Adaptive Channel Queue Routing on k-ary n-cubes *

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

ABSTRACT

This paper introduces a new adaptive method, Channel Queue Routing (CQR), for load-balanced routing on k-ary n-cube interconnection networks. CQR estimates global congestion in the network from its channel queues while relying on the implicit network backpressure to transfer congestion information to these queues. It uses this estimate to decide the directions to route in each dimension. It further load balances the network by routing in the selected directions adaptively. The only other algorithm that uses global congestion in its routing decision is the Globally Adaptive Load-Balance (GAL) algorithm introduced in [13]. GAL performs better than any other known routing algorithm on a wide variety of throughput and latency metrics. However, there are four serious issues with GAL. First, it has very high latency once it starts routing traffic non-minimally. Second, it is slow to adapt to changes in traffic. Third, it requires a complex method to achieve stability. Finally, it is complex to implement. These issues are all related to GAL's use of injection queue length to infer global congestion. CQR uses channel queues rather than injection queues to estimate global congestion. In doing so, it overcomes the limitations of GAL described above while matching its high performance on all the performance metrics described in [13]. CQR gives much lower latency than GAL at loads where non-minimal routing is required. It adapts rapidly to changes in traffic, is unconditionally stable, and is simple to implement.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Routing and Layout.*

General Terms

Algorithms, Performance

Keywords

Adaptive Channel Queue Routing, k-ary n-cubes

1. INTRODUCTION

Interconnection networks are widely used for processor-memory interconnect [10], for I/O interconnect [9], and as switch and router fabrics [3]. Torus, or k-ary n-cube [2], topologies are very popular in all of these applications. Torus networks have high path diversity, offering many alternative paths between a source and destination node which a routing algorithm must choose from.

A good routing algorithm must balance the conflicting constraints of locality and load balance. To achieve minimum latency on local traffic patterns (e.g., nearest-neighbor), a routing algorithm must exploit locality - routing packets over a minimum-length path. On the other hand, on difficult patterns (e.g., tornado - see below) at high loads, the routing algorithm must balance load by routing some traffic over longer, non-minimal, paths to achieve maximum throughput.

To get optimal performance, the decision on how to route a packet depends on the load caused by other traffic. For example, with tornado traffic at low loads all packets should be routed minimally to give low latency, while at high traffic, some traffic must be routed non-minimally to balance load. Thus, to give best performance, the *global* routing decision must be *adaptive*. An *oblivious* routing algorithm [15, 12] is unable to base its decision on other load in the network. Moreover, to balance load, this adaptive decision must be global in nature - choosing between minimal and non-minimal routes - not just choosing between channels to fine-tune a route.

The GAL (Globally Adaptive Load-balanced) routing algorithm, introduced in [13], provides this global adaptivity by basing the routing decision on the length of injection queues associated with sets of destinations. GAL provides near-optimal throughput - matching the throughput of minimal algorithms on local patterns and of load-balanced algorithms on difficult patterns. GAL is the only known algorithm that performs best on a wide range of throughput and

*This work has been supported in part by the Richard and Naomi Horowitz Stanford Graduate Fellowship (Singh), a grant from the Stanford Networking Research Center (Gupta), an NSF Graduate Fellowship, a grant from the MARCO Interconnect Focus Research Center at Stanford University (Towles) and a grant from Cray, Inc. through DARPA Subcontract CR03-C-0002, under U.S. Government Prime Contract Number NBCH3039003.

SPAA'04, June 27–30, 2004, Barcelona, Spain.

latency metrics. However, there are four serious issues with GAL. First, it has very high latency once it starts routing traffic non-minimally. Second, it is slow to adapt to changes in traffic. Third, it requires a complex method to achieve stability. Finally, it is complex to implement. These issues are all related to GAL's use of injection queue length to infer global congestion.

This paper introduces channel queue routing (CQR - pronounced "seeker") which matches the ability of GAL to achieve high throughput on both local and difficult patterns but overcomes the limitations of GAL. CQR gives much lower latency than GAL at loads where non-minimal routing is required. It adapts rapidly to changes in traffic, is unconditionally stable, and is simple to implement.

CQR overcomes the limitations of GAL by using *channel* queues rather than *injection* queues to estimate global congestion. CQR estimates the congestion of a quadrant by the sum of the lengths of the channel queues leading to that quadrant. For each packet, CQR chooses the quadrant (minimal or non-minimal) based on these estimates of quadarant congestion - with a threshold to bias the choice toward shorter paths. Once a quadrant is selected, minimal adaptive routing is used to select the path within the quadrant to deliver the packet to its destination.

The remainder of this paper describes CQR in more detail. In Section 2 we motivate the need to balance locality and load balancing by considering the model problem of routing tornado traffic on a ring. A simple approximate queueing model shows the behavior needed to minimize latency. We then show that GAL has much higher latency on this model problem because it routes too much traffic minimally at high loads. We introduce CQR routing in Section 3 and show that it closely matches the optimal profile on our model problem. In Section 4 we show via a series of experiments that CQR has much better latency, stability, and transient response time than GAL.

