Locality-Preserving Randomized Oblivious Routing on Torus Networks

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

ABSTRACT

We introduce *Randomized Local Balance* (RLB), a routing algorithm that strikes a balance between locality and load balance in torus networks, and analyze RLB's performance for benign and adversarial traffic permutations. Our results show that RLB outperforms deterministic algorithms (25% more bandwidth than Dimension Order Routing) and minimal oblivious algorithms (50% more bandwidth than 2 phase ROMM [9]) on worst-case traffic. At the same time, RLB offers higher throughput on local traffic than a fully randomized algorithm (4.6 times more bandwidth than VAL (Valiant's algorithm) [15] in the best case). RLBth (RLB *threshold*) improves the locality of RLB to match the throughput of minimal algorithms on very local traffic in exchange for a 4% reduction in worst-case throughput compared to RLB. Both RLB and RLBth give better throughput than all other algorithms we tested on randomly selected traffic permutations. While RLB algorithms have somewhat lower guaranteed bandwidth than VAL they have much lower latency at low offered loads (upto 3.65 times less for RLBth).

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Routing and Layout.*; C.1.2 [**Processor Architectures**]: Multiple Data Stream Architectures—*Interconnection architectures.*

General Terms

Algorithms, Performance.

Keywords

Interconnection networks, k-ary n cubes, Locality-Preserving, Randomized, Oblivious packet routing.

1. INTRODUCTION

Interconnection networks based on a *torus* or k-ary n-cube topology [3] are widely used as switch and router fabrics [4], for processor-memory interconnect [12], and for I/O interconnect [10]. In many of these applications, it is essential that the interconnection network guarantee a minimal throughput regardless of the traffic pattern. 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 at line rate or packets will be dropped. At the same time, an efficient interconnection network should exploit locality to achieve high-performance and low power on local traffic patterns.

A routing algorithm must strike a balance between these conflicting goals of exploiting locality and providing high worst-case throughput. 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. Existing randomized routing algorithms based on Valiant's work [15] give good performance on worst-case traffic, but at the expense of completely destroying locality and hence giving very poor performance on local traffic.

In this paper, we introduce Randomized Local Balance (RLB) - a randomized oblivious routing algorithm for torus networks that strikes a balance between the conflicting goals of locality and load balance. Like Valiant's algorithm, RLB works by routing each packet to its destination by way of a randomly chosen intermediate node, q. However, to preserve locality, q is chosen so that for each dimension more traffic traverses the short direction than travels the *long way around*. To avoid certain adversarial patterns, RLB also travels in only a single direction in each dimension - avoiding backtracking - and selects the order in which dimensions

[*]Supported by the Richard and Naomi Horowitz Stanford Graduate Fellowship.
[†]Supported by an NSF Graduate Fellowship with supplement from Stanford University and under the MARCO Interconnect Focus Research Center.
[‡]Supported by a grant from the Stanford Networking Research Center (SNRC) in the School of Engineering at Stanford University.

SPAA'02, August 10-13, 2002, Winnipeg, Manitoba, Canada.

are traversed randomly.

Because RLB distributes traffic over a larger number of links it gives considerably better performance than minimal algorithms on worst-case traffic, providing 25% more bandwidth than dimension-order routing (DOR) and 50% more bandwidth than 2 phase ROMM [9] in the worst case. At the same time, because RLB exploits locality in its choice of an intermediate node, q, it outperforms a fully randomized algorithm by a factor of 4.6 on nearest neighbor traffic.

We further improve the locality of RLB by introducing a variant, RLB *threshold* (RLBth) that routes minimally in a given dimension if the distance in that dimension is less than a threshold. RLBth matches the performance of minimal algorithms on traffic patterns where the distance is below the threshold - providing 8 times the performance of VAL on these patterns. This is achieved at the expense of a modest 4% degradation in throughput on worst-case patterns.

Both RLB and RLBth give better average throughput on 10^6 random traffic permutations than VAL, DOR, or ROMM. At the same time, measures of individual packet latency show that RLB and RLBth provide this throughput with significantly lower latency than VAL.

Measurements of the throughput of variations of RLB indicate that most of its advantage is gained by its weighted random choice of direction in each dimension. Routing a fraction of the traffic the long way around each dimension effectively balances load for many worst-case patterns. Randomly choosing dimension order and picking a random intermediate node provide smaller improvements in performance.

The remainder of this paper describes RLB algorithms (RLB and RLBth) in more detail and evaluates their performance. Section 3 describes the RLB algorithms in detail. We measure the performance of RLB and RLBth in Section 4 and compare them to existing routing algorithms. Section 5 briefly describes previous randomized routing algorithms and puts RLB in context with this work. In Section 6, we discuss certain issues like packet reordering, deadlock and livelock.

2. PRELIMINARIES

The following discussions describe routing algorithms that are *oblivious*. That is, they select a path from source to destination that depends only on the source and destination nodes in order to route a packet, ignoring the state of the network. Oblivious algorithms may use randomization to select among alternative paths. We restrict our discussion to multi dimension torus networks or k-ary n-cube networks. A k-ary n-cube is a n dimension torus network with k nodes per dimension. Each link is unidirectional, so there are two links between any adjacent nodes - one for each direction.

We further assume that the network uses *store-and-forward* flow control with each node having buffers of infinite length. Contention between packets for the same outgoing link in a node is resolved using the *oldest packet first* protocol. Using this idealized model of flow control allows us to isolate the effect of the routing algorithm from flow control issues. The RLB algorithms can be applied to other flow control methods such as virtual channel flow control.

The saturation throughput λ is always normalized to the capacity of the network. The network capacity is the maximum load that the bisection of the network can sustain for uniformly distributed traffic and is given by $\frac{k}{8}$. All addition and subtraction on node coordinates is performed mod k yielding a result that is in the range $[0, k-1]$.

3. RANDOMIZED LOCAL BALANCE ROUTING - RLB AND RLBTH

This section describes the randomized local balance routing algorithms - RLB and RLB threshold (RLBth). We start by describing how to load balance a one-dimensional ring and then extend this concept to higher dimension tori.

3.1 Balancing a 1-Dimensional Ring

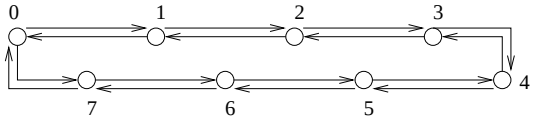

Figure 1: An 8 node ring (8-ary 1-cube).

To see why minimal routing is sub-optimal, consider a 8-node ring (8-ary 1-cube) topology (Figure 1) in which node i sends a message to node $i+3$. We refer to this traffic pattern as *tornado traffic* since with minimal routing the messages all rotate around the ring in a single direction like a tornado. As illustrated in Figure 2, with minimal routing, the clockwise link out of node i carries three messages, from i, $i-1$, and $i-2$. Hence, if the bandwidth of this link is b, the per-node throughput of the network on this traffic pattern is at most $\lambda = b/3 = 0.33b$.

