Butterfly and Torus Networks

Lecture #3: Wednesday, April 11th 2001
Lecturer: Professor Bill Dally
Scribe: Jason Kassoff, Paul Wang Lee
Reviewer: Kelly Shaw

1 Announcements

- See Kelly to sign up for scribing assignments right away. Because of high enrollment, groups of $2 \sim 3$ people must sign up for each lecture. Please follow scribing deadlines, as others use these notes to study for exams.
- Also see Kelly to get photo taken before Monday. Class participation points cannot be given until your picture is taken.
- Homework 1 is due next Wednesday. The research paper and project assignments will be posted later this week.

2 Basic Topology Review

In the previous lecture, we identified three figures of merit for network topologies. They were:

- Throughput
- Latency
- Path Diversity

The first two metrics, throughput and latency, have equations to describe them. Remembering these two equations is essential to understanding network topology.

Throughput Throughput is defined as the ratio of the channel capacity, given the channel bandwidth, to the channel load. The channel load is defined as the sum over all source nodes x and destination nodes y of the fraction of traffic that goes from x to y across that channel, divided by the number of paths from x to y:

$$\gamma_c = \sum_x \sum_y \lambda_{xy} \sum_{P_{xy}} \frac{1}{|R_{xy}|} \quad if\ c\ \epsilon\ P, \quad 0\ otherwise$$

Note that the traffic pattern does not have to be uniform. λ_{xy} can vary to indicate higher or lower probability of traffic between a given pair of nodes. Also, keep in mind that this equation only considers the effects of topology; several other factors can affect channel load, such as routing.

We already saw one example of applying this formula to butterfly networks on Monday. Later today, we'll see another example with a torus topology.

Latency Three things contribute to network latency, the measure of the delay or time that a message takes to traverse a network. They are:

- The number of hops taken multiplied by the time delay of each router

$$T_h = H \times t_r$$

- The time of flight, distance traveled divided by velocity.

$$T_w = \frac{D}{v}$$

- The serialization latency, message length divided by channel bandwidth

$$T_s = \frac{L}{b}$$

The total latency is then:

$$T = H \times t_r + \frac{D}{v} + \frac{L}{b}$$

The behavior of networks can vary widely with respect to these three terms. The sum of the first two terms is called head latency. In some networks, the head latency is dominated by long router delays, with small time of flight; in others the time of flight dominates, while router delays are small. In networks with non-uniform bandwidth, the serialization latency is calculated from the smallest, or bottleneck, bandwidth.

Path Diversity Unlike throughput and latency, there is no general formula for computing path diversity. It is different for each topology.

3 Continuation of Butterfly Networks

Today we will finish the chapter on butterfly networks by examining path diversity in butterflies.

Given a source node x and destination node y in a butterfly network, the path diversity is written as

$$|R_{xy}| = 1$$

Intuitively, this makes sense, since we saw in the last lecture that the destination address completely specifies the route, independent of the source address.

A path diversity of 1 is undesirable for two reasons. First, any node or channel that fails can break the network (i.e. there is no fault tolerance). Second, a nework with low path diversity is vulnerable to bottlenecks. Whether or not a bottleneck exists depends on the traffic pattern applied to the network, and in evaluating a topology we consider the worst patterns that an adversary could apply.

To visualize the bottleneck problem with a butterfly network, consider a 2-ary n-fly mapped into two dimensions as shown in the figure below. The set of vertical planes resolves the first half (MSBs: $n-1$ to $n/2$) of the destination address, while the horizontal planes resolve the second half (LSBs $n/2-1$ to 0) of the address.

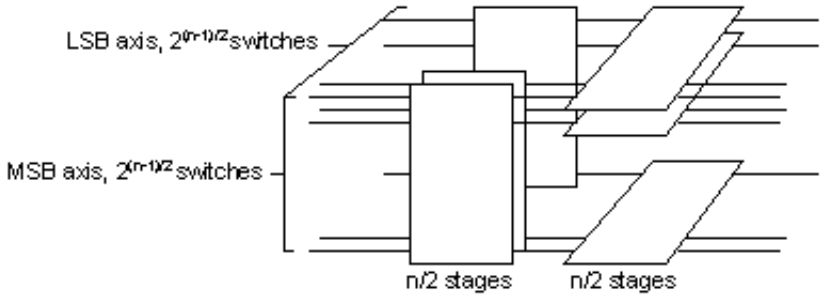

Figure 1: 2-ary n-fly mapped into two dimensions

Now consider permutation traffic that pairs the inputs to each vertical plane with the outputs of a single horizontal plane. That is, map the front vertical plane to the top horizontal plane, the second vertical plane back to the next horizontal plane down, etc. This pattern forces all traffic from a given vertical plane to cross on a single channel to its corresponding horizontal plane. In this example, all 2^n messages are carried on $2^{\frac{n-1}{2}}$ channels at that crossing.

