Stanford University Concurrent VLSI Architecture Memo 121

Stanford University
Computer Systems Laboratory

Finding Worst-case Permutations for Oblivious Routing Algorithms

Brian Towles

Abstract

We present an algorithm to find a worst-case traffic pattern for any oblivious routing algorithm on an arbitrary interconnection network topology. The linearity of channel loading offered by oblivious routing algorithms enables the problem to be mapped to a bipartite maximum-weight matching, which can be solved in polynomial time. Finding exact worst-case performance was previously intractable, and we demonstrate an example case where traditional characterization techniques overestimate the throughput of a particular routing algorithm by 47%.

Keywords: oblivious routing, worst-case traffic, maximum-weight matching

*B. Towles is with the Computer Systems Laboratory in the Department of Electrical Engineering, Stanford University. This work has been supported by an NSF Graduate Fellowship with supplement from Stanford University and under the MARCO Interconnect Focus Research Center. E-mail: btowles@cva.stanford.edu. A brief version of this report has been submitted to Computer Architecture Letters for review.

1 Introduction

As interconnection networks are applied to throughput-sensitive applications, such as packet routing [1] and I/O interconnect [2], the worst-case behavior of a routing function becomes an important design consideration. Specifically in the packet router application, little can be said about the incoming traffic patterns, and there is no path for backpressure to slow the flow of incoming packets. Therefore, the guaranteed throughput of the router is bounded by the worst-case throughput over all traffic patterns. Obviously, a system designer would like to be able to characterize this worst-case situation.

This report presents an efficient technique for finding an exact worst-case pattern for any oblivious routing function on an arbitrary network topology (Section 3). By exploiting the linearity of oblivious routing functions, finding the worst-case traffic pattern can be cast as the maximum-weight matching of a bipartite graph. This graph problem can be solved in polynomial time, quickly yielding exact worst-case results. The solutions are then used to determine the worst-case throughput of a particular system.

This approach can offer a significant improvement in accuracy over existing techniques. Previous studies of routing algorithms generally chose "bad" traffic patterns that the authors felt represented worst-case or near worst-case behavior [3][4]. However, for the example presented in Section 5, the traditional techniques overestimate the worst-case throughput of the ROMM routing algorithm [3] by approximately 47%. Worst-case characterization has also been approached from a theoretical perspective [5][6][7], and while providing strong results, these analyses do not provide exact throughput values for specific topologies and routing algorithms. With the algorithms presented in this report, we hope to enable more quantitative studies of oblivious routing algorithms in the future.

2 Preliminaries

2.1 Network model

The interconnection networks discussed in this report have an arbitrary topology and fixed length data units. We refer to these units as packets, but any fixed size network unit, such as flits or cells, is equivalent. In order to isolate the effects of routing on network throughput, an ideal flow-control technique is assumed. Ideal flow-control ensures that the most heavily loaded channels are 100% utilized. The throughput of the network with ideal flow-control is an upper-bound on the throughput of any actual network, and practical flow-control techniques can typically achieve 60-75% of this bound [8].

2.2 Definitions

Topology:

- N - The number of nodes in the network.
- C - The set of all channels in the network.
- *isomorphic graphs* - Two graphs G and H are isomorphic if there exists a labeling function such that a relabeling of the nodes of G yields a graph identical to H.
- *automorphism* - Any isomorphic labeling of a graph onto itself.
- *edge-symmetric graph* - A graph G is edge-symmetric if for every pair of edges u and v, there exists an automorhpism on G that maps u to v.

Network traffic:

- *traffic matrix* (Λ) - Any doubly-stochastic[1] matrix where entry $\lambda_{i,j}$ represents the fraction of traffic traveling from source i to destination j. An $N \times N$ doubly-stochastic matrix has row and column sums of one:

$$\sum_{i=1}^{N} \lambda_{i,k} = \sum_{j=1}^{N} \lambda_{k,j} = 1, \quad \forall k \in \{1, \ldots, N\}$$

- *permutation traffic* (P) - A traffic matrix where the entries are either 0 or 1.

Routing functions:

- *oblivious routing algorithm* (π) - A routing algorithm that is only a function of the source and destination node of a packet. Oblivious routing algorithms can also be randomized, where a particular route is randomly chosen from a set of possible routes ([9], pp. 121).