2. MOTIVATION: ROUTING TORNADO TRAFFIC ON A 1-DIMENSIONAL RING

For *adversarial* traffic patterns, the focus of an adaptive routing algorithm must change as load is increased. At low loads, packets should be routed minimally in order to minimize delays. As loads increase, minimizing delay requires shifting some packets to non-minimal routes in order to balance channel load. In this section, we derive the optimal adaptive routing algorithm for the tornado traffic pattern on a ring network and compare it to the performance of GAL. While GAL matches this optimal algorithm at both very low loads and loads near saturation, it transitions too slowly from minimal to non-minimal at intermediate loads, resulting in delays approximately 100 cycles more than optimal. This behavior is representative of GAL on all adversarial patterns because its adaptation does not begin until the delays of the network are already large.

2.1 Routing Tornado Traffic to Balance Load

We consider an adversarial traffic pattern (called *tornado* or TOR) on a simple 8-node ring network (Figure 1). In this pattern, node i sends all traffic nearly half-way-around the ring, to node $i + k/2 - 1$.[1] We will consider this pattern on a simple 8-node ring network (Figure 1). Minimally routing TOR results in all traffic traversing the clockwise links like a tornado, leaving the counterclockwise links completely idle (Figure 2). The clockwise link from node i carries traffic from three nodes: i, $i-1$, and $i-2$. Assuming unit bandwidth channels, each node can inject traffic at a rate of at most 1/3 before the clockwise links become saturated. Thus, the throughput is only $\Theta = 1/3$ with minimal routing.

[1]All additions and subtractions on node coordinates are done modulo k ($k = 8$ for an 8 node ring).

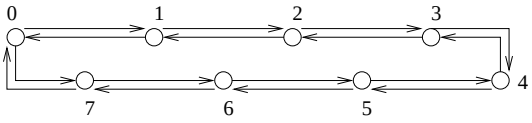

Figure 1: An 8 node ring (8-ary 1-cube).

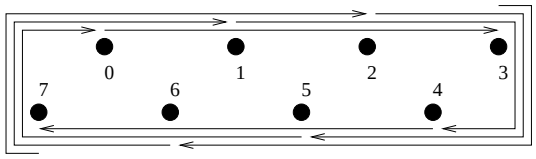

Figure 2: Minimally routed tornado traffic. Clockwise link load is 3. Counter clockwise link load is 0.

To maximize the throughput under tornado, non-minimal routing is required to balance load: each node must send 5/8 of its traffic minimally and 3/8 of its traffic non-minimally at the saturation load as shown in Figure 3. The resulting throughput is $\Theta = 8/15$ and is 60% higher than the throughput under minimal routing.

While this routing approach maximizes throughput, it would unnecessarily increase delay if used at loads below saturation. Next we discuss how the fraction of traffic routed minimally should change as a function of injected load for optimal delay. In order to understand the problem better, we first present an approximate theoretical model from queueing theory.

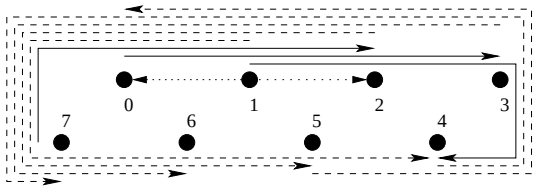

Figure 3: Routing tornado traffic for maximum throughput. Load is shown for a clockwise and a counter clockwise link depicted in dotted lines. The dashed lines contribute a link load of $\frac{3}{8}$ while the solid lines contribute a link load of $\frac{5}{8}$. All links are equally loaded with a load of $\frac{15}{8}$.

2.2 A Model from Queueing Theory

Queueing networks have been studied extensively in literature. A popular way to study a network of queues is

to use the Jackson open queueing network model [6]. This model assumes a *product-form*[2] network in which each queue is an independent M/M/1 queue. However, it differs from the standard model by assuming that service times, instead of being constant, are exponentially distributed with unit mean. Based on an idea first considered by Kleinrock [7], Mitzenmacher [8] observed that assuming each queue to be an independent M/D/1 queue is a good approximation. While the approximate model no longer strictly remains a product-form network, the results prove accurate in practice.

For our queueing model, we assume each queue to be an independent M/D/1 queue with arrival rate λ and service rate 1. The queueing delay of a packet in such a queue is given by [1],

$$Q(\lambda) = \frac{1}{2(1-\lambda)}$$

If a packet traverses a linear network of m such queues, it incurs a queueing delay due to m queues and a hop delay of m. Its total delay is approximated by

$$L^m(\lambda) = mQ(\lambda) + m.$$

We shall use this approximate model in the rest of this discussion.

2.3 Routing Tornado Traffic for Optimal Delay

When routing TOR optimally, there will be two classes of packets — those that are routed minimally and those sent along the non-minimal path. Let the injection load for each node be λ and the rates at which each node sends packets minimally and non-minimally be x_1 and x_2, respectively. Then, below saturation,

$$\lambda = x_1 + x_2 \tag{1}$$

Packets sent minimally (non-minimally) must traverse 3 (5) M/D/1 queues each with an arrival rate of 3λ (5λ).[3] Hence, the delay encountered by a packet sent minimally is given by

$$D_1(x_1) = L^3(3\lambda) = \frac{3}{2(1-3\lambda)} + 3 \tag{2}$$

and the delay of a non-minimally routed packet is given by

$$D_2(x_2) = L^5(5\lambda) = \frac{5}{2(1-5\lambda)} + 5. \tag{3}$$

Then the average delay is simply a weighted sum of these delays,

$$D(\lambda) = \frac{1}{\lambda}[x_1 D_1(x_1) + x_2 D_2(x_2)]. \tag{4}$$

We now analyze the fraction of traffic sent minimally and non-minimally to minimize delay as described in [1]. Clearly, $x_1 \geq x_2$ since it makes no sense sending more traffic along the non-minimal path.

