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 most practical routing functions. 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%.

Categories and Subject Descriptors

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

General Terms

Algorithms, Performance

Keywords

Oblivious routing, worst-case throughput, interconnection networks

1. INTRODUCTION

As interconnection networks are applied to throughput-sensitive applications, such as packet routing [7] and I/O interconnect [3], 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 paper 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 is cast as the maximum-weight matching of a bipartite graph. Using this construction, exact worst-case results can be found in polynomial time for all deterministic routing functions and for most randomized routing functions. For cases where the number of routing states 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. 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 [5, 12]. However, for the example presented in Section 5, the traditional techniques overestimate the worst-case throughput of the ROMM routing algorithm [12] by approximately 47%. Worst-case characterization has also been approached from a theoretical perspective [1, 6, 9, 11]. Despite providing strong results, these analysis do not provide exact throughput values for specific topologies and routing algorithms. With the algorithms presented in this paper, 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 paper have an arbitrary topology and are represented by a directed graph (N, C), where the set of vertices N and the set of edges C correspond to nodes and channels of the network, respectively.[1] Fixed length data units are assumed and are referred to as packets, but any fixed size network unit, such as flits or cells, is equivalent.

To simplify the analysis and isolate our results from any particular flow-control or packet scheduling scheme, the ideal throughput of our network is determined completely by edge congestion — the system is assumed to be stable, if for every channel, the average number of packets which need to cross that channel is less than the bandwidth of the channel. If the number of packets which need to cross a channel meets or exceeds a channel's bandwidth, this

*This work has been supported by an NSF Graduate Fellowship with supplement from Stanford University and under the MARCO Interconnect Focus Research Center. A short version of this paper appears in *Computer Architecture Letters*, vol. 1, Feb. 2002.

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

[1] For convenience, any set used in the scalar sense refers to the size of that set.

channel is *saturated*. This is obviously an upper bound on the performance of any practical network. As discussed in [2], this upper bound is achievable with output queuing in each node router, large queues, a simple scheduling protocol, and a burstiness constraint on the incoming traffic process. Practical systems can typically reach 60-75% of this bound [13].

Since the channels in interconnection networks are usually shared between many nodes, the worst-cast throughput is generally less than the full injection rate of the nodes. This leads to two different definitions of throughput, which we refer to as *balanced* and *unbalanced*. In both cases, we characterize the worst-case throughput as the maximum injection rate per traffic source that is guaranteed to be stable. The cases are distinguished by how the injected traffic is distributed across the destinations. For the balanced case, the ejection rate at each node is equal to the injection rate — the traffic is evenly distributed and both the rows and columns of the corresponding traffic matrix (Section 2.2) sum to the same rate. The unbalanced case loosens this restriction and the rate destined to each node is only bounded by the node's maximum ejection rate — rows of the traffic matrix sum to the injection rate, while column sums must only be less than one. While both definitions have utility, this paper only considers the balanced case. A typical application of the worst-case, balanced throughput of a system would be to characterize the amount of speedup a system requires (increase in channel bandwidth relative to the injection rate) to ensure a worst-case throughput equal to the maximum injection rate.

2.2 Definitions

- *traffic matrix* (Λ) - Any $N \times N$ doubly-substochastic[2] matrix where entry $\lambda_{i,j}$ represents the fraction of traffic traveling from source i to destination j.
- *permutation matrix* (P) - A traffic matrix with a single 1 entry in each row and column.
- *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 ([8], 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.
- *maximum channel load* ($\gamma_{c,\max}(\pi)$) - The maximum load on channel c over all doubly-stochastic traffic matrices.
- *worst-case ideal throughput* ($\Theta_{\text{ideal,wc}}(\pi)$) - The expected amount of bandwidth available to a packet crossing the worst-case channel. For unit injection bandwidth,

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

  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.

[2]A doubly-substochastic matrix has row and column sums of at most one, while a doubly-stochastic matrix has row and column sums of exactly one.