Channel loading and throughput:

- *channel load* ($\gamma_c(\pi, \Lambda)$) - The expected number of packets that cross channel c per cycle for the traffic matrix Λ.
- *pair channel load* ($\gamma_c(\pi)_{i,j}$) - The expected number of packets that cross channel c per cycle when routing algorithm π sends a packet from source i to destination j each cycle. If π is deterministic, $\gamma_c(\pi)_{i,j} \in \{0, 1\}$. Otherwise, when π is randomized, the pair channel load is the probability that a packet uses channel c during any particular cycle and $0 \leq \gamma_c(\pi)_{i,j} \leq 1$. [2]
- *maximum channel load* ($\gamma_{c,\max}(\pi)$) - The maximum load on channel c over all traffic matrices.

[1] We do not consider doubly-substochastic traffic matrices in this report because we are only concerned with worst-case traffic and any sub-stochastic matrix can be augmented with positive entries to create a stochastic matrix.

[2] It is assumed that the channel bandwidth equals the injection (ejection) bandwidth at each node. In general, the pair channel load is between 0 and the ratio of the injection (ejection) bandwidth to the channel bandwidth.

- *worst-case channel load* ($\gamma_{\text{wc}}(\pi)$) - The worst-case load on any channel over all traffic matrices:

$$\gamma_{\text{wc}}(\pi) = \max_{c \in C} \gamma_{\text{c,max}}(\pi).$$

- *worst-case ideal throughput* ($\Theta_{\text{ideal,wc}}(\pi)$) - The expected amount of bandwidth available to a packet crossing the worst-case channel:

$$\Theta_{\text{ideal,wc}}(\pi) = b/\gamma_{\text{wc}}(\pi).$$

  Since $\gamma_{\text{wc}}(\pi)$ packets are expected on the channel per cycle, the bandwidth of the channel b must be divided between them.

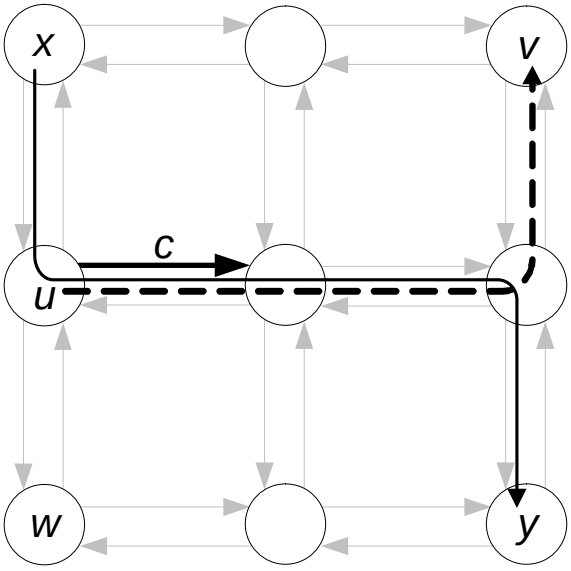

Figure 1: Two independent contributions to channel c's load

3 Finding the worst-case

Creating worst-case traffic patterns for oblivious routing algorithms is simplified by their *linearity of channel loading*. Linearity implies that the load on a particular channel is simply the sum of the loads caused by each source-destination pair. This fact can be used to constrain the search for worst-case patterns to permutation traffic. Then, by representing permutations with a bipartite graph and weighting the edges of the graph with source-destination channel loads, a maximum weight matching algorithm yields the exact worst-case permutation for a particular channel and its corresponding load in polynomial time. Finally, the maximum-weight matching is repeated over the set of all channels in the network to find the worst-case channel load and thus the worst-case ideal throughput.

3.1 Linearity of channel loading

The key to finding the worst-case of oblivious routing algorithms is to take advantage of their *linearity of channel loading*. That is, the load on a particular channel c is the sum of all the loads contributed by each source-destination pair in a traffic pattern:

$$\gamma_c(\pi, \Lambda) = \sum^{i,j} \lambda_{i,j}\gamma_c(\pi)_{i,j}.$$

An example of this property is shown in Figure 1 for an oblivious routing function π. One packet is being sent from node x to node y, crossing channel c. Another packet is sent from node u to v and also uses channel c. Both of these routes contribute a load of one packet per cycle across channel c or $\gamma_c(\pi)_{x,y} = \gamma_c(\pi)_{u,v} = 1$. Now consider a traffic matrix where $\lambda_{x,y} = \lambda_{u,v} = 1$. Then, for this example, the net load on channel c is $\lambda_{x,y}\gamma_c(\pi)_{x,y} + \lambda_{u,v}\gamma_c(\pi)_{u,v} = 2$ packets per cycle.