[2]A network is product-form if in equilibrium distribution each queue appears as though it were an independent process with Poisson arrivals.

[3]At steady state and due to the symmetry of the network and the traffic pattern, we assume all nodes route traffic identically.

CLAIM 1. *In order to minimize the average packet delay, each node must start sending traffic non-minimally at an injection load of* $\lambda_s = 0.14$.

PROOF. For minimum delay, each node should send all traffic along the minimal path until the incremental delay of routing minimally is greater than that of routing non-minimally. The routing will be strictly minimal as long as

$$\frac{\partial D(\lambda)}{\partial x_1} \leq \frac{\partial D(0)}{\partial x_2} \tag{5}$$

Solving Equations (4) and (5), we get $\lambda \leq 0.14$. Hence, the switch point from strictly minimal to non-minimal occurs at $\lambda_s = 0.14$.

□

CLAIM 2. *The optimal fraction of traffic sent minimally and non-minimally is given by Figure 4.*

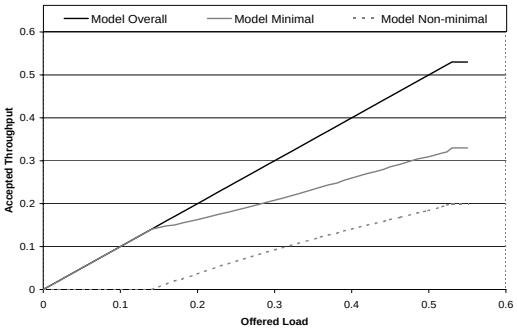

Figure 4: The optimal fraction of traffic in the minimal and non-minimal paths for tornado traffic on an 8-node ring.

PROOF. Once each node starts to send traffic non-minimally ($\lambda > \lambda_s$), the optimal fraction of load sent minimally (x_1) and non-minimally (x_2) will be such that the incremental delay along either directions is the same:

$$\frac{\partial D(x_1)}{\partial x_1} = \frac{\partial D(x_2)}{\partial x_2}. \tag{6}$$

Solving Equations (6), (1) numerically results in the plots shown in Figure 4. $x_1 = \lambda$ and $x_2 = 0$ until $\lambda = \lambda_s$. After that, both $x_1, x_2 > 0$ and are consistent with Equation (6). Finally, after the network saturates at $\lambda = 0.53$, $x_1 = 0.33$ and $x_2 = 0.2$. □

Substituting the values of x_1 and x_2 in Equations (2) and (3) we get the average minimal, non-minimal and overall delay of the packets according to our model. As shown in Figure 5, the average delays along both paths are similar and give a low overall average latency.

2.4 GAL on Tornado Traffic

The GAL algorithm [13] has per-destination injection queue sets in every node to sense network load imbalance. Each of queue set is separated into a *minimal* and a *non-minimal* injection queue. GAL routes a packet minimally if the occupancy of the minimal injection queue associated with the

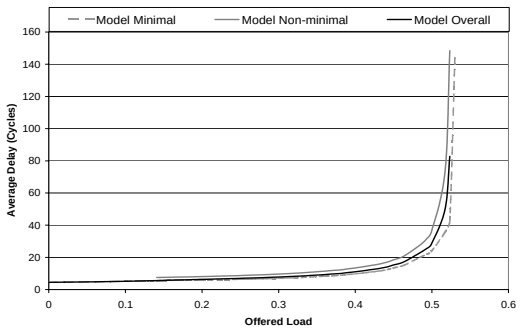

Figure 5: Optimal minimal, non-minimal and overall latency of the theoretical model for tornado traffic on an 8-node ring.

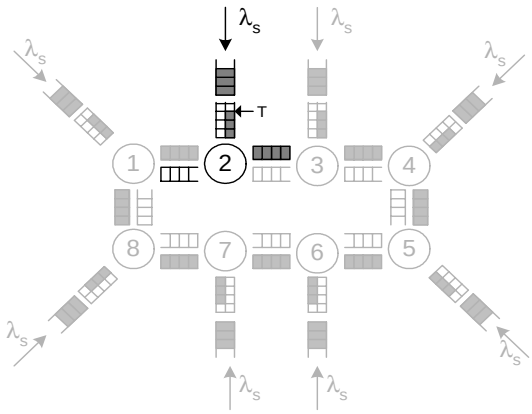

Figure 6: GAL on tornado traffic on an 8 node ring at the point when it switches from minimal to non-minimal. Only 1 set of injection queues corresponding to the destination is shown.

packet's destination is below a threshold. Otherwise, the threshold is exceeded and the packet is injected into the non-minimal injection queue and routed non-minimally.

