2 - 1, j) \). A minimal routing algorithm, \( R_{min} \), will only employ links in one direction in the ring, leaving the links going the other direction idle. Links in only one dimension will be used. All other links will be idle. Figure 3 shows how minimally routing tornado traffic on an 8-ary 1-cube causes the packets to rotate around the ring in a single direction like a tornado, keeping the links in the other direction completely idle. Hence,

\[
\gamma_{wc}(R_{min}, \Pi_{tor}) = \frac{k/2-1}{k/8} \approx 0.25
\]

\[
\Theta_{wc}(R_{min}) \leq \Theta(R_{min}, \Pi_{tor}) = \frac{k/8}{k/2-1} \approx 0.25
\]

\[
(\text{for } k/2 >> 1)
\]

**Figure 3.** Minimally routed tornado traffic on an 8-ary 1-cube. Clockwise link load is 3. Counter clockwise link load is 0.

Therefore, from a worst-case throughput point of view, a minimal algorithm performs sub-optimally.

4 GOAL

The GOAL algorithm routes a packet from a source node \( s = \{s_1, \ldots, s_n\} \) to the destination node \( d = \{d_1, \ldots, d_n\} \) by obliviously choosing the direction to travel in each of the \( n \) dimensions to exactly balance channel load (as is done by
the RLB algorithm [14]). Choosing the directions selects the quadrant in which a packet will be routed in a manner that balances load among the quadrants. Then, traveling only in the selected directions, the packet is routed adaptively from s to d. At each hop, the router advances the packet in the productive dimension that has the shortest queue.

In each dimension, the choice of direction is made to exactly balance the load on the channels in the two directions. In each dimension $i$, the distance in the shorter of the two directions is $\Delta_i = \min(d_i - s_i, s_i - d_i)$ . The direction of the short path is $r_i = 1$ if $\Delta_i = d_i - s_i$ and $r_i = -1$ otherwise. The selected direction is based on a probability distribution favoring the short path. To exactly balance the load due to symmetric traffic we send each packet in the short direction, $r_i$, with probability $P_{r_i} = \frac{k-\Delta_i}{k}$ and in the long direction, $-r_i$, with probability $P_{-r_i} = \frac{\Delta_i}{k}$.

For example, suppose we are routing from $s = (0, 0)$ to $d = (2, 3)$ in an 8-ary 2-cube network ($8 \times 8$ 2-D torus). The distance vector is $\Delta = (2, 3)$, the minimal direction vector is $r = (1, 1)$, and the probability vector is $P = (0.75, 0.625)$. We have four choices for choosing directions, $(+1, +1)$, $(+1, -1)$, $(-1, +1)$, and $(-1, -1)$ which we choose with probabilities 0.469, 0.281, 0.156, and 0.094 respectively. Each of these four directions describes a quadrant of the 2-D torus as shown in Figure 4. The weighting of directions routes more traffic in the minimal quadrant $(+1, +1)$ (shaded darkest) and less in the quadrant that takes the long path in both dimensions $(-1, -1)$ (shaded lightest).

In each dimension, the choice of direction is made to exactly balance the load on the channels in the two directions. In each dimension $i$, the distance in the shorter of the two directions is $\Delta_i = \min(d_i - s_i, s_i - d_i)$ . The direction of the short path is $r_i = 1$ if $\Delta_i = d_i - s_i$ and $r_i = -1$ otherwise. The selected direction is based on a probability distribution favoring the short path. To exactly balance the load due to symmetric traffic we send each packet in the short direction, $r_i$, with probability $P_{r_i} = \frac{k-\Delta_i}{k}$ and in the long direction, $-r_i$, with probability $P_{-r_i} = \frac{\Delta_i}{k}$.