- *isomorphic graphs* - Two graphs G and H are isomorphic if there exists a labeling function such that a relabeling of the vertices 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 automorphism on G that maps u to v.

3. FINDING THE WORST-CASE

Since we are interested in measuring the stable throughput of a network, none of the channels can be saturated. This allows their *linearity of channel loading* to 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.

3.1 Linearity of channel loading

The key to finding the worst-case of oblivious routing functions is to take advantage of their *linearity of channel loading*. By making mild assumptions on the incoming traffic and routing processes, the net channel load can be written as a linear combination of individual source-destination contributions to this load.

First, the average load on a particular channel c can be expressed using a set of random processes: $A(t)_{i,j}$ is 1 if a packet is injected at node i destined for node j at time t, and $R_c(\pi, t)_{i,j}$ is 1 if the oblivious routing function π uses channel c as part of a route from node i to node j at time t. Both functions are zero otherwise. Let the system begin at an arbitrary time $t = 0$ and then the time average load on channel c is

$$\gamma_c(\pi, \Lambda) = \lim_{n \to \infty} \left[\frac{1}{n} \sum_{t=0}^{n-1} \sum_{i,j} A(t)_{i,j} R_c(\pi, t)_{i,j} \right]$$

as long as channel c is not saturated. If each of arrival and routing processes are then assumed to be stationary and ergodic and all of the arrival processes are independent of all of the routing processes, then

$$\begin{aligned} \gamma_c(\pi, \Lambda) &= E\left[\sum_{i,j} A(t)_{i,j} R_c(\pi, t)_{i,j} \right] \\ &= \sum_{i,j} E\left[A(t)_{i,j}\right] E\left[R_c(\pi, t)_{i,j}\right] \\ &= \sum_{i,j} \lambda_{i,j} \gamma_c(\pi)_{i,j}. \end{aligned}$$

An example of this property is shown in Figure 1.

3.2 Narrowing the search to permutation traffic

Using the linearity results of the previous section, the set of traffic patterns that must be examined is narrowed to permutation traffic.

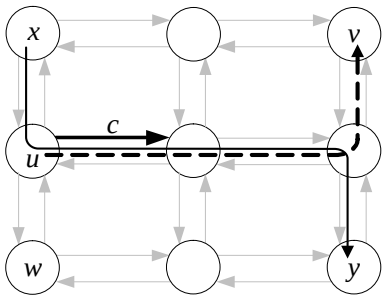

Figure 1: An example of two independent contributions to channel c's load. One packet is being sent from node x to node y, crossing channel c. Another packet is sent from node u to node v and also uses channel c. Both of these routes contribute a load of one packet per cycle across channel c. Because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

We are interested in finding a traffic matrix with the smallest row and column sums that loads any channel in the network to that channel's bandwidth. Focusing on a single channel c, assume that a doubly-stochastic traffic matrix Λ (row and column sums of exactly one) maximizes the channel load on c. Then, by definition

$$\gamma_c(\pi, \Lambda) = \gamma_{c,\max}(\pi).$$

The traffic matrix Λ may not correspond to a realizable channel load because the channel could have saturated at $b_c < \gamma_{c,\max}$. However, the traffic pattern can be "scaled back" to a feasible solution:

$$\gamma_c\left(\pi, \frac{b_c\Lambda}{\gamma_{c,\max}(\pi)}\right) = b_c.$$

THEOREM 1. *The scaling factor $b_c/\gamma_{c,max}(\pi)$ is the smallest fraction of the injection rate needed to saturate channel c.*

PROOF. Assume a scaling factor $\alpha < b_c/\gamma_{c,\max}(\pi)$ saturates channel c for some doubly-stochastic traffic matrix Λ:

$$\gamma_c(\pi, \alpha\Lambda) = b_c.$$

Applying linearity and substituting,

$$\alpha = \frac{b_c}{\gamma_c(\pi, \Lambda)} < \frac{b_c}{\gamma_{c,\max}(\pi)}.$$

Then $\gamma_{c,\max}(\pi) < \gamma_c(\pi, \Lambda)$, which is a contradiction. So, $b_c/\gamma_{c,\max}(\pi)$ is the smallest fraction of the injection rate needed to saturate c. □

This result allows the search for worst-case traffic patterns to be restricted to doubly-stochastic matrices, which can be strengthened further to include only permutation matrices.