With this approach, GAL routes packets minimally until the capacity along the minimal path is saturated. When this happens, the queues along the minimal path start to fill up and the network back-pressure implicitly transfers this information to the injection queues in the source node. When the occupancy of the minimal injection queue surpasses a threshold, T, packets are injected into the non-minimal injection queue and the routing switches from strictly minimal to non-minimal. This switching does not happen until all the queues along the minimal path are completely filled as shown in Figure 6. Hence, the load at which this switch occurs is simply the saturation load for strictly minimal routing of the tornado pattern and is given by $\lambda_s = 0.33$. Figure 7 shows how GAL starts to send traffic non-minimally only after the injection load, $\lambda \geq 0.33$. The accepted throughput at saturation is still the optimal value of 0.53.

However, the subtle point to note is that while GAL routing is throughput-optimal on the tornado pattern, delay in the switching from strictly minimal to non-minimal incurs

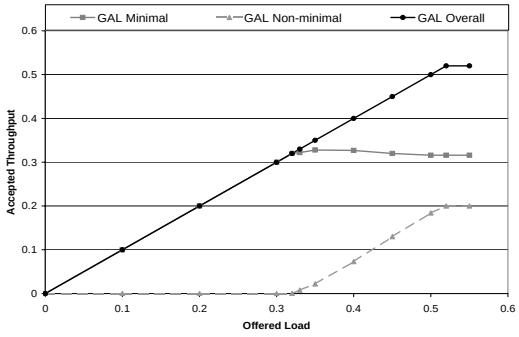

Figure 7: GAL throughput for tornado traffic on an 8 node ring

a high price as far as the latency is concerned.

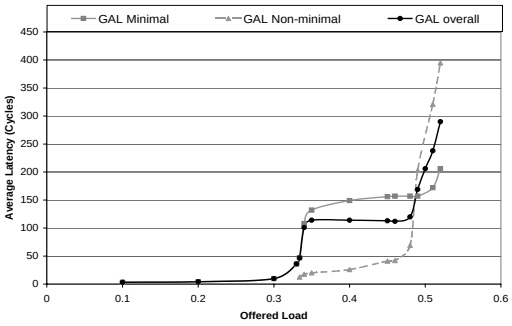

Figure 8: Latency-load plot for GAL on tornado traffic on an 8 node ring

Figure 8 shows the average minimal, non-minimal and overall packet delay for GAL. Since GAL routes minimally all the way to the saturation point for the strictly minimal routes, the latency incurred by the minimal packets after a load of 0.33 is very high. This causes the average overall packet delay, which is a weighted average of the minimal and non-minimal delays, to be of the order of 100s of cycles for a 8 node network much before the network saturation point of 0.53.

3. ADAPTIVE CHANNEL QUEUE ROUTING (CQR)

3.1 A Case for Channel Queue Routing

We observe that the reason GAL performs so poorly on tornado traffic is that it waits too long to switch its policy from strictly minimal to non-minimal. It does so because its congestion sensing mechanism uses the occupancy of the injection queues which is not very responsive to network congestion. CQR addresses this problem by sensing network load imbalance using its channel queues while at the same time relying on the network's implicit backpressure to propagate information from further parts of the network. Consider the highlighted node 2 in Figure 9. For the tornado traffic pattern, the minimal and non-minimal queues

for this node are the clockwise and counterclockwise outgoing queues, respectively. The occupancy of both these queues are labeled as q_m and q_{nm}. The node calculates the average queue occupancy $\overline{q} = (q_m + q_{nm})/2$. Then, if $q_m - \overline{q} < T$, a threshold value, the packet is routed minimally else it is sent along the non-minimal path. Such a congestion sensing scheme is more responsive than the one used in GAL and unlike GAL, the switch occurs much before all the minimal queues are filled as shown in Figure 9. We call this routing method Channel Queue Routing (CQR).

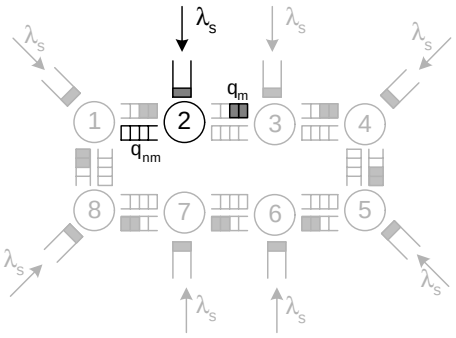

Figure 9: CQR on tornado traffic on an 8 node ring at the point when it switches from minimal to non-minimal.

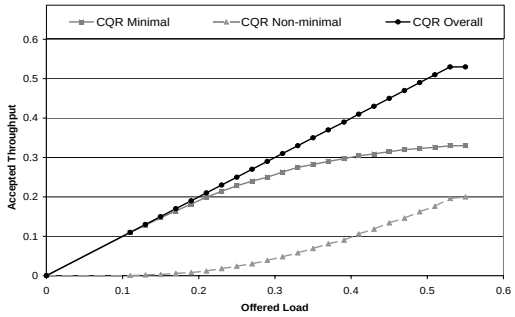

Figure 10: CQR throughput on tornado traffic on an 8 node ring

Figure 10 shows the fraction of traffic sent the minimal and non-minimal way as the injected load increases. The switch point is very close to the the one that we derived in or model. Since, the switch point for CQR on TOR is near-optimal, it gives a much smaller average overall delay as shown in Figure 11. The latency-load curve is very similar to the one we derived in Figure 5. It should be noted that the plots do not exactly match the model that we described as the model itself is based on an approximation and the queues are not strictly a network of independent M/D/1 queues.