For example, suppose we are routing from $s = (0, 0)$ to $d = (2, 3)$ in an 8-ary 2-cube network ($8 \times 8$ 2-D torus). The distance vector is $\Delta = (2, 3)$, the minimal direction vector is $r = (1, 1)$, and the probability vector is $P = (0.75, 0.625)$. We have four choices for choosing directions, $(+1, +1)$, $(+1, -1)$, $(-1, +1)$, and $(-1, -1)$ which we choose with probabilities 0.469, 0.281, 0.156, and 0.094 respectively. Each of these four directions describes a quadrant of the 2-D torus as shown in Figure 4. The weighting of directions routes more traffic in the minimal quadrant $(+1, +1)$ (shaded darkest) and less in the quadrant that takes the long path in both dimensions $(-1, -1)$ (shaded lightest).

Once we have selected the quadrant, the packet is routed adaptively within that quadrant. A dimension $i$ is productive if, the coordinate of the current node $x_i$ differs from $d_i$. Hence, it is productive to move in that dimension since the packet is not already at the destination coordinate. At each hop, the productive dimension with the shortest output queue is selected 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 routing algorithm has obliviously chosen the minimal quadrant, $(+1, +1)$. One possible route of the packet is shown in Figure 5. On the first hop, the productive dimension vector is $p = (1, 1)$ that is both the $x$ and $y$ dimensions are productive. Suppose the 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.

4.1 Virtual Channels and Deadlock

Our implementation of GOAL employs 3 virtual channels (VCs) per unidirectional physical channel (PC) to achieve deadlock freedom in the network. This is an extension of the scheme proposed in the *-channels algorithm [8] applied to the non-minimal GOAL algorithm. There are two types of virtual channels per PC, * and non-*. Packets will move through the *-channels only when traversing the most significant productive dimension. The non-* channels are fully adaptive and can be used at any time. In order to make the *-channel subnetwork free from deadlock, we have two *-channels per PC - *0 and *1. *1 (⋆0) is used if the packet has (has not) crossed a wrap-around edge in the current dimension. With these constraints it can be seen
that the channel dependency graph\(^3\) for the \(\ast\)-channels associated with GOAL is acyclic. Moreover, no \(\ast\)-channel can ever participate in a deadlock cycle. Hence, every packet that has not reached its destination always has a \(\ast\)-channel in the set of virtual channels it can possibly use to make forward progress towards its destination. Therefore, if VCs are assigned fairly, deadlock can never arise. The formal proof is a simple extension of that provided in Appendix A of [8] and is not presented in this paper.

### 4.2 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. All minimal algorithms like DOR, ROMM and MIN AD\(^4\) 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. Both the RLB and GOAL algorithms while non-minimal, provide deterministic freedom from livelock. Once a route has been selected for a packet, the packet monotonically makes progress along the route, 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 a randomized or 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 [16].

### 5 Performance evaluation

In this section we compare the throughput and latency of the seven routing algorithms described in Table 1 on the local and adversarial traffic patterns described in Table 2 and on 1,000 random permutations. We also compare the latency histograms for these algorithms on random traffic, their performance on hot-spot traffic and their stability post saturation.

The first two traffic patterns in Table 2, NN and UR, are benign in the sense that they naturally balance load and hence give good throughput with simple 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 [14], [17], [2], [7], [8], [10]. 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. As shown in [17], the worst-case pattern for oblivious algorithms such as DOR, ROMM, VAL and RLB can be restricted to traffic permutations and may be analytically computed. These patterns are reported in [14]. However, there is no known analytical method to determine the worst case traffic pattern for adaptive routing algorithms.

#### 5.1 Experimental Setup

Measurements in this section have been made on a cycle-accurate network simulator that models the pipeline of each

---

\(^3\)For a detailed explanation of channel dependency graphs, see [6].