Although the total load on each channel is determined by a traffic matrix, the linearity property can be used to constrain the search for worst-case traffic patterns to permutation traffic only.

Theorem 1 *For any oblivious routing algorithm, a permutation matrix can always realize the ideal worst-case throughput.*

Proof Assume that the traffic matrix Λ gives a throughput lower than any permutation traffic pattern. By the result of Birkhoff [10], any doubly-stochastic traffic matrix Λ can be written as a weighted combination of permutation matrices:

$$\Lambda = \sum_{i=1}^{n} \phi_i P_i, \quad P_i \in \mathbb{P}.$$

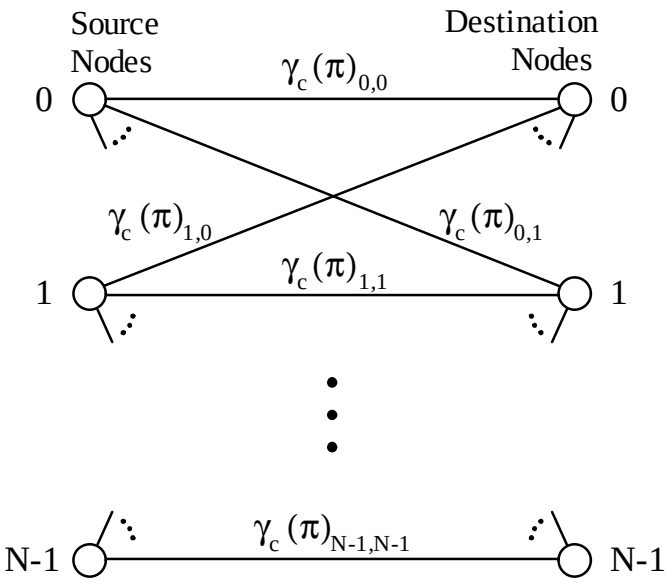

Figure 2: Construction of the bipartite graph

Given an oblivious routing algorithm π, the corresponding total channel load can be written using the independence property:

$$\gamma_c(\pi, \Lambda) = \sum_{i=1}^{n} \phi_i \gamma_c(\pi, P_i).$$

Considering the most heavily loaded channel c^* find the permutation P^* such that

$$P^* = \underset{P \in \{P_1, \ldots, P_n\}}{\operatorname{argmax}} \gamma_{c^*}(\pi, P).$$

Then $\gamma_{c^*}(\pi, P^*) \geq \gamma_{c^*}(\pi, P_i)$ for $i = 1, \ldots, n$ and substituting P^* as the traffic pattern gives a throughput less than or equal to Λ. This is a contradiction, and therefore a permutation matrix can always give the ideal worst-case throughput. ∎

Using this result, the worst-case channel load for a routing function π is

$$\gamma_{\text{wc}}(\pi) = \max_{c \in C} \left[\max_{P \in \mathbb{P}} \gamma_c(\pi, P) \right]$$

where $\mathbb{P}$ is the set of all permutation matrices.

3.2 Bipartite graph representation

Using the linearity of oblivious routing functions, a bipartite graph can be used to represent the load on a single channel due to any particular permutation. For our graph, the first set of N nodes are used to represent packet sources and the second set of N nodes represent the packet destinations. Edges are added between every source and destination node for a total of N^2 edges, as shown in Figure 2. The edge labels shown in the figure are explained in the following paragraphs. Also, note that this graph's structure is unrelated to the topology of the underlying interconnection network.