3.2 CQR on Higher Dimension Torus Networks

Unlike the one dimensional case, where there are just two possible paths for each packet — one short and one long — there are many possible paths for a packet in a multi-

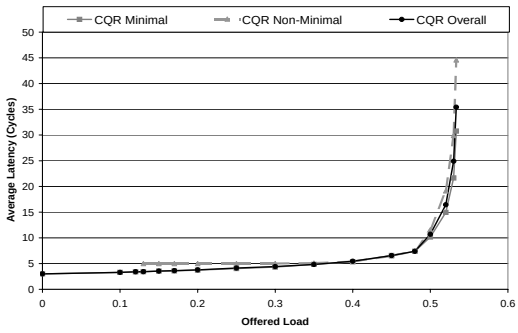

Figure 11: Latency-load plot for CQR on tornado traffic on an 8 node ring

dimensional network. We divide these possible paths based on the directions (+ or −) in each dimension. Hence, for a 2 dimension network, there are 4 quadrants (++, +−, −+ and −−) for each source-destination pair. Figure 12 shows the 4 quadrants for the source-destination pair $(0,0),(2,3)$. Quadrant I is the minimal quadrant while the others are non-minimal.

First, let us look at how we select the quadrant to route 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 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). Choosing a quadrant to route in simply means choosing a quadrant vector q where for each dimension i we could choose $q_i = r_i$ ($q_i = -r_i$) if we want to route minimally (non-minimally) in that dimension. In order to get approximate quadrant congestion information just like in the 1 dimension case, each node records its outgoing channel queue occupancies along both directions (+ and −) in each dimension. Then the congestion, Q_j, for quadrant j is approximated by the sum of the outgoing queues in the corresponding directions in each dimension given by q. Next the average quadrant congestion, $\overline{Q} = \frac{\sum_{j=1}^{2^n} Q_j}{2^n}$ is computed. Finally, the quadrant m with the smallest distance from s to d whose congestion satisfies the inequality $Q_m - \overline{Q} < T$, (where T is a threshold value), is selected.

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

For example, consider the case above where we are routing from $s = (0,0)$ to $d = (2,3)$ in an 8-ary 2-cube network. Suppose the minimal quadrant congestion is small enough such that it is less than a threshold above the average quadrant congestion. Then the algorithm routes the packet completely within quadrant I $(+1,+1)$. One possible route of the packet is shown in bold in Figure 13. On the first hop, the productive dimension vector is $p = (1,1)$ that is both the x and y dimensions are productive. Suppose the channel

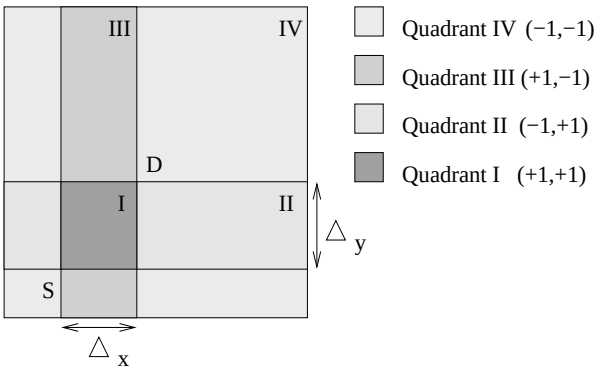

Figure 12: Quadrants in a k-ary 2-cube for a given source S (0,0) and destination D (2,3).

queue in the x dimension is shorter so the packet proceeds to node (1,0). At (1,0) p is still (1,1) so the packet can still be routed in either x or y. At this point, suppose the queue in the y dimension is shorter, so the packet advances to node (1,1). At (1,1) p is still (1,1) and this time the route is in x to (2,1). At this point, the packet has reached the destination coordinate in x so $p = (0, 1)$. Since the only productive dimension is y, the remaining hops are made in the y dimension regardless of queue length. A second possible route is shown in dashed as an example of routing in quadrant II $(-1, +1)$.

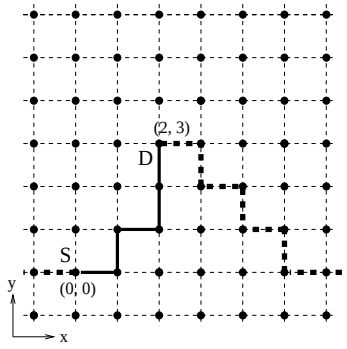

Figure 13: Example route from S (0,0) to D (2,3) through the minimal quadrant (+1,+1) and a non-minimal quadrant (-1,+1).

3.3 Virtual Channels and Deadlock

Our implementation of CQR requires 3 virtual channels (VCs) per unidirectional physical channel to achieve deadlock freedom in the network. This is an extension of the scheme proposed in the $*$-channels algorithm [5] for wormhole flow control developed for the non-minimal GOAL algorithm. For a proof of deadlock freedom for the scheme, refer to [11].

4. PERFORMANCE EVALUATION

In this section, we present results on steady-state throughput, latency at intermediate loads, stability, and transient response to dynamic traffic.