\(^4\)See Table 1 for a description of these algorithms.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAL</td>
<td>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 [19].</td>
</tr>
<tr>
<td>DOR</td>
<td>Dimension-order routing - route in the minimal quadrant in (x) first, then in (y). Deadlock is avoided using 2 VCs [15].</td>
</tr>
<tr>
<td>ROMM</td>
<td>Two-phase ROMM - route to random node (q) in the minimal quadrant, then to destination [10]. Deadlock is avoided using the same scheme as in VAL.</td>
</tr>
<tr>
<td>RLB</td>
<td>Randomized Local Balance [14] - choose a quadrant (Q) to route in according to a weighted probability distribution, then route within (Q) first to a random intermediate node then to the destination randomizing the order of matching dimensions. Deadlock is avoided using Linder and Harden’s scheme [9].</td>
</tr>
<tr>
<td>CHAOS</td>
<td>The Chaos routing algorithm. Deadlock is avoided using deflection routing [2].</td>
</tr>
<tr>
<td>MIN AD</td>
<td>Minimal Adaptive (or the (\ast)-channels algorithm) - always route in the minimal quadrant, routing adaptively within it. Deadlock is avoided using 3 VCs [8].</td>
</tr>
<tr>
<td>GOAL</td>
<td>Globally Oblivious Adaptive Locally - choose a quadrant (Q) to route in according to a weighted probability distribution, then route within (Q) adaptively. Deadlock is avoided using 3 VCs [8].</td>
</tr>
</tbody>
</table>

Table 1. The routing algorithms evaluated in this paper.
Table 2. Traffic patterns for evaluation of routing algorithms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NN</td>
<td>Nearest Neighbor - each node sends to one of its four neighbors with probability 0.25 each.</td>
</tr>
<tr>
<td>UR</td>
<td>Uniform Random - each node sends to a randomly selected node.</td>
</tr>
<tr>
<td>BC</td>
<td>Bit Complement - ((x, y)) sends to ((k - x - 1, k - y - 1)).</td>
</tr>
<tr>
<td>TP</td>
<td>Transpose - ((x, y)) sends to ((y, x)).</td>
</tr>
<tr>
<td>TOR</td>
<td>Tornado - ((x, y)) sends to ((x + \frac{y}{k} - 1, y)).</td>
</tr>
<tr>
<td>WC</td>
<td>Worst-case - the traffic pattern that gives the lowest throughput by achieving the maximum load on a single link [17].</td>
</tr>
</tbody>
</table>

Figure 6 shows the saturation throughput for each algorithm. Here the balanced algorithms RLB and GOAL offer the best performance because they efficiently balance load across the two directions in the \(x\) dimension. VAL does nearly as well but has more than twice the zero load latency as the balanced algorithms because of its two-phase

5.2 Throughput on Specific Permutations

Figure 6 shows the saturation throughput for each algorithm on each traffic pattern. The two benign traffic patterns are shown in Figure 6(a) while the four adversarial

patterns are shown in Figure 6(b) with an expanded vertical scale. The figure shows that GOAL achieves performance at least as good as VAL on the adversarial patterns while offering significantly more performance on the benign patterns - 52% higher throughput on random traffic and 4.6× the throughput on nearest-neighbor. However, the throughput of GOAL does not match the performance of minimal algorithms on the local patterns. This is the price of good worst-case performance.

Figure 6 also shows that the minimal algorithms, DOR, ROMM and MIN AD, offer high performance on benign traffic patterns but have very poor worst-case performance. Because the adaptivity of CHAOS is local in nature, its performance is comparable to that of MIN AD, a minimal algorithm. VAL gives provably optimal worst-case performance but converts every traffic pattern to this worst case giving very poor performance on the benign patterns and sub-optimal performance on several of the adversarial patterns.

The exact worst-case throughput for the oblivious algorithms — DOR, ROMM, VAL and RLB — is shown in Figure 6(b). Since there is no known method to evaluate the worst case pattern for adaptive algorithms, the worst case graphs shown for CHAOS, MIN AD and GOAL show the lowest throughput over all traffic patterns. We know from claim (3) that MIN AD saturates at 0.33 for tornado traffic for \(k = 8\) and further deteriorates to 0.285 for \(k = 16\). Chaos does not do appreciably better on tornado traffic saturating at 0.35 and 0.30 for \(k = 8\) and \(k = 16\) respectively. The worst case pattern for GOAL that we know of at this point is the \(I_{dia}\) permutation discussed in claim (1) on which it saturates at 0.50.