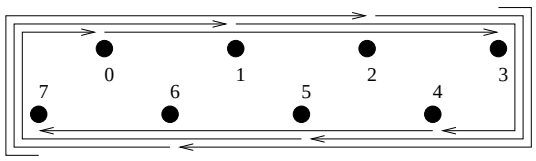

Figure 2: Minimally routed tornado traffic. Clockwise link load is 3. Counter clockwise link load is 0.

With this traffic pattern a minimal routing algorithm results in considerable load imbalance. All of the clockwise links are fully loaded while all of the counterclockwise links are idle.

We could ofcourse balance this traffic by randomizing the routing, sending from node i to a random intermediate node j and then from j to $i+3$. Each of these two phases is a perfectly random route and hence uses $k/4 = 2$ links on average for a total of 4 links traversed per packet. These links are evenly divided between the clockwise and counterclockwise rings, two each. Thus, even though we are traversing one more link on average than for minimal routing, the per-node throughput for randomized routing is higher, $\lambda = b/2 = 0.5b$.

The problem with purely randomized routing is that it destroys locality. For a nearest-neighbor traffic pattern, in which each node i sends half of its traffic to $i+1$ and half to $i-1$, throughput is still $\lambda = 0.5b$ while a minimal routing algorithm gives a throughput of $\lambda = 2b$ on nearest-neighbor traffic.

Now consider the tornado traffic pattern but with a non-minimal routing algorithm that sends 5/8 of all messages in the short direction around the ring - three hops clockwise - and the remaining 3/8 of all messages in the long, counterclockwise direction (see Figure 3). Each link in the clockwise direction carries 5/8 of the messages from 3 nodes for a total of 15/8 messages. Similarly each link in the counterclockwise direction carries 3/8 of the messages from 5 nodes and hence also carries a total of 15/8 messages. Thus, the traffic is perfectly balanced - each link has identical load. As a result of this load balance, the per-node throughput is increased by 60% to $\lambda = 8b/15 = 0.53b$ compared to that of a minimal scheme.

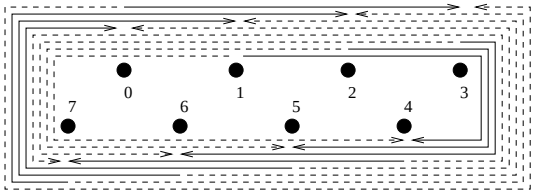

Figure 3: Non-minimally routing tornado traffic based on locality. The dashed lines contribute a link load of $\frac{3}{8}$ while the solid lines contribute a link load of $\frac{5}{8}$. All links equally loaded with load = $\frac{15}{8}$.

With *randomized local balance* (RLB) routing, if source node s sends traffic to destination node d then the distance in the short direction around the loop is $\Delta = \min(|s-d|, k-|s-d|)$ and the direction of the short path is $r = +1$ if the short path is clockwise, and $r = -1$ if the short path is counterclockwise. To exactly balance the load due to symmetric traffic we send each packet in the short direction, r, with probability $P_r = \frac{k-\Delta}{k}$ and in the long direction, $-r$, with probability $P_{-r} = \frac{\Delta}{k}$. This loads $k-\Delta$ channels in the long direction with load P_{-r} and Δ channels in the short direction with load P_r for a total load of $\frac{\Delta(k-\Delta)}{k}$ in each direction.

With nearest-neighbor traffic, for example, $\Delta = 1$, so $P_r = \frac{k-1}{k}$ so for $k = 8$, 7/8 of the traffic traverses a single link and 1/8 traverses seven links. On average each packet traverses $14/8 = 1.75$ channels - evenly distributed in the two directions - and hence throughput is $\lambda = 2b/1.75 = 1.14b$.

This simple comparison in one dimension shows the capability of RLB to give good performance on an adversarial traffic pattern. Here it achieves $0.53b$ on tornado traffic, much better than the $0.33b$ of a minimal algorithm, and it achieves $1.14b$ on nearest neighbor traffic, not as good as the $2b$ of a minimal algorithm, but much better than the $0.5b$ of fully random routing.

In order to improve RLB's performance on local traffic like nearest neighbor, we can modify the probability function of picking the short or long paths so that for very local traffic RLB always routes minimally. Specifically, if $\Delta < \frac{k}{4}$ (the average hop distance in a k node ring), then the message must be routed minimally. Hence, $P_r = 1$ and $P_{-r} = 0$ if $\Delta < \frac{k}{4}$, else P_r is the same as that in RLB. We call this modified version RLB threshold or RLBth. With this modification, RLBth achieves a throughput of 2b on nearest neighbor traffic while retaining a throughput of 0.53b on tornado traffic pattern.

3.2 RLB Routing in Two or More Dimensions

In multiple dimensions RLB works, as in the one dimensional case, by balancing load across multiple paths while favoring shorter paths. Unlike the one dimensional case, however, 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. RLB exploits this path diversity to balance load.

To extend RLB to multiple dimensions, we start by independently choosing a direction for each dimension just as we did for the one-dimensional case above. Choosing the directions selects the *quadrant* in which a packet will be routed in a manner that balances load among the quadrants. To distribute traffic over a large number of paths within each quadrant, we route first from the source node s to a randomly selected intermediate node q within the selected quadrant and then from q to the destination d. For each of these two phases we route in dimension order, traversing all of one dimension before starting on the next dimension, but randomly selecting the order in which the dimensions are traversed.

First, lets look at how we select the quadrant to route in by choosing a direction for each of the n dimensions in a k-ary n-cube. Suppose the source node is $s = \{s_1, s_2, \ldots, s_n\}$ and the destination node is $d = \{d_1, d_2, \ldots, d_n\}$, where x_i is the coordinate of node x in dimension i. We compute a distance vector $\Delta = \{\Delta_1, \Delta_2, \ldots, \Delta_n\}$ where $\Delta_i = min(|s_i - d_i|, k - |s_i - d_i|)$. From the distance vector, we compute a minimal direction vector $r = \{r_1, r_2, \ldots, r_n\}$, where for each dimension i, we choose r_i to be $+1$ if the short direction is clockwise (increasing node index) and -1 if the short direction is counterclockwise (decreasing node index). Finally we compute an RLB direction vector r' where for each dimension i we choose $r'_i = r_i$ with probability $P_{ri} = \frac{k-\Delta_i}{k}$ and $r'_i = -r_i$ with probability $1 - P_{ri} = \frac{\Delta_i}{k}$.

For example, suppose we are routing from $s = (0, 0)$ to $d = (2, 3)$ in a 8-ary 2-cube network (8×8 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 r', $(+1, +1)$, $(+1, -1)$, $(-1, +1)$, 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 $r' = (+1, +1)$ and less in the quadrant that takes the long path in both dimensions $r' = (-1, -1)$. Moreover, this weighting of directions will exactly balance the load for any traffic pattern in which node $s = (x, y)$ sends to node $d = (x + \Delta_x, y + \Delta_y)$ - a 2-D generalization of tornado traffic.