In general, this kind of congestion can occur in any network where the number of paths is small compared to the number of source-destination pairs, because paths must be shared. An adversary could pick a channel, find which pairs of nodes use that channel, and send traffic between those pairs first. To beat an adversary at this game, the network must have enough paths (N^2 for a network with N nodes) to prevent any from being overloaded.

How can a we improve the path diversity of a butterfly network? Consider the example of a 2-ary 3-fly. If we duplicate the output stage in front of the input stage, we end up with the network shown in the figure below.

The darkened channels show that there are now 2 paths between nodes 0 and 7, and in fact the path diversity of the network has increased to 2. To choose between the two paths, we could use adaptive routing (route messages along the path that has lower traffic) or random routing (flip a coin). In either case, the path chosen no longer depends

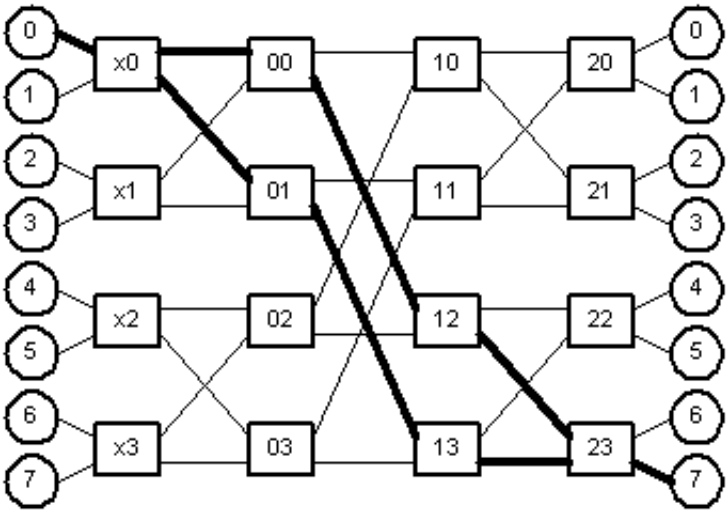

Figure 2: extended 2-ary 3-fly

solely on the destination address. Adding the extra stage effectively reshapes the traffic before it enters the original butterfly, applying a new permutation matrix to a portion of the traffic, with adjacent inputs swapped from the original permutation.

In the above example,the path diversity increased from 1 to 2 when we added an extra stage. To generalize for an n-stage butterfly, consider again dividing the network stages into horizontal and vertical planes. Instead of duplicating only the last stage of the network, we duplicate the last $n/2$ stages and place them in front of the original input stages as shown in the figure below.

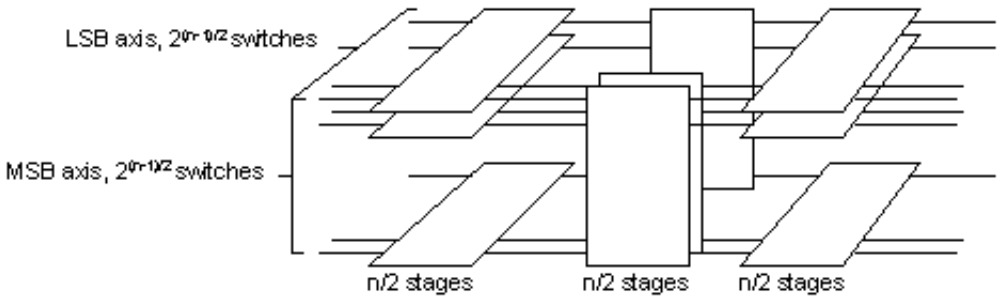

Figure 3: 2-ary n-fly with last half of network duplicated

Now the incoming traffic can be spread equally between all $2^{n/2}$ vertical planes, the original input stages. Thus, the path diversity is increased to $2^{n/2}$. Have we solved the congestion problem completely? That is, is the above network non-blocking? Not quite.

An adversary could still induce a non-uniform load in the vertical planes. For example,

this could be done by applying a shuffle permutation on the $n/2$ MSBs of the node addresses. The additional input stage spreads traffic in the z direction (into the page), but on average, each subnetwork represented by a vertical plane would still have to route a shuffle permutation. This can be fixed by duplicating the vertical planes as well, which gives us a complete duplication of the original butterfly network, or a Beneš(pronounced Ben-ish) network. The first butterfly effectively spreads the input traffic across all N nodes, in y and z, before it reaches the inputs of the second butterfly. The resulting path diversity is N, and we have a non-blocking network.