The latency-load curves for the benign UR pattern for all the algorithms are shown in Figure 7. On this benign pattern the minimal algorithms give the best performance, VAL gives the worst performance, and GOAL falls midway between the two. CHAOS, MIN AD, and DOR all offer minimal zero-load latency and unit throughput because they always take the shortest path. Because VAL performs two rounds of randomized routing, its zero load latency is twice that of the minimal algorithms and its saturation throughput is half of theirs. Because GOAL occasionally routes the long way around its latency is increased and its throughput is reduced compared to the minimal algorithms. However, it offers substantially better performance than VAL while giving the same performance as VAL on worst-case traffic.

Figure 8 shows the latency-load curves for each algorithm on the adversarial TOR pattern for each of the algorithms. Here the balanced algorithms RLB and GOAL offer the best performance because they efficiently balance load across the two directions in the \(x\) dimension. VAL does nearly as well but has more than twice the zero load latency as the balanced algorithms because of its two-phase

router as described in [11]. Routing is assumed to take one 20 FO4 pipeline stage for each algorithm as supported by gate-level designs. All the assumptions regarding the network model stated in Section 2.1 hold for the experimental setup. Each packet is assumed to be one flit long and each virtual channel is 8 flits deep. The total buffer resources are held constant across all algorithms, i.e. 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 the age-based arbitration, always giving priority to a packet with an older timestamp since injection. All latency numbers presented are measured since the time of birth of the packets and in-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, 8-ary 2-cube and 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 trend.

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, all injected packets were labeled for measurement for 100/load cycles. The simulation was then run until all labeled packets reached their destinations. The packet sample size was chosen to ensure that measurements are accurate to within 3% with 99% confidence.
nature and because it takes gratuitous hops in the $y$ dimension. The minimal algorithms, DOR, MIN AD, and ROMM all perform poorly (37% lower throughput than GOAL) on the tornado pattern because they route all of the traffic in the shorter direction, leaving the channels in the other direction idle. While CHAOS is not a minimal algorithm, its adaptivity is local in nature and thus is not able to globally balance load across directions. Thus the performance of CHAOS closely matches the performance of MIN AD on all adversarial patterns, including TOR.

5.3 Throughput on Random Permutations

One might ask how often permutations as bad as the adversarial patterns of Figure 6 occur in practice. To address this question, we measured the performance of each algorithm on 1,000 random permutations\(^5\). Histograms of saturation throughput over these permutations for six of the algorithms are shown in Figure 9. No histogram is shown for VAL because its throughput is always 0.5 for all permutations. All the other routing algorithms have bell-shaped histograms. The highest, average and worst throughput in this experiment for each of the algorithms are presented in Figure 10.

The figures show that over the 1,000 permutations, GOAL is the only algorithm with a worst-case throughput that matches or exceeds that of VAL. The minimal algorithms and CHAOS do substantially worse. GOAL outperforms the best of these algorithms, MIN AD, by 31%. The figures also show that despite the fact that it obliviously routes a fraction of traffic the long way around, GOAL has the highest average case throughput of all of the algorithms, 5These $10^3$ permutations are selected from the $N! = k^n!$ possible permutations on an $N$-node $k$-ary $n$-cube.
outperforming MIN AD by 5%. This shows clearly that even for an average permutation, global load balance enhances performance and it is worth obliviously misrouting (as in GOAL) to achieve this balance.

The figure also shows the importance of using adaptive routing to achieve local balance. GOAL has 49% higher average throughput and 75% higher worst-case throughput than RLB which also uses quadrant selection to achieve global balance but attempts to obliviously achieve local balance. For the same reason, the unbalanced adaptive algorithms MIN AD and CHAOS outperform the minimal oblivious algorithms, DOR and ROMM. MIN AD slightly outperforms CHAOS in terms of both average and worst-case throughput. This suggests that for most permutations, local misrouting is not advantageous.