THEOREM 2. *For any oblivious routing function π, a permutation matrix can always load a channel c as heavily as a doubly-stochastic traffic matrix Λ.*

PROOF. Assume that Λ gives a throughput lower than any permutation matrix. This implies Λ loads channel c more heavily than any permutation. By the result of Birkhoff [4], any doubly-stochastic traffic matrix Λ can be written as a weighted combination of permutation matrices:

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

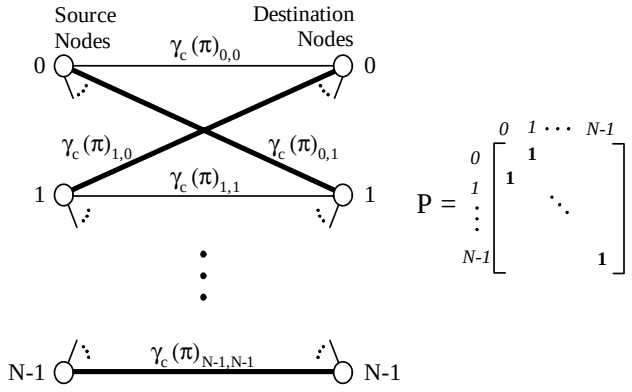

Figure 2: Construction of the bipartite graph for finding channel load due to a particular permutation. A perfect matching (bold edges) and its corresponding permutation P are also shown. The rows (columns) of P correspond to the source (destination) nodes of the bipartite graph. As an example, the matching's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

A permutation P^* is found such that

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

The corresponding total load on c can be written using linearity:

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

P^* loads channel c at least as heavily as Λ, but this is a contradiction. Therefore, a permutation matrix can always give the same load on c as a doubly-stochastic traffic matrix. □

Using these two theorems, worst-case traffic patterns can be found by first searching all permutation matrices while momentarily ignoring the feasibility of the solutions. Then, the permutation matrix that most heavily loads a channel is scaled to account for the actual channel bandwidth, and the scaling factor gives the smallest fraction of injection bandwidth needed to saturate that channel.

3.3 Bipartite graph representation

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). Techniques for finding the

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

edge weights are discussed in Section 3.5. 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.

3.4 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

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

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:

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

where C is the set of channels. An $O(N^3)$ maximum-weight matching algorithm exists [10], and therefore, finding the worst-case channel load requires $O(CN^3)$ time. For typical fixed-degree networks, such as tori or meshes, the size of C is proportional to N and the run time is $O(N^4)$. The maximum size 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 functions, the problem of examining all $N!$ permutations has been reduced to a polynomial-time algorithm.

3.5 Computing Edge Weights

A straightforward approach to computing the edge weights for the bipartite graph is to exhaustively examine all paths for each source-destination pair (s, d). This exhaustive approach enumerates each path from s to d generated by the routing function and sums the probability of each of these paths that includes the channel c to compute the edge weight $\gamma_c(\pi)_{s,d}$. While this approach yields exact edge weights, some practical, randomized routing functions generate a number of paths that is larger than polynomial.

The problem of a non-polynomial number of paths can be addressed by first expressing an oblivious routing function incrementally, where routes are constructed by evaluating the routing function R at the current location of the packet to determine the next hop for that packet:

$$R : S \times N \mapsto \mathcal{P}(S \times N).$$

N represents the current location of a packet and S is the set of *routing states*. A routing state is simply information associated with a packet so that its path may be constructed incrementally and, as shown, the routing state can be updated at each hop of a route. If the routing function is randomized, its output is a set of possible state-node pairs (denoted by the power set $\mathcal{P}$), one of which is randomly selected.