4.1 Experimental Setup

Measurements in this section have been made on a cycle-accurate network simulator in which the routing decision one cycle for each algorithm. Each node has an infinite source queue to model the network interface. The router is output queued and queues packets into one of the 3 (16 flit deep) VC buffers per physical output channel. Each packet is assumed to be one flit long to separate the routing algorithm study from flow control issues. The total buffer resources are held constant across all algorithms (the product of the number of VCs and the VC channel buffer depth is kept constant). The threshold value for CQR used in the experiments is 2.0. All contention is resolved using age-based arbitration, always giving priority to a packet with an older time-stamp since injection. All latency numbers presented are measured since the time of birth of the packets and include the time spent by the packets in the source queues. We have simulated two topologies, an 8-ary 2-cube and a 16-ary 2-cube, but present only the results for the 8-ary 2-cube topology due to space constraints. The results obtained for the 16-ary 2-cube topology follow the same trends.

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

4.2 Summary of Steady-State Results

Both CQR and GAL are non-minimal, adaptive algorithms that combine the best features of minimal algorithms on benign traffic and of load-balancing algorithms on adversarial traffic patterns. Table 1 shows how these two algorithms incorporate the plus points of a minimal adaptive algorithm (MIN AD) and a load-balancing one called GOAL. MIN AD (or the $*$-channels algorithm [5]) always routes in the minimal quadrant, routing adaptively within it. GOAL (Globally Oblivious Adaptive Locally [11]) obliviously chooses a quadrant, q, to route in according to a weighted probability distribution and then routes within q adaptively.

	MIN AD	GOAL	GAL	CQR
Θ_{benign}	1.0	0.75	1.0	1.0
Θ_{hard}	0.33	0.53	0.53	0.53
Θ_{avg}	0.63	0.67	0.73	0.7
Θ_{HS}	0.46	0.48	0.49	0.49
Low Load Latency	4.45	6.17	4.45	4.45

Table 1: Table summarizing the advantages of CQR and GAL over a minimal algorithm (MIN AD) and a load-balancing one (GOAL). The throughput is normalized to network capacity and latency is presented in cycles.

The table shows that at low loads and on a representative benign traffic pattern, Uniform Random (UR), (Nearest Neighbor traffic is another benign traffic pattern), CQR and GAL route all traffic minimally and thus match the low latency and high throughput (Θ_{benign}) of minimal routing

algorithms on such "friendly" traffic. On a *hard* traffic pattern, TOR, (and also patterns such as Transpose and Bit Complement[4]) that cause load imbalance in the network, CQR and GAL route minimally at low loads and then switch to non-minimal routing as load imbalance is detected. Thus, both CQR and GAL combine the best features of minimal algorithms (low latency at low load and high throughput on benign traffic) and obliviously load balanced algorithms (high throughput on adversarial traffic). GAL has the highest average throughput across a sample of $1,000$ random permutations (depicted as Θ_{avg}). Finally, since all the algorithms in the table are adaptive, they are able to route around hot-spots giving high throughput on *Hot-Spot* traffic (shown as Θ_{HS}).

To summarize, CQR is able to exactly match the performance of GAL across all the figures of merit described above. The only place where CQR's performance marginally degrades over that of GAL is that its average throughput over $1,000$ random permutations is 0.70 compared to 0.73 for GAL. In Section 2, we have already discussed the tremendous improvement of CQR over GAL in latency when it has to adapt to route non-minimally at loads close to saturation. In the rest of this section, we focus on CQR's latency at intermediate loads, stability and transient response to dynamic traffic.

4.3 Latency at Intermediate Loads

Probably the biggest improvement of CQR over GAL is that it delivers packets with much lower latency than GAL for patterns which require the algorithms to route non-minimally for optimal throughput. This effect is more pronounced for injection loads closer to saturation when the non-minimal paths have to be used to give higher system throughput. Section 2 shows an example of CQR's improvement over GAL giving upto 20× lower average delay than GAL on TOR in the injection load range of $0.30 - 0.53$.

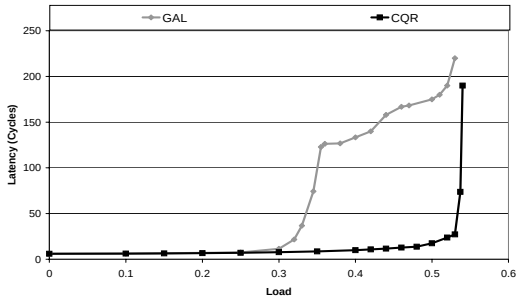

Figure 14: Latency profile of CQR and GAL for the 2D-tornado traffic.

Figure 14 shows how the advantage of CQR over GAL extends to a two dimension network — an 8-ary 2-cube. The traffic pattern is the 2-dimension version of the tornado traffic on a ring i.e., source (i, j) sends to $(i+k/2-1, j+k/2-1)$. GAL's latency-load plot rises up in *steps* which correspond to the points where a particular quadrant saturates and GAL starts routing along the next non-minimal quadrant. There are only 2 such steps for 3 non-minimal quadrants as quadrants 2 and 3 are identical in terms of the distance from the source to the destination. In contrast, CQR has a very sharp latency profile giving upto 15× lower average delay than GAL in the injection load range of $0.30 - 0.53$. Finally, Figure 15 shows a similar latency-load profile for GAL and CQR on the 8-ary 2-cube network on a randomly picked traffic permutation.

[4]For details of all the traffic patterns refer to [11].