Once we have selected the quadrant we need to select a path within the quadrant in a manner that balances the load across the quadrant's channels. There are a large number of unique paths across a quadrant which is given by:

$$N_p = \prod_{i=0}^{n-2} \left(\begin{array}{c} \sum_{j=i}^{n-1} \Delta_j \\ \Delta_i \end{array} \right) \quad (1)$$

However, we do not need to randomly select among all of these paths. To balance the load across the channels, it suffices to randomly select an intermediate node q within

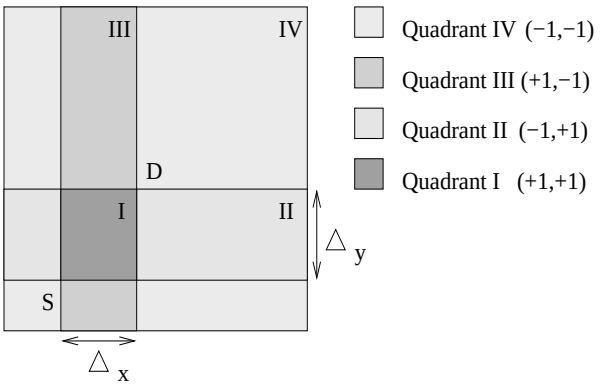

Figure 4: Probability distribution of the location of the intermediate node in RLB. (All nodes in a similarly shaded region (quadrant) have equal probability of being picked.)

the quadrant and then to route first from s to q and then from q to d. We then pick a random order of dimensions, o, for our route where o_i is the step during which the i^{th} dimension is traversed. We select this random ordering separately for both phases of routing. This is similar to the two-phase approach taken by a completely randomized algorithm. However, in this case the randomization is restricted to the selected quadrant.

It is important that the packet not backtrack during the second phase of the route, during which it is sent from q to d. If minimal routing were employed for the second phase, this could happen since the short path from q to d in one or more dimensions may not be in the direction specified by r'. To avoid backtracking, which unbalances the load, we restrict the routing to travel in the directions specified by r' during both routing phases: from s to q and from q to d. Figure 5 shows how the directions are fixed based on the quadrant the intermediate node q lies in.

We need to randomly order the traversal of the dimensions to avoid load imbalance between quadrant links, in particular the links out of the source node and into the destination. Figure 6 shows how traversing dimensions in a fixed order (say x first, then y) leads to a large imbalance between certain links.

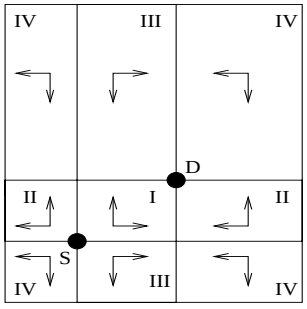

Figure 5: Example direction sets assigned to different quadrants on an 8-ary 2 cube

Suppose in our example above, routing from (0,0) to (2,3)

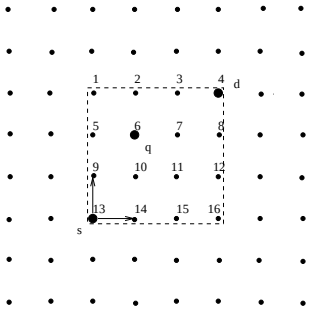

Figure 6: If one dimension (say x) is always traversed before the other(say y), all the links are not evenly balanced. Here, if q is in the boxed local quadrant, then the upward link 13-9 will only be used if q is one of nodes 9,5 or 1 while the right-going link 13-14 is used if q is any of the other nodes in the local quadrant. This, increases the likelihood of using 13-14 over 13-9 thereby unnecessarily overloading 13-14 .

in an 8-ary 2-cube, we select the quadrant $r' = (-1, +1)$. Thus, we are going in the negative direction in x and the positive direction in y. We then randomly select q_x from $[3, 4, 5, 6, 7, 0]$ and q_y from $[0, 1, 2]$. Suppose this selection yields intermediate point $q = (7, 1)$. Finally we randomly select an order $o = (1, 2)$ for the 1^{st} phase and also $o = (1, 2)$ for the 2^{nd} phase (note that the two orderings could have been different) implying that we will route in x first and then in y in both phases. Putting our choice of direction, intermediate node, and dimension order together gives the final route as shown in Figure 7. Note that if backtracking were permitted, a minimal router would choose the $+x$ direction after the first step since its only three hops in the $+x$ direction from q to d and five hops in the $-x$ direction.

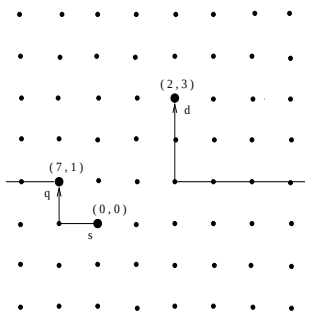

Figure 7: An example of routing using RLB.

Figure 8 shows how backtracking is avoided if directions are fixed for both the phases. The dotted path shows the path taken if Dimension Order Routing (traverse x dimension greedily, i.e. choosing the shortest path in that dimension and then traverse y dimension greedily) is followed in each phase when going from s to q to d. Fixing the direction sets based on the quadrant q is in, avoids the undesirable backtracking as shown by the bold path.

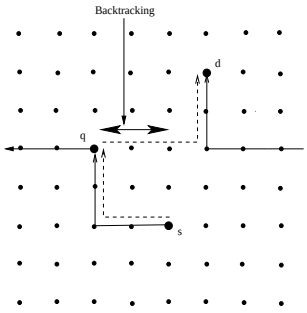

Figure 8: Avoiding backtracking in the RLB scheme. When the directions are fixed for both phases, routing is done along the bold path instead of the dotted path.

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

Table 1: Traffic patterns for evaluation of routing algorithms

3.3 RLBth in Two or More Dimensions

As in the one dimension case, RLBth works the same as RLB even for higher dimensions with a modification in the probability function for choosing the quadrants. Specifically, if $\Delta_i < \frac{k}{4}$, then $P_{r_i} = 1$ and $P_{-r_i} = 0$, else $P_{r_i} = \frac{k-\Delta_i}{k}$ and $P_{-r_i} = \frac{\Delta_i}{k}$. The threshold value of $\frac{k}{4}$ comes from the fact that it is the average hop distance for a k node ring in each dimension.

4. PERFORMANCE EVALUATION

4.1 Throughput of RLB on Various Traffic

We measure the saturation throughput of RLB on the six traffic patterns described in Table 1. The first two patterns are *benign* in the sense that they naturally balance load and hence give good throughput with simple routing algorithms. The next three patterns are *adversarial* patterns that cause load imbalance. These patterns have been used in the past to stress and evaluate routing algorithms. Finally, the worst-case pattern is the traffic permutation (selected over all possible permutations) that gives the lowest throughput. In general, the worst-case pattern may be different for different routing algorithms.