So, for example, consider a routing algorithm that randomly chooses a dimension traversal order. A particular traversal order for each packet can be chosen at the source node and stored in the routing state of that packet. By also storing the destination of the packet as part of its state, the entire path can be constructed incrementally. Viewed another way, the routing state of a packet differentiates it from other packets that could pass through a common intermediate node. Conversely, partial paths that intersect at the same node with the same routing state can continue as a single path with a probability equal to the sum of the combined paths.

Each routing state-node pair is represented as a vertex labeled with that pair in a *routing state-node transition graph* T. Directed edges in the graph indicate a non-zero transition probability from one state-node pair to another. For each edge from vertex v_1 to vertex v_2, the function $E(v_1, v_2)$ is the probability of a transition from v_1 to v_2. The graph T and the probability function E are constructed by visiting each state-node pair (t, n) and evaluating $R(t, n)$. For each element $(u, m) \in R(t, n)$, an edge is added to T from vertex (t, n) to vertex (u, m) and given probability $E((t, n), (u, m)) = 1/|R(t, n)|$.

An example construction is shown for an oblivious routing function 5TURNS for the two-dimensional mesh. The 5TURNS function allows any minimal path from a source to destination that contains less than five turns from one dimension to the other with packets always starting in the x dimension. A portion of the incremental routing function is

$$R(t, x, y) = \begin{cases} \{(0, x+1, y), & \text{if } t = 0,\ d_x > x, \\ \ (1, x, y+1)\} & \text{and } d_y > y, \\ & \\ \{(1, x, y+1), & \text{if } t = 1,\ d_x > x, \\ \ (2, x+1, y)\} & \text{and } d_y > y, \\ & \cdots \end{cases}$$

where the routing state t is the number of turns, (x, y) is the packet's current location, and (d_x, d_y) is the destination node. Since we only consider one source-destination pair in isolation when finding edge weights, the destination is constant and does not need to be included as part of the packet's state. Several example paths generated by 5TURNS for a particular source-destination pair are shown in Figure 3a with the current number of turns (routing state) labeled along each path. Part of the corresponding graph T is shown in Figure 3b.

Once T has been constructed, the probability of a given path visiting each node is calculated. For a particular state-node pair (t, n), the probability of visiting that pair $V(t, n)$ is the sum of the probabilities of transitioning from each predecessor times that predecessor's probability, or

$$V(t, n) = \sum_{u \in S} \sum_{m \in N} V(u, m) E((u, m), (t, n)).$$

This recursive formulation is initialized using the source node s and the initial state t_0 and setting $V(s, t_0) = 1$. Then, there will always exist a vertex in the graph whose predecessors' probabilities have already been computed if T is acyclic. This is guaranteed if the underlying routing function is acyclic (a path cannot revisit a node). Given the vertex probabilities, the load on a channel $c = (x, y)$ is the sum of visiting that channel over all possible routing states

$$\gamma_{(x,y)}(\pi)_{s,d} = \sum_{u \in S} \sum_{t \in S} V(t, x) E((t, x), (u, y)).$$

Constructing T requires visiting each of the NS state-node pairs and computing the output set of the routing function. The output set has at most NS elements[4], so the overall time for constructing T is $O(N^2 S^2)$. Then T is traversed to calculate the actual channel loads, visiting each edge once to compute V and again to compute the channel loads, which again requires $O(N^2 S^2)$ time. So, for an acyclic, oblivious routing function with a polynomial number of routing states in N, the channel loads, and therefore edge weights, can be computed in polynomial time.

[4]For simplicity, we assume each element of a routing function's output set is computed in constant time. This computation time could be a function of S and N, but would still be polynomial for practical systems.