The bipartite graph gives a simple connection between permutation traffic patterns and a *matching* of the graph. A matching is a subset of the graph edges such that no node has more than one of its edges in the matching. In our original example, a packet is routed from node x to node y. This can be represented by adding the edge from source node x to destination node y to a matching (Figure 3). We can continue by adding the edge from node u to node v. However, the constraints of the matching do not allow an additional edge from node x to w, for example, and these are the

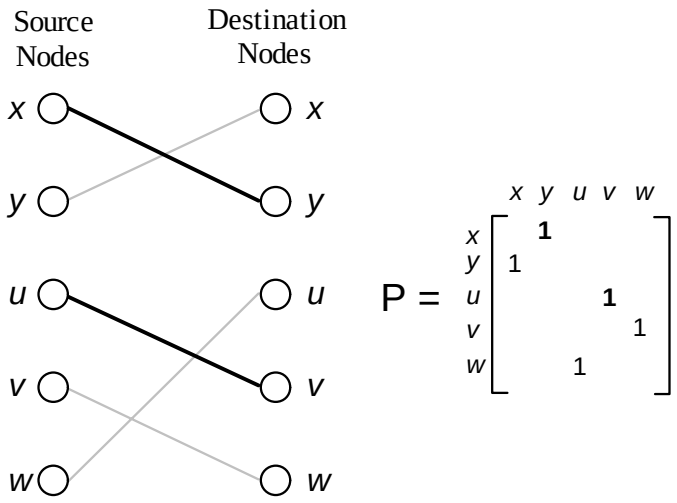

Figure 3: Example of a perfect matching in a bipartite graph and the corresponding permutation traffic pattern

exact constraints of the permutation — no node can be used as a source for more than one destination. This matching is illustrated in Figure 3, along with additional edges, shown in gray, which make the matching perfect. A *perfect matching* contains exactly one edge associated with every node in the graph and there is a one-to-one correspondence between permutation traffic patterns and perfect matchings (Figure 3).

We finish the graph's construction by weighting each edge from s to d with the amount of load contributed to a particular channel c when packets are routed from s to d, which is $\gamma_c(\pi)_{s,d}$ (Figure 2). Using these weights, the amount of load due to a specific permutation is just the sum of the edge weights in its corresponding bipartite matching. In the example, the edge from node x to y and the edge from u to v are labeled with one. When both are included in a matching they contribute a total load of two packets per cycle — the sum of their edge weights. The sum of the edge weights of any matching is called the weight of that matching.

The edge weights of the bipartite graph are calculated from the routing algorithm using either an exhaustive or statistical approach. For each source destination pair (s, d), the exhaustive approach enumerates all paths from s to d generated by the routing algorithm and sums the probability of each of these paths that includes the channel c to compute the edge weight $\gamma_c(\pi)_{s,d}$.

For some routing algorithms, the number of paths generated may be larger than polynomial. With such a large number of paths the edge weight can be accurately approximated by choosing a random sample of the paths generated by the routing algorithm from s to d and summing the contribution of the paths that include a particular channel to compute its edge weight. The contribution in this case is the probability of all of the paths represented by the sample.

3.3 Maximum-weight matching

Given the bipartite construction from the previous section, a *maximum-weight matching* of the graph is found. From the correspondence between matchings and permutations, finding a maximum-weight matching is equivalent to evaluating

$$\gamma_{\text{c,max}}(\pi) = \max_{P \in \mathbb{P}} \gamma_c(\pi, P).$$

By repeating this operation over all the channels, the worst-case channel loading can be determined:

$$\gamma_{\text{wc}}(\pi) = \max_{c \in C} \gamma_{\text{c,max}}(\pi).$$

An $O(N^3)$ maximum-weight matching algorithm exists [11], and therefore, finding the worst-case channel load requires $O(|C|N^3)$ time. For typical fixed-degree networks, such as tori or meshes, $|C|$ is proportional to N and the run time is $O(N^4)$. The maximum value of $|C|$ is N^2, corresponding to a fully-connected network, which bounds the time of the overall algorithm to $O(N^5)$. So, by exploiting the linearity of oblivious routing algorithms, the problem of examining all $N!$ permutations has been reduced to a polynomial-time algorithm.