The latency-throughput curve for each traffic pattern (except NN) applied to an 8-ary 2-cube network with store and forward flow control using RLB is shown in Figure 9 [1]. Each curve starts at the y-axis at the zero load latency for that traffic pattern which is determined entirely by the number of hops required for the average packet and the packet length. As offered traffic is increased latency increases because of queueing due to contention for channels. Ultimately a point is reached where the latency increases without bound. The point where this occurs is the *saturation throughput* for the traffic pattern, the maximum bandwidth that can be input to each node of the network in steady state. The numerical values of this saturation throughput for each traffic pattern are given in Table 2.

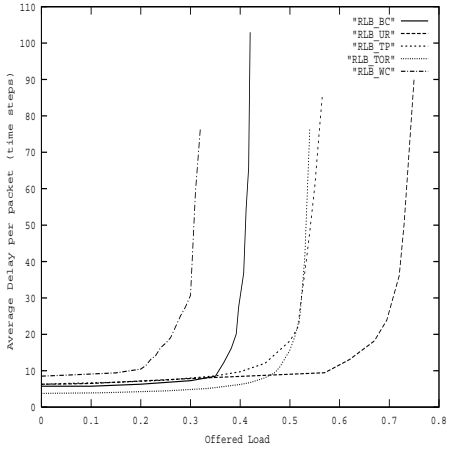

Figure 9: RLB delay-load curves for various traffic patterns.

4.2 Effect of Backtracking

In describing RLB in Section 3 we saw qualitatively that it was important to avoid backtracking during the second phase of routing. Table 2 shows quantitatively how backtracking affects performance. The first column shows the saturation throughput of RLB on each of the six traffic patterns - the asymptotes of the curves in Figure 9. The second column shows throughput on each traffic pattern using a variation of RLB in which backtracking is permitted. With this algorithm, after routing to intermediate node q, the packet is routed over the shortest path to the destination, not necessarily going in the same direction as indicated by the dotted lines in Figure 8.

The table shows that backtracking improves performance for the two benign cases but gives significantly lower performance on tornado and worst-case traffic. The improvement on benign traffic occurs because RLB with backtracking is closer to minimal routing - its traversing fewer hops than RLB without backtracking. The penalty paid for this is poorer performance on traffic patterns like TOR that require non-minimal routing to balance load.

We discuss some other variations on RLB in Section 4.4.

4.3 Comparison to Other Routing Algorithms

In this section, we compare the performance of RLB and

[1]The NN curve is omitted to allow the throughput scale to be compressed improving clarity.

Traffic	RLB	Backtrack
NN	2.33	2.9
UR	0.76	0.846
BC	0.421	0.421
TP	0.565	0.50
TOR	0.533	0.4
WC	0.313	0.27

Table 2: Saturation throughputs of RLB and its backtracking variation.

Name	Description
DOR	Dimension-order routing [13] - route in the minimal quadrant in x first, then in y.
ROMM	Two-phase ROMM [9] - route to random node q in minimal quadrant, then to destination.
VAL	Valiant's algorithm [15] - route to a random node q anywhere in the network, then to destination.

Table 3: Routing algorithms used in comparison against RLB

RLBth to that of the three oblivious routing algorithms listed in Table 3.

4.3.1 Throughput on Random Permutations

We compare the throughput of RLB and RLBth with VAL, ROMM, and DOR on 10^6 randomly selected permutations on an 8-ary 2-cube[2]. Histograms of the saturation throughput across the simulated permutations are shown in Figure 10. RLB has a smooth bell-shaped histogram centered at 0.51 throughput. RLBth's histogram (not shown) is almost identical to that of RLB but centered at 0.512. VAL achieves the same throughput on all traffic permutations. Hence its histogram is a delta function at 0.5. The histogram for ROMM is noisier and has an average saturation throughput of 0.453 - 12% lower than RLBth's throughput. DOR's histogram has three spikes at 0.25, 0.33 and 0.5 corresponding to a worst case link load of 4, 3 and 2 in any permutation. DOR's average saturation throughput is 0.314, 39% lower compared to RLBth. The average saturation throughputs are summarized in Table 4. RLB algorithms have higher average throughput on random permutations than VAL, ROMM, or DOR.

Algorithm	Average throughput
RLBth	0.512
RLB	0.510
VAL	0.500
ROMM	0.453
DOR	0.314

Table 4: Average Saturation Throughputs for 10^6 random traffic permutations.

[2]These 10^6 permutations are selected from the $N! = k^n!$ possible permutations on an N-node k-ary n-cube.

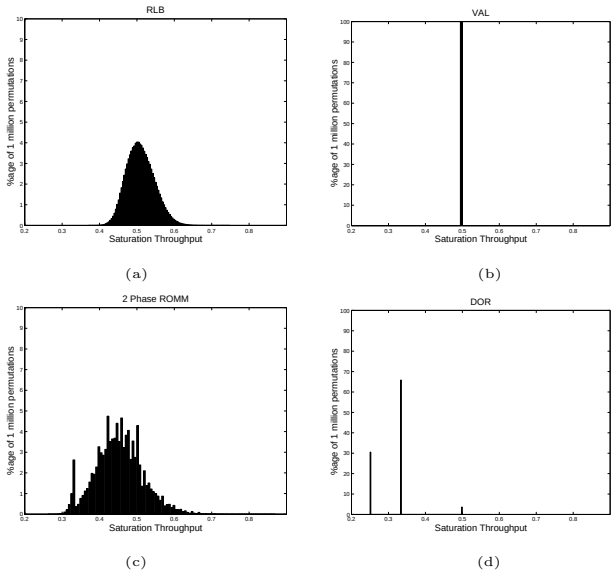

Figure 10: Histograms for the saturation throughputs for 10^6 random permutations. (a) RLB, (b) VAL, (c) ROMM, (d) DOR

4.3.2 Throughput on Specific Traffic Patterns

Table 5 shows the saturation throughput of each algorithm on each traffic pattern[3]. The minimal algorithms, DOR and ROMM, offer the best performance on benign traffic patterns but have very poor worst-case performance. VAL gives the best worst-case performance but converts every traffic pattern to this worst case giving very poor performance on the benign patterns. RLB strikes a balance between these two extremes achieving a throughput of 0.313 on worst-case traffic (50% better than ROMM and 25% better than DOR) while maintaining a throughput of 2.33 on NN (366% better than VAL) and 0.76 on UR (52% better than VAL). RLBth improves the locality of RLB - matching the throughputs of minimal algorithms in the best case and improving the UR throughput of RLB (64% better than VAL). In doing so, however, it marginally deteriorates RLB's worst case performance by 4%.