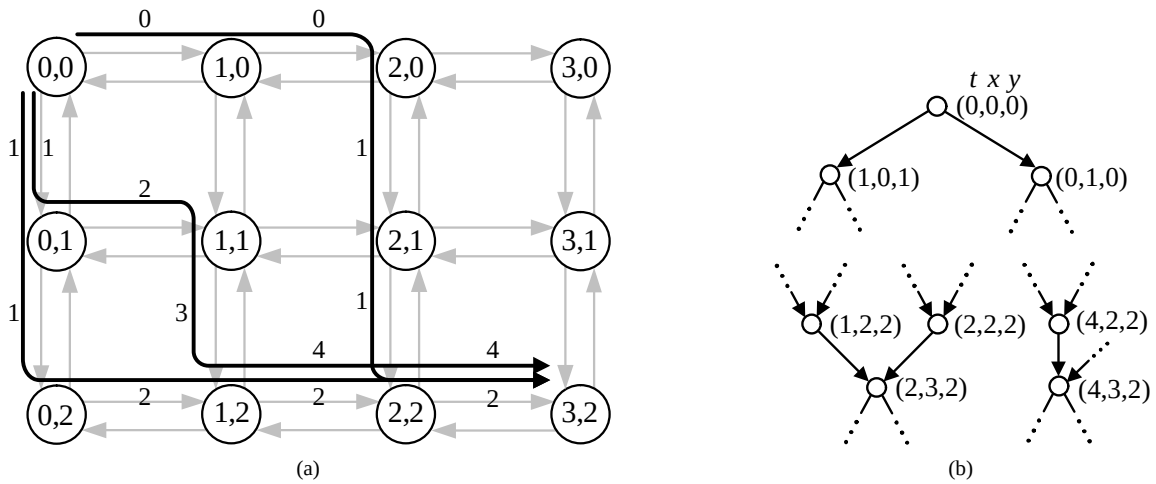

Figure 3: An example of path combining for the 5TURNS routing function. (a) Three possible paths from (0,0) to (3,2) are shown and are labeled with their current state (number of turns). (b) The corresponding routing state-node transition graph shows two of the paths being combined at a single state-node vertex, while the third path is not combined because it has taken more turns.

While it is possible to have a non-polynomial number of states, using the routing-state node transition graph to compute edge weights does have advantages over evaluating all paths. For example, the routing function that randomly selects between all minimal paths has a number of paths that is greater than polynomial in N. However, this routing function can be expressed incrementally where the only state necessary is the destination node. This allows the edge weights to be computed in polynomial time.

For routing functions with a very large number of paths and routing states, an edge weight can still be accurately approximated by choosing a random sample of the paths from s to d generated by the routing function 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.

4. SYMMETRY OPTIMIZATIONS

While the algorithms presented in Section 3 generally run in polynomial time, the large powers of N can still restrict the practical size of networks that can be analyzed. In this section, we present a framework to reduce the number of channels examined for the worst-case for networks with either full or partial symmetry. Also, for a class of routing functions in fully symmetric networks, a reduction in the time required to the find edge weights of the bipartite graph is shown.

4.1 Symmetry

Previously, 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 function, 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 f such that $f \in F$. The automorphism must also maintain symmetry in the channel bandwidth and the routing function, so that $b_c = b_f$ and $\gamma_c(\pi)_{i,j} = \gamma_f(\pi)_{g(i),g(j)}$ 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 symmetry in the channel bandwidth and routing function is also preserved. The figure also shows two automorphisms, g and h, that map particular channels to the focus set.

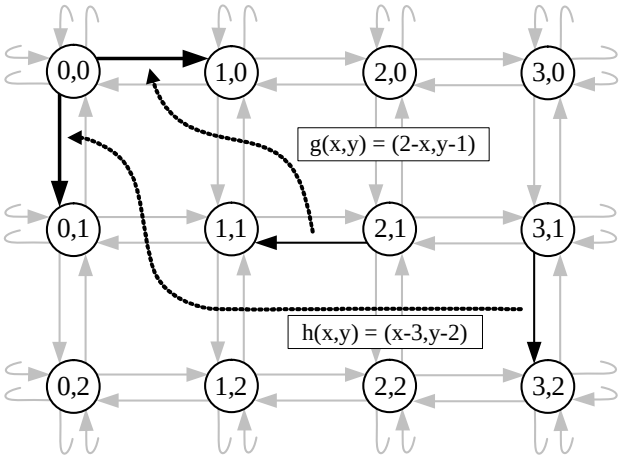

Figure 4: A 4,3-ary 2-cube where the channels from (0,0) to (0,1) and from (0,0) to (1,0), which are shown in bold, form the focus set. Two automorphisms, g and h, that map other channels into the focus set are also shown.