4 Torus Networks

A torus network can be described as a k-ary n-cube, with k nodes in each dimension and n dimensions. The figure below shows a 4-ary 1-cube, also known as a ring, and a 4-ary 2-cube.

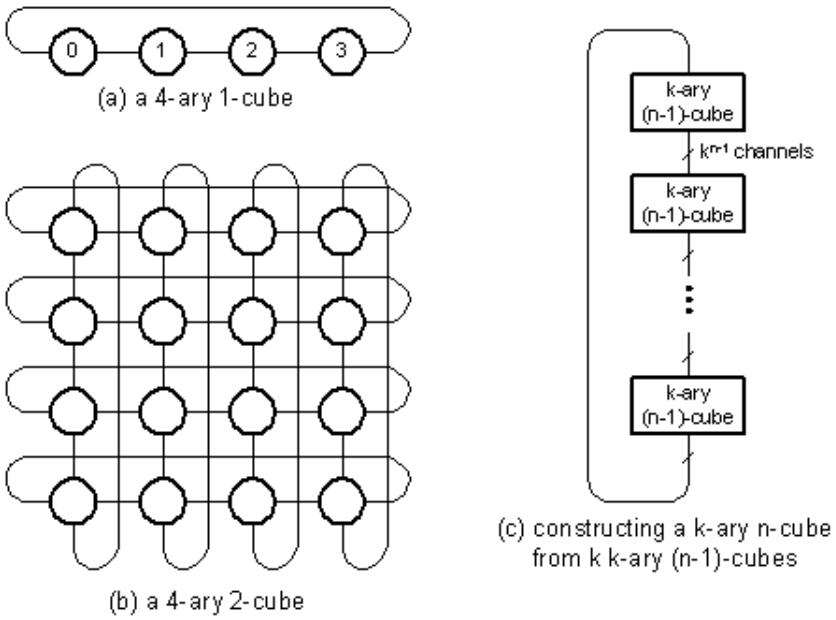

Figure 4: examples of 4-ary n-cubes

Notice that the 4-ary 2-cube is just a "ring" of 4 4-ary 1-cubes. In general, a k-ary n-cube can be built up by joining the nodes of k k-ary $(n-1)$-cubes with k^{n-1} channels. The 2-cube torus is topologically equivalent to a donut shape; this can be seen by mapping the x dimension through the hole of the donut and the y dimension around the circumference of the donut.

Although meshes also consist of k nodes in each of n dimensions, they have different properties from tori. A mesh does not have wraparound links, so it is not symmetric.

That is, its nodes are not interchangeable and a node does not have the same number of neighbors in every direction. For example, the edge nodes of a 2-mesh have only 3 links, and the corner nodes have only 2 links, while all of the nodes of a 2-cube have 4 links. This lack of symmetry, apparent in the figure below, results in a load imbalance and inefficient pin utilization when the mesh is hit with random traffic. It experiences a much higher concentration of traffic on the central nodes, since they must carry traffic not only between other central nodes but also between edge nodes. Despite this shortcoming, meshes are sometimes used because the wraparound links for tori are hard to include in some packaging schemes. (However, this can be fixed by folding the torus, as will be seen later.)

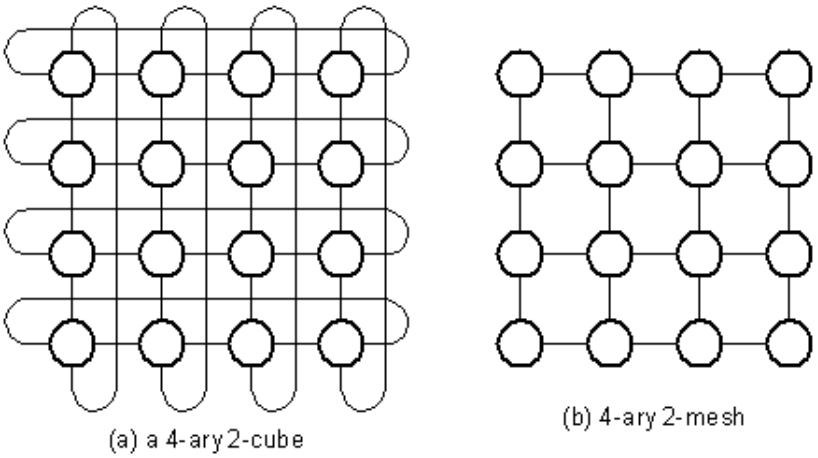

Figure 5: mesh vs. torus topology

A final note about both torus and mesh networks is that they can have different sizes in each dimension. For example, the figure below shows a mixed radix (2,3,4-ary) 3-mesh.