Figure 11 shows the latency-throughput curve for each of our five algorithms on nearest-neighbor (NN) traffic. RLBth, ROMM, and DOR share the same curve on this plot since they all choose a minimal route on this traffic pattern. The VAL curve starts at a much higher zero load latency because it destroys the locality in the pattern.

The latency throughput curves for each algorithm on bit complement (BC) traffic are shown in Figure 12. At almost all values of offered load, VAL has significantly higher latency. However, VAL has a higher saturation throughput than RLB.

The worst case row of Table 5 reflects the lowest throughput for each algorithm over all possible traffic patterns. The worst case throughput and traffic pattern (permutation) for each routing algorithm is computed using the method described in [1]. Using worst-case permutations for this evalu-

[3]The worst-case pattern is different for each algorithm. See Appendix A.

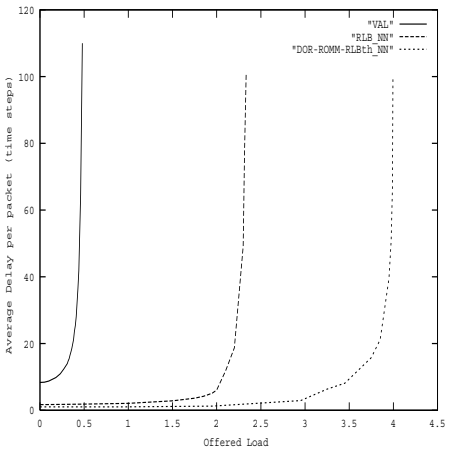

Figure 11: Performance of different algorithms on NN (Nearest neighbor) traffic.

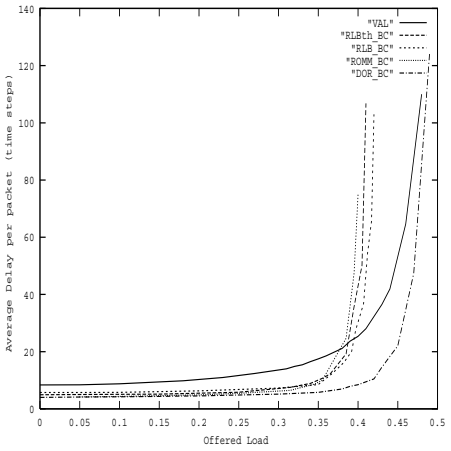

Figure 12: Performance of different algorithms on BC (Bit Complement) traffic.

ation is more accurate than picking some arbitrary adversarial traffic pattern (like BC, TP, or TOR) since the worst-case pattern for an algorithm is often quite subtle.

4.3.3 Latency

RLB gives a lower packet latency than fully randomized routing (VAL). To quantify this latency reduction, we computed latency histograms between representative pairs of source and destination in a network loaded with uniform random traffic for RLB, RLBth, VAL, ROMM, and DOR.

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 with load 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, $H \geq D$.

In an 8-ary 2-cube, 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

Traf	DOR	VAL	ROMM	RLB	RLBth
NN	4	0.5	4	2.33	4
UR	1	0.5	1	0.76	0.82
BC	0.50	0.5	0.4	0.421	0.41
TP	0.25	0.5	0.54	0.565	0.56
TOR	0.33	0.5	0.33	0.533	0.533
WC	0.25	0.5	0.208	0.313	0.30

Table 5: Comparison of saturation throughput of RLB, RLBth and three other routing algorithms on an 8-ary 2-cube for six traffic patterns.

packets from a source to 3 different destination nodes:

- 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 [4] in Figure 13. The histograms are computed by measuring the latency of 10^4 packets for each of these three pairs. For all experiments, offered load was held constant at 0.2. The experiment was repeated for each of the five routing algorithms. The histogram for DOR is almost identical to that of ROMM and is not presented.

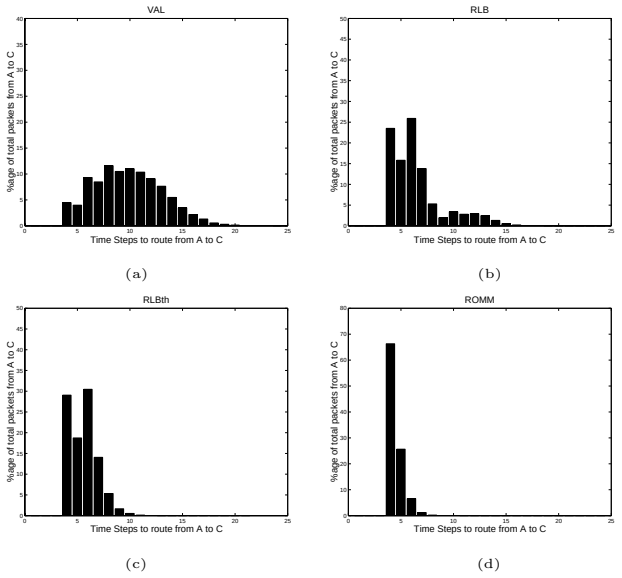

Figure 13: Histograms for 10^4 packets routed from node A(0,0) to node C(1,3). (a) VAL, (b) RLB, (c) RLBth, (d) ROMM. The network is subjected to UR pattern at 0.2 load.

DOR and ROMM have a distribution that starts at 4 and drops off exponentially - reflecting the distribution of queueing wait times. This gives an average latency of 4.28 and 4.43 respectively. Since both these algorithms always

[4]For the other sets of histograms see Appendix B.

route minimally, their H value is 4 and therefore, Q values are 0.28 and 0.43 respectively.

RLBth has a distribution that is the superposition of two exponentially decaying distributions: one with a H of 4 that corresponds to picking quadrant I of Figure 4 and a second distribution with lower magnitude starting at $H = 6$ that corresponds to picking quadrant II. The bar at $T = 6$ appears higher than the bar at $T = 4$ because it includes both the packets with $H = 6$ and $Q = 0$ and packets with $H = 4$ and $Q = 2$. The average H for RLBth is 4.75, giving an average Q of 0.81.

The distribution for RLB includes the two exponentially decaying distributions of RLBth corresponding to $H = 4$ and $H = 6$ and adds to this two additional distributions corresponding to $H = 10$ and $H = 12$ corresponding to quadrants III and IV of Figure 4. The probability of picking quadrants III and IV is low, giving the distributions starting at 10 and 12 very low magnitude. The average H for RLB is 5.5, giving an average Q of 0.98.

VAL has a very high latency with a broad distribution centered at $T = 9.78$. This broad peak is the superposition of exponentially decaying distributions starting at all even numbers from 4 to 12. The average H component of this delay is 8 since each of the two phases is a route involving a fixed node and a completely random node (4 steps away on average). The average Q is 1.78.