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

THEOREM 3. *Given a topology, oblivious routing function π, and their focus channel set F, at least one channel in F saturates at a throughput at or below any channel not in F.*

PROOF. Assume there is a channel $c \notin F$, which saturates at a rate lower than any channel in F. By definition of the focus set, there exists an automorphism g that maps c to an element $f \in F$ where $b_c = b_f$. So, for c to saturate at a lower throughput than f, $\gamma_{c,\max}(\pi) > \gamma_{f,\max}(\pi)$. Let P be a permutation such that $\gamma_c(\pi, P) = \gamma_{c,\max}(\pi)$. Construct a new permutation P' where

$P'_{g(i),g(j)} = P_{i,j}$. Then,

$$\gamma_f(\pi, P') = \sum_{i,j} p'_{i,j}\gamma_f(\pi)_{i,j} = \sum_{i,j} p'_{g(i),g(j)}\gamma_f(\pi)_{g(i),g(j)}$$
$$= \sum_{i,j} p_{i,j}\gamma_c(\pi)_{i,j} = \gamma_{\text{c,max}}(\pi).$$

This implies that $\gamma_{\text{f,max}}(\pi) \geq \gamma_{\text{c,max}}(\pi)$, which is a contradiction. Therefore, a focus channel can always be saturated at a throughput at or below a non-focus channel. □

Using this result, the worst-case throughput can be expressed as

$$\Theta_{\text{ideal,wc}}(\pi) = \min_{f \in F} \left[b_f / \gamma_{\text{f,max}}(\pi) \right].$$

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 oblivious routing functions are less common, but routing functions can often be represented with a small focus set. For example, only two focus channels are needed to find the worst-case for dimension-order routing on the two-dimensional 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 function 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 function to be *relative* or position-independent. That is, the input to the routing function 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 function. As shown in Figure 5, a route from $(0,0)$ to $(1,1)$ follows the same relative path as a route from $(1,2)$ to $(2,0)$. So, the dimension order routing function only needs the vector $(1,1)$ to determine the paths in this example.

A relative routing function 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 function, 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 $k \in \{0, \ldots, N-1\}$. 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 functions are already being evaluated. So, by using the fact that the routing function 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 function.

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)[5]. 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.

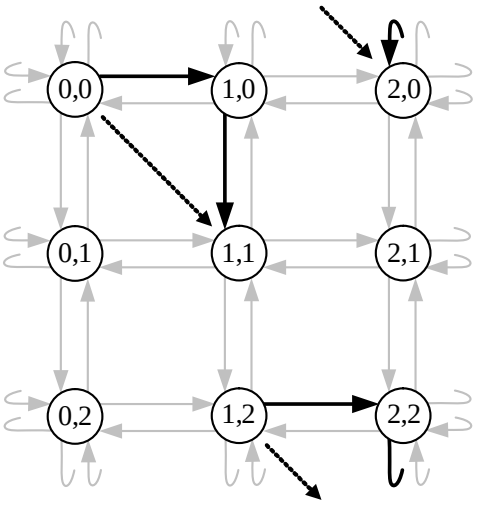

Figure 5: Relative routing example in a symmetric 3-ary 2-cube. Both paths (shown in bold) move in the same pattern — one hop to the right, then one hop down.