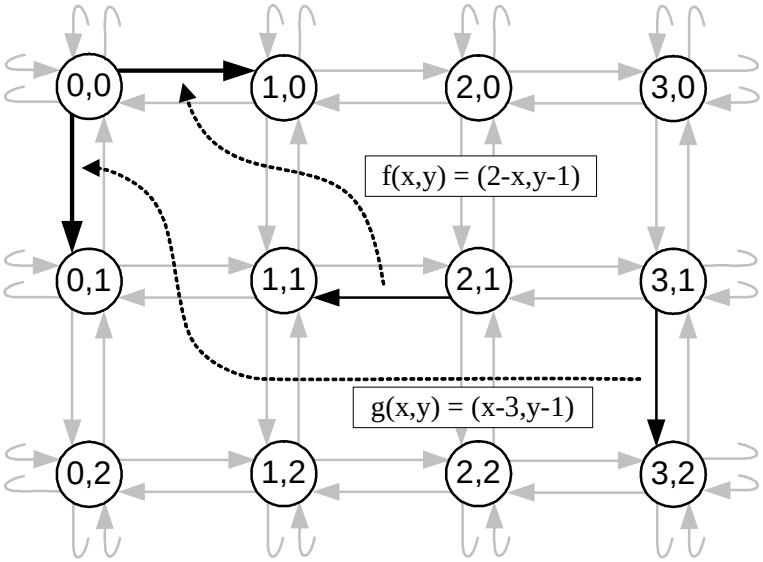

Figure 4: Example of automorphisms mapping channels to the representative set

4 Algorithm Optimizations

4.1 Symmetry

In Section 3, no assumptions were made about the underlying topology of the interconnection network. However, by exploiting the symmetry of a network, the number of channels examined to find the worst case can be greatly reduced. In fact, for a completely edge-symmetric topology and edge-symmetric routing algorithm, only a single channel needs to be considered.

To take advantage of symmetry, a set of *focus channels* F is formed so that for every channel c in the interconnection network, there exists an automorphism g that maps c into c' such that $c' \in F$. The automorphism must also maintain symmetry in the routing algorithm, so that $\gamma_c(\pi)_{i,j} = \gamma_{c'}(\pi)_{g(i),g(j)}$ for every source-destination pair (i, j).

For example, consider the 4,3-ary 2-cube shown in Figure 4, which is partially symmetric. The channel from node $(0, 0)$ to $(1, 0)$ and the channel from $(0, 0)$ to $(0, 1)$ form a focus set, assuming these channels also preserve symmetry in the routing algorithm. The figure also shows two automorphisms, f and g, that map particular channels to the focus set.

Now, instead of considering all of the channels for the worst-case load, only the channels in F are considered.

Theorem 2 *Given a topology, oblivious routing algorithm π, and their focus channel set F, at least one element of F can be loaded as heavily as any other channel in the network for a given traffic pattern.*

Proof Assume there is an channel c that is not in the focus set with a load greater than any element of the focus set. Let the permutation that realizes this load on c be P. By the definition of the focus set, there exists an automorhpism g that maps c to an element $f \in F$. The labeling function is then used to map every every pair of the permutation P into a new permutation P' which contains pairs $(g(i), g(j))$ over all pairs (i, j) in P. However, since g also preserves channel loading, the permutation P' gives the same load on f as P gave on c, which is a contradiction. Therefore, no channel can be loaded more heavily than the channels in the focus set. ∎

Using this result, the worst-case channel loading w can be expressed as

$$\gamma_{\mathrm{wc}}(\pi) = \max_{f \in F} \Big[\max_{P \in \mathbb{P}} \gamma_f(\pi, P) \Big].$$

This implies $|F|$ maximum-weight matchings are required to find the ideal worst-case throughput. Many common topologies, most notably the torus, are edge-symmetric. Completely edge-symmetric oblivous routing algorithms are less common, but routing algorithms can often be represented with a small focus set. For example, only two focus