6This advantage of adaptive routing has also been noted in [2].

5.4 Latency

To compare the latency of the seven algorithms, we computed latency histograms between a representative source-destination pair in a network loaded with uniform random traffic for each algorithm. In an 8-ary 2-cube loaded with uniform traffic, the Manhattan distance between a source and a destination node can range from 1 to 8. In our experiments, we chose to measure the latency incurred by packets from a source to 3 different destination nodes with a background of uniform random traffic at 0.2 offered 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.

The histograms for semi-local paths (packets from A to C) are presented 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 seven 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 queueing 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, $H \geq D$.

7The other sets of histograms are not presented due to space constraints.
The results for all the three representative paths are presented in Figure 12. Under benign traffic at low load, the three minimal algorithms, DOR, ROMM, and MIN AD, and CHAOS (which behaves minimally at low load) give the lowest latency. All of these algorithms have a minimal hop count, $H = 4$, and the queueing delay $Q$ is exponentially distributed with means ranging from 0.44 cycles for MIN AD to 0.76 cycles for ROMM. This difference in queueing delay further shows the advantage of adaptivity. The result is a latency histogram with a peak at $H = 4$ that falls off exponentially.

The balanced algorithms, RLB and GOAL, have higher latency (40% higher than minimal) and broader distributions than the minimal algorithms because they route a fraction of the traffic in the non minimal quadrants. Depending on the quadrant chosen the hop count for a given packet is $H = 4, 6, 10, \text{ or } 12$. 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. However the balanced algorithms offer much lower latency than VAL. GOAL is on average 2.45 times, 1.60 times and 1.12 times faster than VAL on local, semi-local and non-local paths respectively.

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.

The balanced algorithms, RLB and GOAL, have higher latency (40% higher than minimal) and broader distributions than the minimal algorithms because they route a fraction of the traffic in the non minimal quadrants. Depending on the quadrant chosen the hop count for a given packet is $H = 4, 6, 10, \text{ or } 12$. 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. However the balanced algorithms offer much lower latency than VAL. GOAL is on average 2.45 times, 1.60 times and 1.12 times faster than VAL on local, semi-local and non-local paths respectively.

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.

The results for all the three representative paths are presented in Figure 12. Under benign traffic at low load, the three minimal algorithms, DOR, ROMM, and MIN AD, and CHAOS (which behaves minimally at low load) give the lowest latency. All of these algorithms have a minimal hop count, $H = 4$, and the queueing delay $Q$ is exponentially distributed with means ranging from 0.44 cycles for MIN AD to 0.76 cycles for ROMM. This difference in queueing delay further shows the advantage of adaptivity. The result is a latency histogram with a peak at $H = 4$ that falls off exponentially.

The balanced algorithms, RLB and GOAL, have higher latency (40% higher than minimal) and broader distributions than the minimal algorithms because they route a fraction of the traffic in the non minimal quadrants. Depending on the quadrant chosen the hop count for a given packet is $H = 4, 6, 10, \text{ or } 12$. 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. However the balanced algorithms offer much lower latency than VAL. GOAL is on average 2.45 times, 1.60 times and 1.12 times faster than VAL on local, semi-local and non-local paths respectively.

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.
5.6 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 a hot-spot pattern similar to that used in [2]. We first select a background traffic pattern, bit complement ($\Lambda_{BC}$), on which most of the algorithms give similar performance. On top of $\Lambda_{BC}$, five nodes$^8$ are selected which are five times more likely to be chosen as destinations than the other nodes. In the resulting matrix $\Lambda_{HS}$ all rows sum to one, but the five columns corresponding to the five hot-spot nodes sum to five. Since the three adaptive algorithms — CHAOS, MIN AD and GOAL — and two oblivious algorithms — VAL and DOR — give similar performance on $\Lambda_{BC}$, we present results for these five on the resulting $\Lambda_{HS}$ traffic.