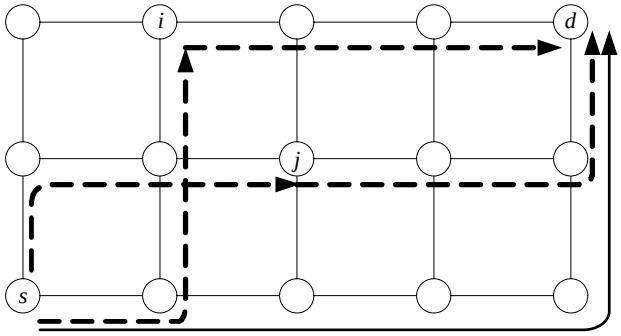

Figure 6: Example dimension-order (solid line) and ROMM routes (dashed lines)

The second algorithm is the two-phase variant of the randomized algorithm (ROMM) described in [12]. 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 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 [12], one might

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

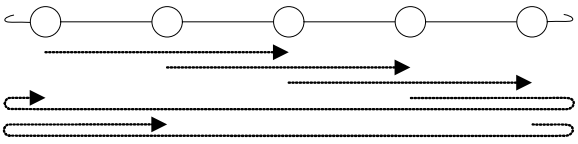

Figure 7: Tornado traffic pattern for $k = 5$

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

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 [5, 12]: 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 matrices 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). Both algorithms are relative routing functions and benefit from symmetry optimizations: only one channel in the X and Y dimensions needed to be considered. This reduced the the time required to compute the worst-case to $O(N^3)$. All calculations were performed using integer arithmetic and the worst-case results are exact. Values shown in the table have be rounded to three significant digits. 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 8 and a worst-case permutation for ROMM 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 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 searching a large

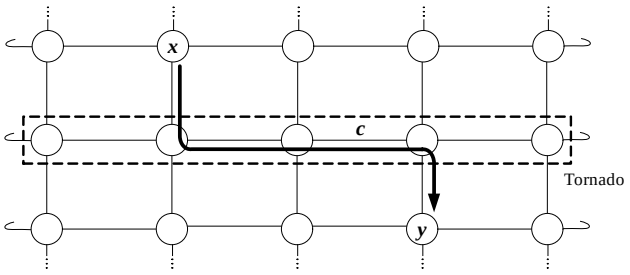

Figure 8: Adversarial pattern for ROMM. Only the nodes in the middle row (dashed box) run the tornado traffic pattern, which loads c the same amount as in DOR's worst-case. However, because ROMM routes through the minimal quadrant, additional load can be placed on c by sending traffic from nodes outside 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.

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

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.

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

6. CONCLUSIONS

In this paper, 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 [12], overestimating the throughput by approximately 47%.

An interesting open question is whether similar worst-case results can be determined for adaptive routing algorithms. While the class of all adaptive algorithms may not be amenable to such a analysis, we conjecture that particular adaptive algorithms can be designed that do have quantifiable worst-case throughput and are provability better than oblivious algorithms.

Finally, we hope the techniques presented in this paper 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.

7. REFERENCES

[1] W. A. Aiello, F. T. Leighton, B. M. Maggs, and M. Newman. Fast algorithms for bit-serial routing on a hypercube. *Mathematical Systems Theory*, 24(4):253–271, 1991.

[2] M. Andrews, B. Awerbuch, A. Fernández, T. Leighton, Z. Liu, and J. Kleinberg. Universal-stability results and performance bounds for greedy contention-resolution protocols. *Journal of the ACM*, 48(1):39–69, January 2001.

[3] I. T. Association. InfiniBand architecture specification. `http://www.infinibandta.org`.

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

[5] K. Bolding, M. Fulgham, and L. Snyder. The case for chaotic adaptive routing. *IEEE Trans. on Computers*, 46(12):1281–1292, December 1997.

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

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

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

[9] 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*, pages 31–36, 1990.

[10] H. Kuhn. The Hungarian method for the assignment problem. *Naval Res. Logist. Q.*, 2:83–97, 1955.

[11] F. T. Leighton, B. M. Maggs, A. Ranade, and S. B. Rao. Randomized routing and sorting on fixed connection networks. *Journal of Algorithms*, 17(1):157–205, July 1994.

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

[13] 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*, pages 255–266, January 2001.