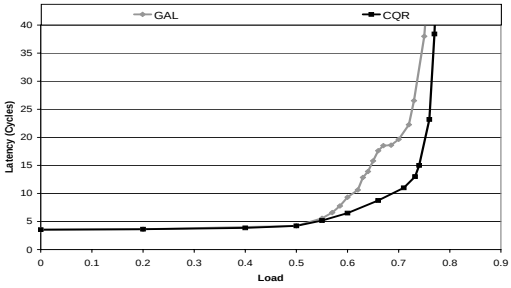

Figure 15: Latency profile of CQR and GAL for a random permutation.

4.4 Stability

An important property of any routing algorithm is its stability. An algorithm is stable if the accepted throughput remains constant even as the offered load is increased beyond the saturation point. In order to be stable, non-minimal, adaptive algorithms must distinguish between congestion caused by load imbalance and congestion caused by saturation of a load balanced network. In the former case, rerouting traffic, possibly non-minimally, can alleviate this imbalance and increase throughput. However, in the load balanced, but saturated case, rerouting traffic may introduce imbalance and decrease throughput. We examine a simple example to demonstrate the necessity of controlling the fraction of packets sent non-minimally.

Consider the case of routing UR traffic with GAL. For an offered load beyond saturation, traffic will initially be routed minimally as the minimal injection queues are empty. Since the network cannot sustain all the traffic, backpressure will cause packets to back up into the minimal injection queues. Although minimal routing perfectly balances load for UR traffic, the occupancy of these queues will eventually exceed the threshold and some traffic will be routed non-minimally. At this point, throughput will begin to drop as channel bandwidth is wasted by non-minimal packets (Figure 16(a)). If, instead, packets had not been allowed to route non-minimally, throughput would have remained stable.

The reason GAL is unstable beyond saturation for a traffic pattern that does not require non-minimal routing is that the source queue traffic interferes with the routing decision made using the injection queue occupancy.[5] Unlike GAL, CQR makes its routing decision using the *channel* queues as described in Section 3. Hence, it does not mix up the congestion due to load imbalance and that due to post-saturation injection load and is therefore, stable (Figure 16(b)).

[5]A way around this problem is to adaptively change the value of the threshold while monitoring the throughput delivered for each source-destination pair (see [13]). However, the scheme is both expensive and complex to implement.

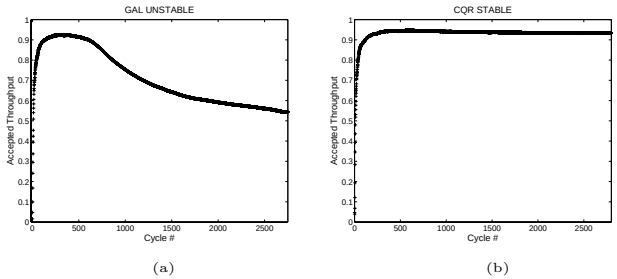

Figure 16: Performance of GAL and CQR on UR traffic at 1.1 offered load. (a) Throughput decreases over time for GAL as it is unstable post saturation (b) CQR is stable post saturation.

4.5 Dynamic Traffic Response

Traditionally, most adaptive routing algorithms in interconnection networks have been evaluated with static traffic patterns and only their steady-state performance has been studied [5, 4, 11]. However, adaptive routing algorithms are characterized by both a steady-state and a transient response. In this section, we demonstrate how CQR's routing mechanism gives much better transient response than GAL.

4.5.1 Step Response

The first experiment we perform is to subject the network to a traffic pattern that requires the algorithm to adapt from minimal to non-minimal routing — tornado traffic. The time taken for the algorithm to adapt to congestion is both a function of the routing decision mechanism and the per-channel flow control. In order to focus on the routing decision, we keep the channel queues for each algorithm the same (16 flit buffers per virtual channel in our experiments). We then impose the tornado traffic pattern at a load of 0.45 at cycle 0 and measure the step response of GAL and CQR.

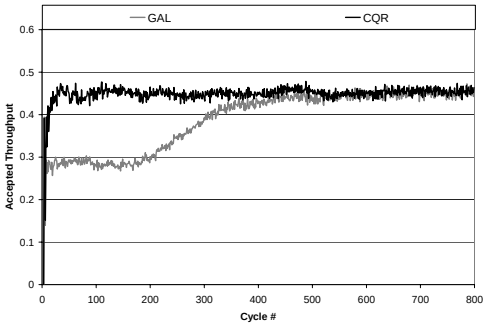

Figure 17: Transient response of CQR and GOAL to the tornado traffic pattern at 0.45 load started at cycle 0.

Figure 17 shows the step response of both algorithms averaged over 100 runs. The response of GAL is much slower than CQR taking as much as 4× the number of cycles to reach peak throughput. This is because in order to adapt to routing non-minimally, GAL must wait till the minimal injection queues fill up and surpass their threshold. However, since CQR senses load imbalance using channel queues, its step response is smooth reaching peak throughput in just 30 cycles.

4.5.2 Barrier Model

In the next transient experiment, we simulate the communication between the processors in a multi-processor environment. In this model we assume that the nodes of a multi-processor have a fixed amount of computation to perform between synchronization primitives called *barriers*. Each processor waits till every processor has completed its communication. This model assumes that all the packets are in the node's source queue before starting instead of the usual Poisson-like injection process as used in the traffic patterns described before. The number of cycles that each processor waits is then a function of how fast the routing algorithm adapts to network congestion.