It is worth noting that some algorithms presented in the literature such as those in [7] show that the accepted throughput degrades after saturation. This is because these schemes either use deadlock recovery techniques [18] or strict escape paths which drain the packets that may be involved in a deadlock. Hence, when deadlock starts to occur frequently after saturation, the throughput of the network degrades to the bandwidth of the deadlock free lanes or escape channels. The four oblivious algorithms — VAL, DOR, ROMM and RLB — use deadlock avoidance, i.e. they achieve deadlock freedom by ensuring that the channel dependency graph (see [6]) of all the virtual channels used is acyclic. Hence, they are stable after saturation. MIN AD and GOAL use the $*$-channels as the deadlock free escape paths for packets that maybe involved in a deadlock in the fully adaptive non-$*$ channels. However, these escape paths are not strictly meant for packets involved in a potential deadlock in the non-$*$ channels, i.e. packets entering the $*$-channels can always go back to the non-$*$ ones and vice versa. Hence, none of the adaptivity is lost and the throughput is sustained post saturation.

Figure 13. Accepted Throughput for BC traffic on (a) GOAL and (b) CHAOS.

Figure 14 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 algorithms because adaptivity is required to route around the congestion resulting from hot nodes. VAL gives throughput lower than 0.5 on hot-spot traffic because the traffic matrix is no longer admissible.

6 Conclusion

In this paper, we give a theoretical justification for the advantage of non-minimal routing that proves that any minimal routing algorithm can have worst-case performance at best half as good as a well balanced non-minimal algorithm on $k$-ary $n$-cubes. Based on that argument, we introduce a load-balanced, non-minimal adaptive routing algorithm for torus networks, GOAL, that achieves high throughput on adversarial traffic patterns while preserving locality on benign patterns. GOAL matches or exceeds the throughput of Valiant’s algorithm on adversarial patterns and exceeds the

$^8$These five hot-spot nodes are chosen very close to each other to stress the adaptivity of the algorithms.
GOAL globally balances network load by obliviously choosing the direction of travel in each dimension, in effect randomly picking a quadrant in which to transport the packet. The random choice of directions is made using distance-based weights that exactly balance load in each dimension. Once the quadrant is selected, GOAL locally balances load by routing adaptively within that quadrant. GOAL employs a new algorithm for deadlock freedom based on an extension of the *-channels approach [8] (which is for minimal routing) to handle the non-minimal case. This provides deadlock freedom with just 3 virtual channels per physical channel. Unlike CHAOS, GOAL is deterministically livelock free since within the selected quadrant distance to the destination is monotonically decreased with each hop.

We compare GOAL to four previously published oblivious algorithms, VAL, DOR, ROMM and RLB and two state-of-the-art adaptive routing methods CHAOS, and MIN AD, and present a comparison in terms of throughput, latency, stability, and hot-spot performance. This evaluation includes throughput histograms on random permutations and latency histograms for CHAOS and MIN AD that have not been previously reported. GOAL provides the highest throughput of the seven algorithms on four adversarial patterns and on the average and worst-case of 1,000 random permutations. The cost of this high worst-case throughput is a modest degradation on local traffic. GOAL achieves only 58% and 76% of the throughput of minimal algorithms on nearest neighbor traffic and uniform traffic respectively. Due to oblivious misrouting, GOAL also has 40% higher latency on random traffic than the minimal algorithms; however it has 38% lower latency than VAL. Finally, the paper analyzes network performance beyond saturation throughput and shows for the first time that due to fairness issues CHAOS is unstable in this regime for certain permutations.

The development of GOAL opens many exciting avenues for further research. Other topologies with path diversity, such as Cayley graphs and hierarchical networks, may benefit from GOAL-like routing algorithms if a method of globally balancing traffic on these topologies can be developed. A method to determine the worst-case traffic pattern for an adaptive routing algorithm, analogous to [17] for oblivious routing algorithms, would provide more precise determination of performance on adversarial traffic patterns. It is also interesting to ask what new topologies may be enabled by GOAL-like routing algorithms.

References