The results for all the three representative paths are summarized in Table 6. VAL performs the worst out of all the algorithms. It has the same high H and Q latency for all paths. DOR and ROMM being minimal algorithms, do the best at this low load of 0.2. They win because their H latency is minimal and at a low load their Q latency is not too high. RLB algorithms perform much better than VAL - in both H and Q values. RLB is on average 2.2 times, 1.5 times and 1.1 times faster than VAL on local, semi-local and non-local paths respectively. RLBth does even better by quickly delivering the very local messages - being 3.65 times, 1.76 times and 1.11 times faster than VAL on the same three paths as above.

4.4 Taxonomy of Locality-Preserving Randomized Algorithms

RLB performs three randomizations to achieve its high degree of load balance: (1) it randomly chooses a quadrant, and hence a direction vector for routing, (2) it randomly chooses an order in which to traverse the dimensions, and (3) it randomly chooses an intermediate waypoint node in the selected quadrant. We can generate eight non-backtracking, locality-perserving randomized routing algorithms by disabling one or more of these randomizations.

In this taxonomy of routing algorithms, each algorithm is characterized by a 3-bit vector. If the first bit is set the quadrant is chosen randomly (weighted to favor locality). Otherwise the minimal quadrant is always used. If this bit is clear the routing algorithm will be minimal. All non-minimal algorithms have random quadrant selection. The dimensions are traversed in a random order, if the second bit is set and in a fixed order (x first, then y, etc...) if this bit is clear. Finally, the third bit, if set, causes the packet to be routed first to a random waypoint in the selected quadrant and then to proceed to the destination - without reversing direction in any dimension. For example a vector of 111 corresponds to RLB - all randomizations enabled and a vector of 000 corresponds to DOR - no randomization. By examining the points between these two extremes we can quantify the contribution to load balance of each of the three randomizations.

Table 7 describes the eight algorithms and gives their performance on our six traffic patterns[5]. All four minimal algorithms have same high performance on the benign traffic patterns (NN and UR) since they never misroute. The first randomization we consider is the order of dimensions. Vector 010 gives us dimension order routing with random dimension order - e.g., in 2-D we go x-first half the time and y-first half the time. This randomization eases the bottleneck on transpose, doubling performance on this pattern, but does not affect worst-case performance. So we can see that randomizing dimension order alone does not improve worst-case performance.

Next, let us consider the effect of a random waypoint in isolation. Vector 001 gives us ROMM, in which we route to a random waypoint in the minimal quadrant and then on to the destination. This randomization, while it improves performance on Transpose, actually reduces worst-case throughput and throughput on bit complement. This is because the choice of a random waypoint concentrates traffic in the center of a region for these patterns. Combining random directions with a random waypoint, vector 011, while it improves Transpose further does not affect the other patterns. Thus, routing to a random waypoint alone actually makes things worse, not better.

Finally, we will consider the non-minimal algorithms. Vector 100 corresponds to random direction routing (RDR) in which we randomly select directions in each dimension, in effect selecting a quadrant, and then use dimension-order routing within that quadrant. As described in Section 3, this selection is weighted to favor locality. Randomly selecting the quadrant by itself gives us most of the benefits (and penalties) of RLB. We improve worst-case performance by 14% compared to the best minimal scheme, and we get the best performance of any non-minimal algorithm on bit complement. However, performance on transpose suffers, it is equal to worst-case, due to the fixed dimension order. Randomizing the dimension order, vector 110, fixes the transpose problem but does not affect the other numbers.

Routing first to a random waypoint within a randomly-selected quadrant, vector 101, gives slightly better worst-case performance 24% better than minimal and 8% better than RDR. However using a random waypoint makes transpose and bit complement worse. Putting all three randomizations together, which yields RLB as described in Section 3, gives slightly better worst-case, transpose, and nearest-neighbor performance.

Overall, the results show that randomization of quadrant selection has the greatest impact on worst-case performance. Non-minimal routing is essential to balance the load on adversarial traffic patterns. Once quadrant selection is randomized, the next most important randomization is selection of a random waypoint. This exploits the considerable path diversity within the quadrant to further balance load. However, applying this randomization by itself actually reduces worst-case throughput. The randomization of dimension order is the least important of the three having little impact on worst-case throughput. However, if a random

[5]The worst-case pattern is not the same for all eight algorithms.

Algorithm	T_{A-B}	H_{A-B}	Q_{A-B}	T_{A-C}	H_{A-C}	Q_{A-C}	T_{A-D}	H_{A-D}	Q_{A-D}
DOR	2.3	2	0.3	4.28	4	0.28	8.24	8	0.24
ROMM	2.34	2	0.34	4.43	4	0.43	8.42	8	0.42
RLBth	2.68	2	0.68	5.56	4.75	0.81	8.81	8	0.42
RLB	4.31	3.5	0.81	6.48	5.5	0.98	8.92	8	0.92
VAL	9.78	8	1.78	9.78	8	1.78	9.78	8	1.78

Table 6: Average total, hop and queueing latency (in Time Steps) for 10^4 packets for 3 sets of representative traffic paths at 0.2 load. $A-B$, $A-C$ and $A-D$ represent local, semi-local and non-local paths respectively. All other nodes send packets in a uniformly random manner at the same load.

Vector	Description	NN	UR	BC	Tpose	Tor	WC
000	DOR-F - dimension-order routing	4	1	0.5	0.25	0.33	0.25
010	DOR-R - with randomized dimension order	4	1	0.5	0.5	0.33	0.25
001	ROMM-F - fixed dimension order - route first to a random node q in the minimal quadrant and then to the destination	4	1	0.4	0.438	0.33	0.208
011	ROMM-R - random dimension order - like 001 but the order in which dimensions are traversed is randomly selected for both phases.	4	1	0.4	0.54	0.33	0.208
100	RDR-F - randomly select a quadrant (weighted for locality) and then route in this quadrant using a fixed dimension order	2.28	0.762	0.5	0.286	0.533	0.286
110	RDR-R - with random dimension order	2.286	0.762	0.5	0.571	0.533	0.286
101	RLB-F - with fixed dimension order	2.286	0.762	0.421	0.49	0.533	0.310
111	RLB-R	2.33	0.76	0.421	0.565	0.533	0.313

Table 7: Taxonomy of locality preserving randomized algorithms. Saturation throughputs are presented for a 8-ary 2 cube topology.

waypoint is not used, randomizing dimension order doubles throughput on traffic patterns like Transpose.

5. PREVIOUS WORK

Dimension-order routing (DOR), sometimes called e-cube routing, was first reported by Sullivan and Bashkow [13]. With DOR routing, each packet first traverses the dimensions one at a time, arriving at the correct coordinate in each dimension before proceeding to the next. Because of its simplicity it has been used in a large number of interconnection networks [5, 11]. The poor performance of dimension-order routing on adversarial traffic patterns motivated much work on adaptive routing.