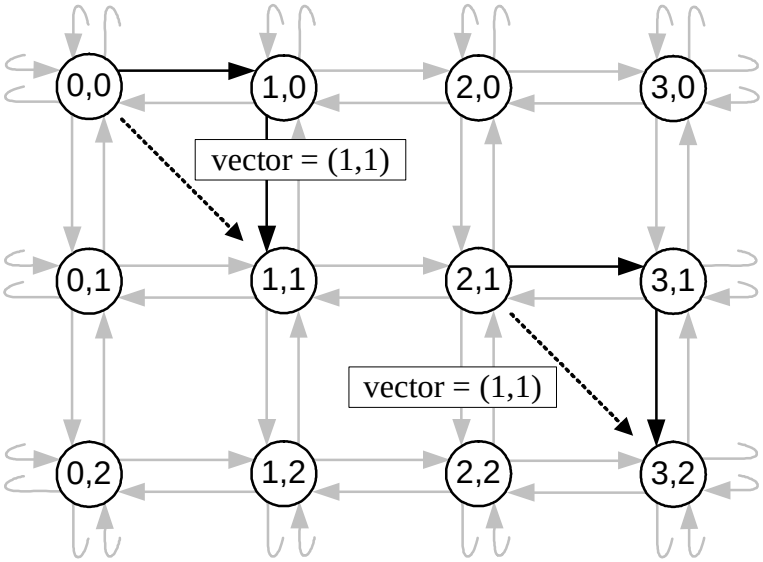

Figure 5: Example of relative routing

channels are needed to find the worst-case for dimension-order routing on the torus, which reduces the run time to $O(N^3)$.

4.2 Relative routing

For a designer to use maximum-weight matchings to determine the worst-case permutation the edge weights $\gamma_c(\pi)_{i,j}$ must be determined. For a general routing algorithm π, each source-destination pair must be considered to determine its contribution to the load on the focus channel(s). In an implementation, determining the edge weights often dominates the overall run-time for practical size networks.

However, if the topology is edge-symmetric, it is common for an oblivious routing algorithm to be *relative* or position-indepedent. That is, the input to the routing algorithm can be a "vector" that points from the source to destination. Then the paths a packet takes from the source to destination only depend on their relative placement in the network. For example, dimension-order routing in a torus is a relative routing algorithm. As shown in Figure 5, a route from $(0, 0)$ to $(1, 1)$ follows the same relative path as a route from $(2, 1)$ to $(3, 2)$. So, the dimension order routing algorithm only needs the vector $(1, 1)$ to determine the paths in this example.

A relative routing algorithm can be exploited to decrease the number of source-destination pairs considered to find all the required edge weights. If π is a relative routing algorithm, for a given source-destination pair (i, j) and a focus link from node u to node v,

$$\gamma_{(u,v)}(\pi)_{i,j} = \gamma_{(u+k,v+k)}(\pi)_{i+k,j+k},$$

where $k \in \{0, \ldots, N-1\}$. This relationship becomes useful in a practical situation when the designer does not have an explicit formula for $\gamma_c(\pi)$. In this case, finding the load on *all* channels in the network due to a particular source-destination pair does require an increase in storage proportional to the number of channels, but little additional work since complete paths for the routing algorithms are already being evaluated. So, by using the fact that the routing algorithm is relative, a single source-destination pair (i, j)'s loading of all channels can be used to determine N loadings of a focus channel for the source-destination pairs $(i + k, j + k)$, where $k \in \{0, \ldots, N-1\}$. This reduces the total number of pairs considered to N compared to N^2 for a non-relative routing algorithm.

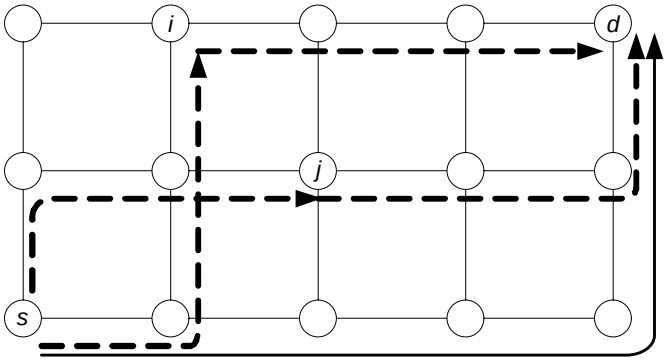

Figure 6: Example dimension-order (solid line) and ROMM routes (dashed lines)

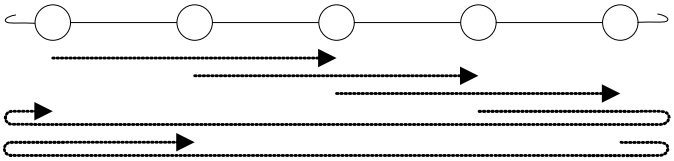

Figure 7: Tornado traffic pattern for $k = 5$

5 Experiments

As an illustration of the importance of finding exact worst-case permutations, a comparison of two minimal, oblivious routing algorithms is presented for a 2-dimensional torus network (k-ary 2-cube)[3]. The first algorithm is dimension-order routing (DOR). DOR deterministically routes a packet completely in the first dimension before routing in the second. An example dimension-order route from source s to destination d is shown as a solid line in Figure 6.

