Worst-case Traffic for Oblivious Routing Functions

Brian Towles and William J. Dally
Department of Electrical Engineering
Stanford University
{btowles,billd}@cva.stanford.edu

Abstract—**This paper presents 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 for routing functions with a polynomial number of paths. 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 throughput**

I. 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 routing 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 letter presents an efficient technique for finding an exact worst-case pattern for any oblivious routing function on an arbitrary network topology (Section III). By exploiting the linearity of oblivious routing functions, finding the worst-case traffic pattern is cast as the maximum-weight matching of a bipartite graph. Using this construction, the problem is generally solved in polynomial time, quickly yielding exact worst-case results. For cases where the number of paths is non-polynomial in the size of the network, the construction of the exact bipartite graph can also be non-polynomial. In this situation, the worst-case can still be estimated (Section III-B). The worst-case pattern is then used to determine the worst-case throughput of a particular system.

This approach offers a significant improvement in accuracy over existing techniques used to estimate the worst-case. 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 IV, 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]. Despite providing strong results, these analyses do not provide exact throughput values for specific topologies and routing algorithms. With the algorithms presented in this letter, we hope to enable more quantitative studies of oblivious routing algorithms in the future.

This work has been supported by an NSF Graduate Fellowship with supplement from Stanford University and under the MARCO Interconnect Focus Research Center. The authors would like to thank the anonymous reviewers for their helpful suggestions. Manuscript submitted 14 Dec. 2001. Manuscript accepted 1 Feb. 2001. Final manuscript received 8 Feb. 2001.

II. Preliminaries

A. Network model

The interconnection networks discussed in this letter have an arbitrary topology and fixed length data units. These units are referred to 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].

B. Definitions

- N - The number of nodes in the network.
- C - The set of all channels in the network.
- *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 exactly one.
- *permutation matrix* (P) - A traffic matrix whose entries are either 0 or 1.
- *oblivious routing algorithm* (π) - A routing algorithm that is only a function of the source and destination nodes of a packet. Oblivious routing algorithms can also be randomized ([9], pp. 121).
- *channel load* ($\gamma_c(\pi, \Lambda)$) - The expected number of packets that cross channel c per cycle for the traffic matrix Λ and routing function π.
- *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.
- *worst-case ideal throughput* ($\Theta_{\text{ideal,wc}}(\pi)$) - The expected amount of bandwidth available to a packet crossing the worst-

[1] Doubly-substochastic traffic matrices are not considered in this letter because we are only concerned with worst-case traffic and any substochastic 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.

case channel:

$$\Theta_{\text{ideal,wc}}(\pi) = \min_{c \in C} \left[b_c / \gamma_{\text{c,max}}(\pi) \right].$$

To prevent the worst-case channel from saturating, the injection bandwidth of each node sharing the channel is scaled so that the average number of requests per cycle is equal to the channel's bandwidth b_c.

III. Finding the worst-case

Before determining the worst-case throughput of a network, we describe how the throughput of a particular traffic pattern is found. Suppose the channel capacities are infinite. Each node sends one packet per cycle (a throughput of one) according to the traffic pattern and the average load on all the channels is calculated. However, each channel c can only sustain a throughput of b_c packets per cycle. If the requested load is greater than b_c the channel is saturated and actual throughput of the network must be scaled back. For a given traffic pattern, the maximum sustainable throughput is the smallest ratio of channel bandwidth to channel load over all channels. If, for example, every channel supported 1 packet per cycle, but the maximum channel load was 3 packets per cycle, the network could only sustain a throughput of 1/3. To ensure no channel is saturated, the network must be operated at this throughput.

Since the network channels are not saturated, their *linearity of channel loading* can be exploited. 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 matrices. Then, by representing all permutations as matchings within a single bipartite graph and weighting the edges of the graph with source-destination channel loads, a maximum-weight matching yields the exact worst-case permutation for a particular channel and its corresponding load. Finally, the maximum-weight matching is repeated over the set of all channels in the network to find the worst-case ideal throughput.

A. Linearity of channel loading

The key to finding the worst-case of oblivious routing algorithms is to take advantage of the algorithm's *linearity of channel loading*. That is, as long as the channels of the network are not saturated, 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.

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 matrices.

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

Proof: Assume that a traffic matrix Λ gives a throughput lower than any permutation matrix. This implies Λ loads at least one channel more heavily than any permutation. By the result of Birkhoff [10], any doubly-stochastic traffic matrix Λ can be written as a weighted combination of permutation matrices:

$$\Lambda = \sum_{i=1}^{n} \phi_i P_i, \quad \text{s.t.} \sum_{i=1}^{n} \phi_i = 1.$$

Without loss of generality, for any channel c, a permutation P^* is found such that

$$P^* = \underset{P \in \{P_1, \ldots, P_n\}}{\text{argmax}} \gamma_c(\pi, P).$$

Given an oblivious routing algorithm π, the corresponding total load on c can be written using linearity:

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

P^* loads any channel at least as heavily as Λ, but this is a contradiction. Therefore, a permutation matrix can always give the ideal worst-case throughput. □

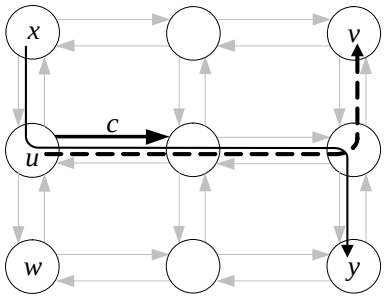

Fig. 1. An example of two independent contributions to channel c's load. One packet is being sent from node x to node y, crossing channel c. Another packet is sent from node u to node v and also uses channel c. Both of these routes contribute a load of one packet per cycle across channel c. Because none of the channels in the network are saturated, the net load on channel c is simply 2 packets per cycle.