Valiant first described how to use randomization to provide guaranteed throughput for an arbitrary traffic pattern [15]. His method perfectly balances load by routing to a randomly selected intermediate node (phase 1) before proceeding to the destination (phase 2). Dimension order routing is used during both phases. While effective in giving high guaranteed performance on worst-case patterns, this algorithm destroys locality - giving poor performance on local or even average traffic.

In order to preserve locality while gaining the advantages of randomization, Nesson and Johnson suggested ROMM [9], - Randomized, Oblivious, Multi-phase Minimal routing. ROMM randomly selects one of the minimal paths for each packet. While [9] reports good results on a few permutations, we have shown here that ROMM actually has lower worst-case throughput than DOR. The problem is that it is impossible to achieve good load balance on adversarial patterns, such as tornado traffic, with minimal routing.

Adaptive routing is an alternative method of dealing with adversarial traffic. Several adaptive routing algorithms have been developed for torus networks [6, 8, 2]. An adaptive routing algorithm based on [6] was employed in the Cray T3E for this reason [12]. However, most of these proposed adaptive routing methods balance load locally but not globally. They would all route tornado traffic along minimal routes giving poor performance.

6. DISCUSSION

6.1 Deadlock and livelock

RLB algorithms, while non-minimal, are inherently livelock free. 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.

As stated in Section 2, we assume a store and forward flow control with unbounded buffers for the results presented in this paper so deadlock due to channel or buffer dependency is not an issue. The results here can be extended to virtual channel flow control by using multiple virtual networks each employing a variant of the turn model ([7]). However, such an extension is beyond the scope of this paper.

6.2 Packet Reordering

The use of a randomized routing algorithm can and will

cause out of order delivery of packets. While this may be acceptable for multiprocessor systems with a relaxed memory coherence model, memory systems with strict coherence and internet routers require in-order delivery.

Several methods can be used to guarantee in order delivery of packets where needed. One approach is to ensure that packets that must remain ordered (e.g., memory requests to the same address or packets that belong to the same flow) follow the same route. This can be accomplished, for example, by using a packet group identifier (e.g., the memory address or the flow identifier) to select the intermediate node for the route. Packet order can also be guaranteed by reordering packets at the destination node. For example, the well known sliding window protocol [14] can be used for this purpose.

7. CONCLUSION

Randomized Local Balance (RLB) is a non-minimal oblivious algorithm that balances load by randomizing three aspects of the route: the selection of the routing *quadrant*, the order of dimensions traversed, and the selection of an intermediate waypoint node. RLB weights the selection of the routing quadrant to preserve locality. The probability of misrouting in a given dimension is proportional to the distance to be traversed in that dimension. This exactly balances traffic for symmetric traffic patterns like tornado traffic. RLBth is identical to RLB except that it routes minimally in a dimension if the distance in that dimension is less than a threshold value ($\frac{k}{4}$).

RLB strikes a balance between randomizing routes to achieve high guaranteed performance on worst-case traffic and preserving locality to maintain good performance on average or neighbor traffic. On worst-case traffic RLB outperforms all minimal algorithms achieving 25% more throughput than dimension-order routing and 50% more throughput than ROMM, a minimal oblivious algorithm. The worst-case throughput of RLB, however, is 37% lower than the throughput of a fully randomized routing algorithm. This degradation in worst-case throughput is balanced by a substantial increase in throughput on local traffic. RLB (RLBth) outperforms VAL by 4.6 (8) on nearest-neighbor traffic and 1.52 (1.69) on uniform random traffic. RLBth improves the locality of RLB, matching the performance of minimal algorithms on local traffic, at the expense of a 4% degradation in worst-case throughput.

RLB algorithms do not match the worst-case throughput of a fully randomized algorithm, achieving 62% of the worst case throughput of VAL. However, both RLB and RLBth give higher saturation throughput on average for 10^6 random traffic permutations. Also, RLB and RLBth provide much lower latency, upto 3.65 times less, than VAL.

By selectively disabling the three sources of randomization in RLB we are able to identify the relative importance of each source. Our results show that the advantages of RLB are primarily due to the weighted random selection of the routing quadrant. Routing a fraction of the traffic the long way around each dimension effectively balances load for many worst-case patterns. By itself, randomly choosing dimension order has little effect on worst-case performance and by itself, picking a random intermediate node actually reduces worst-case throughput.

The development of RLB opens up many exciting avenues for future work in locality-preserving routing algorithms. Studying the worst-case permutations for RLB indicates that it should be possible to get even higher performance by allowing limited routing outside the selected quadrant - particularly for quadrants with high aspect ratios. We are also interested in applying some of the principles of RLB, in particular weighted random quadrant selection, to adaptive algorithms and in comparing the performance guarantees of adaptive and oblivious algorithms.

8. REFERENCES

[1] B.Towles and W.J.Dally. Worst-case traffic for oblivious routing functions. *Computer Architecture Letters*, 1, Feb 2002. http://www.cs.virginia.edu/ tcca.

[2] W. Dally. Aoki: Deadlock-free adaptive routing in multicomputer networks using virtual channels, 1993.

[3] W. J. Dally. Performance analysis of k-ary n-cube interconnection networks. *IEEE Transactions on Computers*, 39(6):775–785, 1990.

[4] William Dally, Philip Carvey, and Larry Dennison. Architecture of the avici terabit switch/router. In *Proceedings of Hot Interconnects Symposium VI, August 1998*, pages 41–50, 1998.

[5] William J. Dally and Charles L. Seitz. The torus routing chip. *Distributed Computing*, 1(4):187–196, 1986.

[6] Jose Duato. New theory of deadlock-free adaptive routing in wormhole networks. *IEEE Transactions on Parallel and Distributed Systems*, 4(12):1320–1331, 1993.

[7] Christopher J. Glass and Lionel M. Ni. The turn model for adaptive routing. In *25 Years ISCA: Retrospectives and Reprints*, pages 441–450, 1998.

[8] D. Linder and J. Harden. An adaptive and fault tolerant wormhole routing strategy for k-ary n-cubes, 1991.

[9] Ted Nesson and S. Lennart Johnsson. ROMM routing on mesh and torus networks. In *Proc. 7th Annual ACM Symposium on Parallel Algorithms and Architectures SPAA'95*, pages 275–287, Santa Barbara, California, 1995.

[10] G. Pfister. An introduction to the infiniband arechitecture. High Performance Mass Storage and Parallel I/O, IEEE Press, 2001., 2001.

[11] S. Scott and G. Thorson. Optimized routing in the Cray T3D. *Lecture Notes in Computer Science*, 853:281–294, 1994.

[12] S. Scott and G. Thorson. The cray t3e network: adaptive routing in a high performance 3d torus, 1996.

[13] H. Sullivan, T. Bashkow, and D. Klappholz. A large scale, homogeneous, fully distributed parallel machine, ii, 1977.

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

[15] L. G. Valiant. A scheme for fast parallel communication. *SIAM Journal on Computing*, 11(2):350–361, 1982.