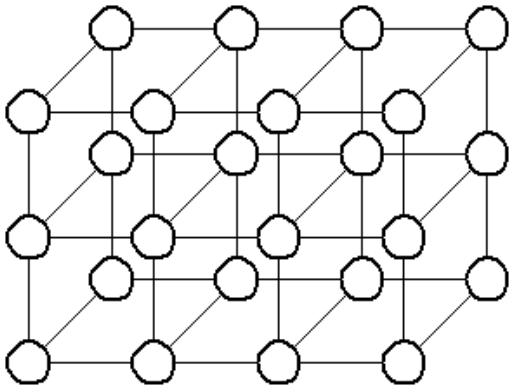

Figure 6: mixed radix mesh topology

Fun things about torus networks:

- Physical locality: if two things are close together in a network, traffic between them traverses fewer channels. Compare to the butterfly, where all paths are $\log_k N$ in length.
- Path diversity: there are many paths between each pair of nodes, especially if non-minimal routes are considered.
- Direct network: each node contains both a terminal and a switch.
- Bidirectional channels: can be a big win with bidirectional signalling.
- Uniformly short channels: with low n, the time of flight delay is low. This makes for easier packaging and is key to high bandwidth with electrical signalling, since frequency times wire length squared is constant above a critical length.
- Tunable performance: varying k and n can vary the suitability for different packaging options. Compare to the butterfly, where typically k is simply the maximum radix that can fit on a single chip.

4.1 Throughput

In the following analysis, we consider only the effects of topology on throughput, assuming frequency is independent of our design choices.

Bandwidth Two factors limit the bandwidth of a torus network. First, the channel width is limited by the number of pins on a node divided by the degree of the node. The figure below shows that the degree for each node is $4n$.

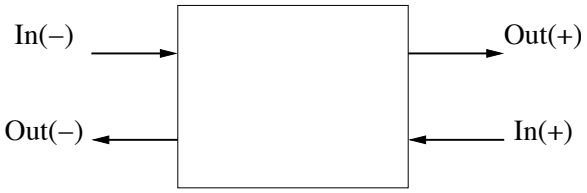

Figure 7: channels in a dimension for a torus network

The number 4 comes from one positive output channel, one positive input channel, one negative output channel, and one negative input channel. These 4 channels are needed in each of n dimensions, for a total of:

$$\delta = 4n$$

This gives the following limitation on channel width:

$$w \leq \frac{W_n}{\delta} = \frac{W_n}{4n}$$

The second limit on channel width comes from the number of wires in the bisection divided by the number of channels in the bisection. The number of channels in the bisection can be computed for a k-ary n-cube by remembering that it is a ring of k-ary (n-1)-cubes with k^{n-1} channels in each direction between each pair of cubes. The bisection cuts the ring in two places, crossing $2k^{n-1}$ channels at each cut, for a total channel bisection of:

$$B_c = 4k^{n-1} = \frac{4k^n}{k} = \frac{4N}{k}$$

This gives another limitation on channel width:

$$w \leq \frac{W_s}{B_c} = \frac{W_s k}{4N}$$

Combining the node limit and bisection limit gives:

$$w \leq min\left(\frac{W_n}{4n}, \frac{W_s k}{4N}\right)$$

If we assume that we want to maximize bandwidth given physical constraints on W_n and W_s, then it appears we just want to increase k (and decrease n). This implies that we would only want to build ring networks, because then all of the node and bisection bandwidth would be dedicated to just four channels. The missing constraint is the channel load, which increases with increasing k and causes throughput to fall.