The second algorithm is the two-phase variant of the randomized algorithm (ROMM) described in [3]. ROMM routes a packet from source to destination by uniformly choosing a random intermediate node within the minimal quadrant. The minimal quadrant is the set of nodes along any minimal length path between the source and destination. The packet then uses DOR, but with a randomized order of dimension traversal, from the source to intermediate and repeats the same algorithm from the intermediate to the destination. Two example ROMM routes, which use intermediate nodes i and j respectively, are shown in Figure 6 as dashed lines.

Compared to DOR, where all traffic between a source-destination pair is concentrated along a single path, ROMM more evenly distributes a source-destination's traffic across a larger number of channels. From this qualitative description of the behavior of ROMM and based on the discussion presented in [3], one might expect that ROMM would have better worst-case performance than DOR.

To test this intuition, the performance of these two algorithms was compared against uniform random traffic and two permutations that are typically relied upon to demonstrate poor performance [3][4]: bit-complement and transpose. The tornado pattern was also considered, where each node sends packets $(k-1)/2$ hops to the right in the lowest dimension (Figure 7). In addition to these patterns, a trial of 10^4 random permutation traffic patterns was generated and the worst-case throughput for both algorithms over the 10^4 permutations was determined. As shown in Table 1, ROMM generally performed as well as DOR on these conventional metrics.

Next, the algorithm presented in Section 3 was used to determine the worst-case for both DOR and ROMM (Table 1). Edge weights were calculated using the exhaustive method described in Section 3.2. All calculations were performed using integer arithmetic, so no round-off error occurred and the worst-case results are exact. The worst-case of DOR matched the result of 0.278 of capacity found in the random permutations. However, ROMM's exact worst-case of 0.173 was significantly less - only 62.3% of DOR's worst-case throughput.

In ROMM, the tornado pattern in a single row gives the same loading as DOR. However, because ROMM routes through the minimal quadrant, and not just around the edges as DOR does, source-destination pairs in other rows can add additional load to channels in the tornado row, reducing the throughput of ROMM below that of DOR. An

[3]Only odd values of k are considered to simplify the explanation of the worst-case, but even values of k follow the same trends

Table 1: Ideal throughput of DOR and ROMM over several patterns on an 9-ary 2-cube (fraction of network capacity)

Pattern	DOR	ROMM
Uniform	1	1
Bit-complement	0.556	0.362
Transpose	0.278	0.556
Tornado	0.278	0.278
Worst of 10^4 permutations	0.278	0.255
Worst-case	0.278	0.173

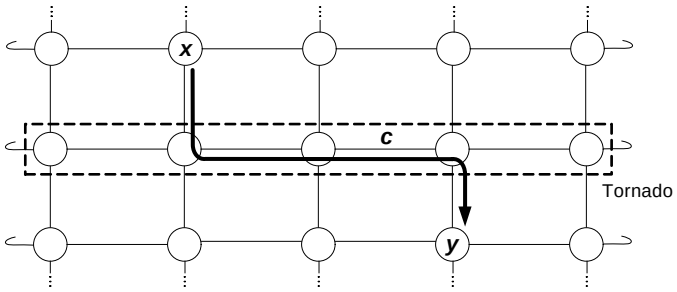

Figure 8: Adversarial pattern for ROMM

example of this is shown in Figure 8, where tornado traffic is set up for the nodes in one row. This loads channel c to the worst-case of DOR. The remaining rows do not participate in the tornado pattern, but are set up to send additional traffic across channel c. For example, the minimal quadrant between nodes x and y overlaps c (an example path is shown in bold). So, sending packets from x to y increases the load on c beyond the simple tornado case. The complete permutation found by the algorithm is shown in Figure 9.