In order to stress the adaptivity of the two algorithms, we choose the destinations of each packet according to the 2D-tornado traffic pattern — i.e., each node (i, j) sends to $(i + \frac{k}{2} - 1, j + \frac{k}{2} - 1)$. The number of packets that a node has to communicate is called the *batch size*. The latency measured is the number of cycles taken for all the processors to completely route their batch of packets. We report this latency normalized to the batch size.

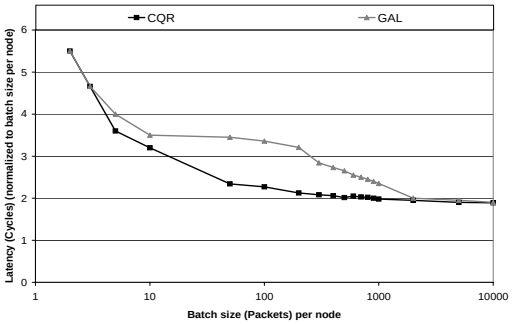

Figure 18: Dynamic response of CQR and GAL v.s. the batch size per node showing CQR's faster adaptivity.

Figure 18 shows the response of CQR and GAL as the batch size is swept from 2 to 10^4 packets (shown in logarithmic scale). For small batch sizes the latency is the same for both the algorithms as there is little congestion in the network. For extremely large batch sizes, the latency is again the same and is given by the reciprocal of the throughput of the algorithm on the 2D-tornado traffic (0.53 for both CQR and GAL). The interesting region in the plot is when the batch size ranges from 5 to $2,000$ packets. This region underscores the importance of the more responsive congestion sensing mechanism of CQR which gives as much as 1.5× performance improvement over GAL.

4.6 Livelock and Packet Ordering

Livelock is a condition whereby a packet keeps circulating within the network without ever reaching its destination. Freedom from such a critical condition must be guaranteed. Minimal algorithms like MIN AD guarantee livelock freedom with fair arbitration since each channel traversed by a packet

reduces the distance to the destination. Non-minimal algorithms like GOAL, GAL and CQR also provide deterministic freedom from livelock. Once a quadrant has been selected for a packet, the packet monotonically makes progress in that quadrant, reducing the number of hops to the destination at each step. Since there is no incremental misrouting, all packets reach their destinations after a predetermined, bounded number of hops.

The use of an adaptive routing algorithm can and will cause out of order delivery of packets. If an application requires in-order packet delivery, a possible solution is to reorder packets at the destination node using the well known sliding window protocol [14].

5. CONCLUSION

In this paper we have introduced channel queue routing (CQR), a globally adaptive routing algorithm for Torus, k-ary n-cube topology, networks. Like GAL, CQR senses global congestion and makes an adaptive global routing decision to route minimally at low loads and on local traffic while routing non-minimally to load balance difficult traffic patterns at high loads. CQR, like GAL, matches the throughput of minimal algorithms on local patterns, and load-balanced oblivious algorithms on difficult patterns — something that other algorithms, that do not make a global adaptive decision, cannot do.

Channel queue routing overcomes a number of issues associated with GAL. Most importantly, the latency for CQR at loads that require non-minimal routing is much lower than for GAL. This is because it does not need to run the minimal traffic well into saturation, with the resulting high latencies, before switching to non-minimal routing. CQR starts sending packets along non-minimal routes as soon as the delays due to minimal and non-minimal routing are matched — resulting in minimum latency.

CQR also has a much faster transient response than GAL. This is because the channel queue rapidly reflects global congestion while the injection queue (used for sensing in GAL) requires that all of the queues along the minimum paths fill up before sensing congestion. CQR is also unconditionally stable while GAL requires a threshold adaptation mechanism to give stable performance. Finally, CQR is very simple to implement compared to GAL.

In the future we plan to look at extending the benefits of CQR to other network topologies. We plan to generalize the concepts of quadrants to sets of routes in an arbitrary topology and to develop methods based on channel queues to sense congestion in these path sets.

6. REFERENCES

[1] D. Bertsekas and R. Gallager. *Data Networks: Second Edition*. Prentice-Hall, 1992.

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

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

[4] P. Gaughan and S. Yalamanchili. Adaptive routing protocols for hypercube interconnection networks. *IEEE Computer*, 26(5):12–23, May 1993.

[5] L. Gravano, G. Pifarre, G. Pifarre, P. Berman, and J. Sanz. Adaptive deadlock- and livelock-free routing with all minimal paths in torus networks. *IEEE Transactions on Parallel and Distributed Systems*, 5(12):1233–1252, Dec. 1994.

[6] J. Jackson. Jobshop-like queueing systems. *Management Science*, 10:131–142, 1963.

[7] L. Kleinrock. *Queueing Systems: Volume II*. John Wiley, 1975.

[8] M. Mitzenmacher. Bounds on the greedy routing algorithm for array networks. In *Proc. Symposium on Parallel Algorithms and Architectures SPAA*, pages 346–353, Cape May, New Jersey, June 1994.

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

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

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

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

[13] A. Singh, W. J. Dally, B. Towles, and A. K. Gupta. Globally adaptive load-balanced routing on tori. *Computer Architecture Letters*, 3, Mar 2004.

[14] A. 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.