B. 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. There is a one-to-one correspondence between permutation matrices and perfect matchings[3] of this bipartite graph. Also, note that this graph's structure is unrelated to the topology of the underlying interconnection network.

The graph's construction is finished by weighting each edge from source node s to destination node 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. This sum is called the *weight* of that matching.

[3] A perfect matching is a subset of the graph edges such that each node is incident with exactly one edge in the subset.

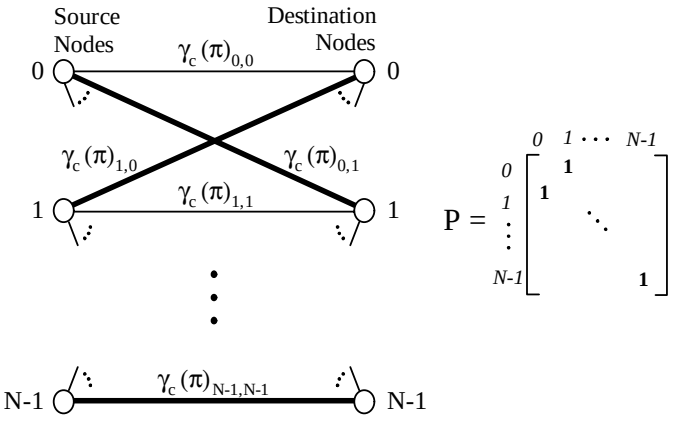

Fig. 2. Construction of the bipartite graph for finding channel load due to a particular permutation. A perfect matching (bold edges) and its corresponding permutation P are also shown. The rows (columns) of P correspond to the source (destination) nodes of the bipartite graph. As an example, the matching's edge from source node 0 to destination node 1 corresponds to the 1 in entry $(0, 1)$ of the permutation matrix.

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 from s to d generated by the routing algorithm and summing the contribution of the paths that include c to compute the weight. The contribution in this case is the probability of all of the paths represented by the sample.

C. 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

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

where $\mathbb{P}$ is the set of all permutation matrices. By repeating this operation over all the channels, the ideal worst-case throughput can be determined:

$$\Theta_{\text{ideal,wc}}(\pi) = \min_{c \in C} [b_c / \gamma_{\text{c,max}}(\pi)]$$

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 $O(N^5)$. If symmetries exist in the routing function and topology, then it is only necessary to examine a subset of the channels, as discussed in [12]. So, by exploiting the linearity of oblivious routing algorithms, the problem of examining all $N!$ permutations has been reduced to a polynomial-time algorithm.

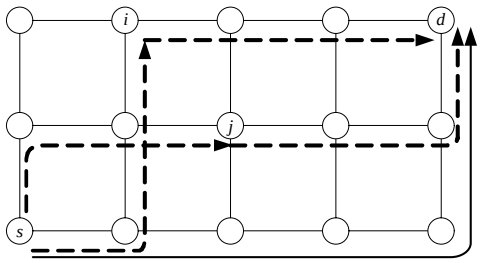

Fig. 3. Example dimension-order (solid line) and ROMM routes (dashed lines)

IV. 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)[4]. 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 3.

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 3 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 pair'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 4). In addition to these patterns, a trial of 10^4 random permutation matrices was generated and the worst-case throughput for both algorithms over the 10^4 permutations was determined. As shown in Table I, ROMM generally performed as well as DOR on these conventional metrics.

Next, the algorithm presented in Section III was used to determine the worst-case for both DOR and ROMM (Table I). Edge weights were calculated using the exhaustive method described

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

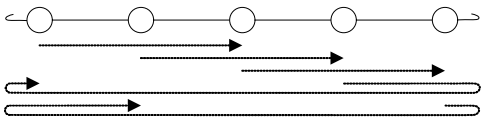

Fig. 4. Tornado traffic pattern for $k = 5$

TABLE I
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

in Section III-B. 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.

The reason for ROMM's lower worst-case throughput is illustrated by constructing an adversarial traffic pattern. 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 example of this is shown in Figure 5 and a worst-case permutation for ROMM is shown in Figure 6.

A further comparison of the worst-cases of ROMM and DOR on k-ary 2-cubes showed that as k increases beyond 9, 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 traffic patterns or

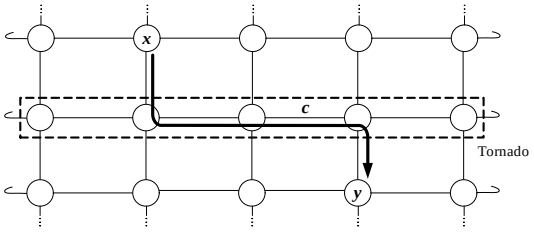

Fig. 5. Adversarial pattern for ROMM. Only the nodes in the middle row (dashed box) run the tornado traffic pattern, which loads c the same amount as in DOR's worst-case. However, because ROMM routes through the minimal quadrant, additional load can be placed on c by sending traffic from nodes outside the middle row. For example, the minimal quadrant of nodes x and y contains c, so sending traffic from x to y increases c's load beyond that of DOR.

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

Fig. 6. 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.

searching a large set of random permutations, showing the practical benefit of the maximum-weight matching approach.

V. CONCLUSIONS

In this letter, we presented an algorithm that can find the worst-case throughput of most oblivious routing algorithms in polynomial time, 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 letter 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.

[12] B. Towles, "Finding worst-case permutations for oblivious routing algorithms," CVA Technical Report 121, Stanford University, December 2001.