A further comparison of the worst-cases of ROMM and DOR on k-ary 2-cubes showed that as k increases, DOR approaches approximately 0.26 of capacity, while ROMM approaches 0.14 or about half that of DOR. So, although ROMM might qualitatively seem to be a more "balanced" routing algorithm, these experiments show that simple DOR has superior worst-case performance on k-ary 2-cubes. This result was not immediately obvious from applying standard "difficult" traffic patterns or searching a large set of random permutations, showing the practical benefit of the maximum-weight matching approach.

$$\begin{bmatrix}
(4,0) & (4,6) & (4,7) & (4,8) & (8,8) & (6,7) & (4,1) & (4,2) & (4,3) \\
(0,0) & (6,0) & (6,6) & (7,1) & (6,5) & (8,2) & (1,5) & (0,4) & (5,4) \\
(6,8) & (7,0) & (5,6) & (8,1) & (5,5) & (0,3) & (5,3) & (1,4) & (6,4) \\
(5,8) & (8,0) & (0,6) & (0,2) & (5,2) & (4,5) & (6,3) & (2,4) & (7,4) \\
(7,8) & (0,1) & (5,1) & (8,5) & (6,2) & (3,5) & (7,3) & (3,4) & (8,4) \\
(5,0) & (7,6) & (6,1) & (7,5) & (7,2) & (2,5) & (8,3) & (4,4) & (0,5) \\
(1,0) & (1,6) & (1,7) & (1,8) & (0,8) & (5,7) & (1,1) & (1,2) & (1,3) \\
(2,0) & (2,6) & (2,7) & (2,8) & (8,7) & (0,7) & (2,1) & (2,2) & (2,3) \\
(3,0) & (3,6) & (3,7) & (3,8) & (7,7) & (8,6) & (3,1) & (3,2) & (3,3)
\end{bmatrix}$$

Figure 9: Worst-case permutation for ROMM on a 9-ary 2-cube. Entry (i, j) of the matrix denotes the destination node of the source on row i column j.

6 Conclusions

In this report, we presented an algorithm that can find the worst-case throughput of oblivious routing algorithms in $O(|C|N^3)$ time (bounded by $O(N^5)$), which makes worst-case analysis tractable. Additionally, a comparison of two minimal routing algorithms illustrated that intuition, difficult traffic patterns, and random sampling of permutations do not necessarily provide an accurate view of the worst-case performance of a particular routing algorithm. These traditional approaches poorly characterized the worst case of the ROMM algorithm [3], overestimating the throughput by approximately 47%.

We hope the techniques presented in this report will be a useful tool in the design and quantitative comparison of routing algorithms. Moreover, using the bipartite graph construction to analyze oblivious routing algorithms may prove to be a powerful technique for finding optimal worst-case routing algorithms.

References

[1] W. J. Dally, P. P. Carvey, and L. R. Dennison, "The Avici terabit switch/router," in *Conference Record of Hot Interconnects 6*, August 1998, pp. 41–50.

[2] InfiniBand Trade Association, "InfiniBand architecture specification," `http://www.infinibandta.org`.

[3] T. Nesson and S. L. Johnsson, "ROMM routing on mesh and torus networks," in *Proc. 7th Annual ACM Symposium on Parallel Algorithms and Architectures*, 1995, pp. 275–287.

[4] K. Bolding, M. Fulgham, and L. Snyder, "The case for chaotic adaptive routing," *IEEE Trans. on Computers*, vol. 46, no. 12, pp. 1281–1292, December 1997.

[5] A. Borodin and J. Hopcroft, "Routing, merging, and sorting on parallel models of computation," *Journal of Computer and System Sciences*, vol. 30, pp. 130–145, 1985.

[6] C. Kaklamanis, D. Krizanc, and A. Tsantilas, "Tight bounds for oblivious routing in the hypercube," in *Proc. 2nd Annual ACM Symposium on Parallel Algorithms and Architectures*, 1990, pp. 31–36.

[7] F. T. Leighton, B. M. Maggs, A. Ranade, and S. B. Rao, "Randomized routing and sorting on fixed connection networks," *Journal of Algorithms*, vol. 17, no. 1, pp. 157–205, July 1994.

[8] L. Peh and W. J. Dally, "A delay model and speculative architecture for pipelined routers," in *Proc. of the 7th Int. Symposium on High-Performance Computer Architecture*, January 2001, pp. 255–266.

[9] J. Duato, S. Yalamanchili, and L. Ni, *Interconnection Networks: an engineering approach*, IEEE Computer Society Press, 1997.

[10] G. Birkhoff, "Tres observaciones sobre el algebra lineal," *Univ. Nac. Tucumán Rev. Ser. A*, vol. 5, pp. 147–151, 1946.

[11] H. Kuhn, "The Hungarian method for the assignment problem," *Naval Res. Logist. Q.*, vol. 2, pp. 83–97, 1955.