Appendix A: Worst case permutations for the algorithms described

In this Appendix, we enumerate the worst case permutations for each of the five algorithms we have used in Table 5.

- Dimension Order : The transpose traffic permutation - (i, j) sends to (j, i)- is a worst-case permutation for this scheme. This skewed loading pattern overloads the last right-going link of the 1^{st} row resulting in an offered bandwidth of 0.25 the network capacity.
- Valiant : Any traffic permutation is the worst case permutation.
- 2 phase ROMM : The following is the worst case permutation that [1] obtains which gives a saturation load of 0.208 the network capacity. Figure 14 shows the destination of each source node (i, j) in the worst case permutation.

	0	1	2	3	4	5	6	7
0	(0,3)	(0,0)	(7,5)	(7,0)	(7,6)	(4,1)	(0,1)	(0,2)
1	(6,3)	(0,5)	(0,6)	(0,7)	(1,0)	(5,1)	(6,1)	(6,2)
2	(7,3)	(6,0)	(5,4)	(5,2)	(4,6)	(7,7)	(7,1)	(7,2)
3	(7,4)	(1,5)	(1,6)	(1,7)	(2,0)	(6,7)	(6,6)	(6,5)
4	(0,4)	(4,7)	(4,2)	(5,0)	(2,4)	(5,7)	(5,6)	(5,5)
5	(1,4)	(2,5)	(2,6)	(2,7)	(3,0)	(5,3)	(4,5)	(4,3)
6	(1,3)	(4,4)	(3,2)	(3,3)	(3,4)	(6,4)	(1,1)	(1,2)
7	(2,3)	(3,5)	(3,6)	(3,7)	(4,0)	(3,1)	(2,1)	(2,2)

Figure 14: Worst case traffic permutation for 2 phase ROMM. Element $[i, j]$ of the matrix gives the destination node for the source node (i, j).

- RLB : The following (figure 15) is the worst case permutation that [1] obtains which gives a saturation load of 0.313 the network capacity.

	0	1	2	3	4	5	6	7
0	(0.1)	(0,0)	(4,1)	(3,1)	(1,1)	(7,1)	(0,2)	(0,3)
1	(0,4)	(5,0)	(6,6)	(2,6)	(5,1)	(6,1)	(7,2)	(7,3)
2	(7,4)	(6,0)	(3,7)	(4,4)	(4,2)	(5,2)	(6,2)	(6,3)
3	(7,5)	(7,6)	(7,7)	(5,5)	(3,5)	(5,4)	(5,3)	(6,4)
4	(0,7)	(7,0)	(5,6)	(4,5)	(4,6)	(2,7)	(6,5)	(2,5)
5	(0,6)	(6,7)	(4,7)	(1,6)	(4,0)	(3,4)	(2,4)	(1,5)
6	(1,4)	(5,7)	(1,7)	(2,0)	(4,3)	(3,3)	(2,2)	(2,3)
7	(0,5)	(1,0)	(3,6)	(3,0)	(3,2)	(2,1)	(1,2)	(1,3)

Figure 15: Worst case traffic permutation for RLB. Element $[i, j]$ of the matrix gives the destination node for the source node (i, j)

- RLBth : The worst case permutation for RLBth is very similar to that for RLB and is not presented.

Appendix B: Latency at low load

In this Appendix, we present the histograms for average latency for two source destination pairs, A(0,0) to B(1,1) and A(0,0) to D(4,4) (see Figure 16) representing local and non-local paths. The rest of the network is subjected to uniform random traffic at load 0.2. Minimal algorithms do best at this load while completely randomized algorithms like VAL do very poorly especially for local paths.

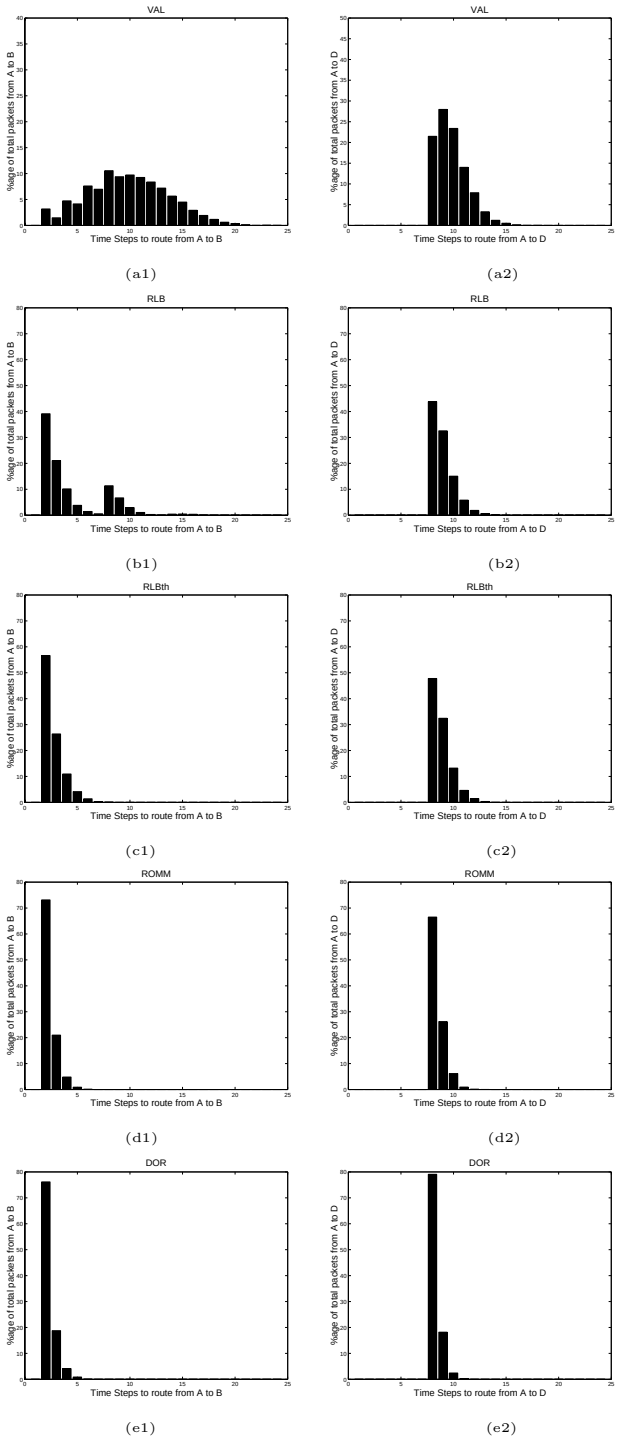

Figure 16: Latency histograms for 10^4 packets. (a) VAL, (b) RLB, (c) RLBth, (d) ROMM (e)DOR, 1 - from node A (0,0) to B (1,1), 2 - from node A (0,0) to D (4,4)