Channel Load What portion of traffic crosses the bisections channels? On average, 1/4 stays on the left side of the network, 1/4 stays on the right, 1/4 goes from left to right, and 1/4 goes from right to left. Therefore, 1/2 of the total traffic crosses the bisection channels. The amount of traffic carried by bisection channels is then $N/2$; and the number of channels in the bisection, figured above, is $4N/k$. Thus the channel load, the traffic carried on each channel, is $N/2$ divided by $4N/k$:

$$\gamma_c = \frac{N/2}{4N/k} = \frac{k}{8}$$

Now that we have the channel width and channel load for a torus network, the throughput can be computed from the frequency times the channel width divided by the load:

$$\Theta = \frac{fw}{\gamma} = \frac{8}{k} \cdot f \cdot min\left(\frac{W_n}{4n}, \frac{W_s k}{4N}\right) = min\left(\frac{2B_n}{nk}, \frac{2B_s}{N}\right)$$

n	k	nk	w_s	w_n	w	Θ
1	4096	4096	4096	64	*64*	0.125
2	64	128	64	32	*32*	4
3	16	**48**	16	21	16	8
4	8	32	8	16	8	8
6	4	24	4	10	4	8
12	2	24	2	5	2	8

Table 1: effect of k and n on throughput

Consider what happens to the throughput as k and n vary. As k gets bigger, the first term gets smaller because even though channels are getting wider, the load is increasing faster, and the net effect is to get a lower BW. Meanwhile, nothing happens to the second term as k varies, indicating that the maximum throughput you can get from the network is the maximum BW across the middle of the network divided by the number of nodes that are sending. The factor of 2 comes from the fact that, with a uniform load, only half of the traffic crosses the bisection.

How is the throughput maximized given a set of constraints? Setting the node-limited term equal to the bisection-limited term will give the saturation point, the point at which the throughput is equally limited by both factors. If the dimension n is increased, the B_s term begins to dominate because the B_n term increases while B_s stays constant. (Remember, k increases much faster than n decreases, for constant k^n.) If n is decreased and B_s stays constant, the B_n term dominates as it gets smaller than B_s.

Which term do we want to limit the channel width? Both pin constraints and wire constraints are cost elements. However, we would rather be bisection bandwidth limited (wire constraint), because that gives the maximum throughput for nominally the same cost. In fact, the bisection limited network could end up being cheaper because it could allow for fewer pins per node. To demonstrate the effect of varying k and n on throughput, consider the following example: a torus with $N = 2^{12}$ nodes, $W_s = 2^{14}$, $W_n = 2^8$, and $f = 1Gb/s$.

The above table shows that the throughput increases with dimension until the bisection channels are saturated at a throughput of 8.

4.2 Latency

Earlier we defined latency in terms of head latency and serialization latency. Serialization latency is computed from packet length divided by minimum channel bandwidth:

$$T_s = \frac{L}{b} = \frac{L}{min(\frac{B_n}{4n}, \frac{kB_s}{4N})}$$

n	k	w	Θ	T_h	T_s	T
1	4096	64	0.125	10240	8	10248
2	64	32	4	320	16	336
3	16	16	8	120	32	152
4	8	8	8	80	64	144
6	4	4	8	60	128	188
12	2	2	8	60	256	316

Table 2: effect of k and n on latency and throughput

The above expression indicates the time it takes to squeeze a message across a channel. As the dimension is lowered, the serialization latency goes down because we get very wide, high-bandwidth channels. If the dimension is too low, though, the bisection channels fall out of saturation, we become node bandwidth limited, and throughput decreases.

Head latency depends on hop count. The average number of hops for a ring is $k/4$. This can be seen if we consider a loop with k nodes. The farthest destination from a given source node is $k/2$ hops away, since messages can travel around the loop in either direction. With random traffic, each message travels half of this maximum, or $k/4$ hops, in each of n dimensions. Thus, the hop count is

$$H = \frac{nk}{4}$$

To meet both latency and throughput constraints, we try to make latency as small as possible without decreasing throughput. To see how this works, we return to the earlier example with $N = 2^{12}$ nodes, specifying packet size $L = 512$ and latency per hop of 10ns. The table below shows what happens to both latency and throughput as we vary n and k.

A plot of the different latency components versus dimension is shown in the figure below. Note that the packet length, L, is an important factor although it has been fixed in this example. As L gets smaller, the serialization latency becomes smaller, and the curves will shift.

The total latency T is the sum of the head latency and the serialization latency. Note that T_h dominates for low dimensions, T_s dominates for high dimensions, and total latency is minimized when T_h and T_s are roughly equal. This minimum, which can be seen in the plot below, occurs when n is between 3 and 5. In this entire range, throughput is saturated at 8, so as designers we would probably choose the minimum value of $n = 4$. Typically, the order is reversed: we would pick n high enough to reach the flat part of the throughput curve and then minimize latency within that region.

The ability to tune network performance by varying k and n is one of the attractive properties of the torus topology. Contrast this to the butterfly, where nothing happens to throughput (except that it might go down if channel lengths are too small), and latency is optimized by picking biggest k that fits on a chip without shrinking the channel lengths.

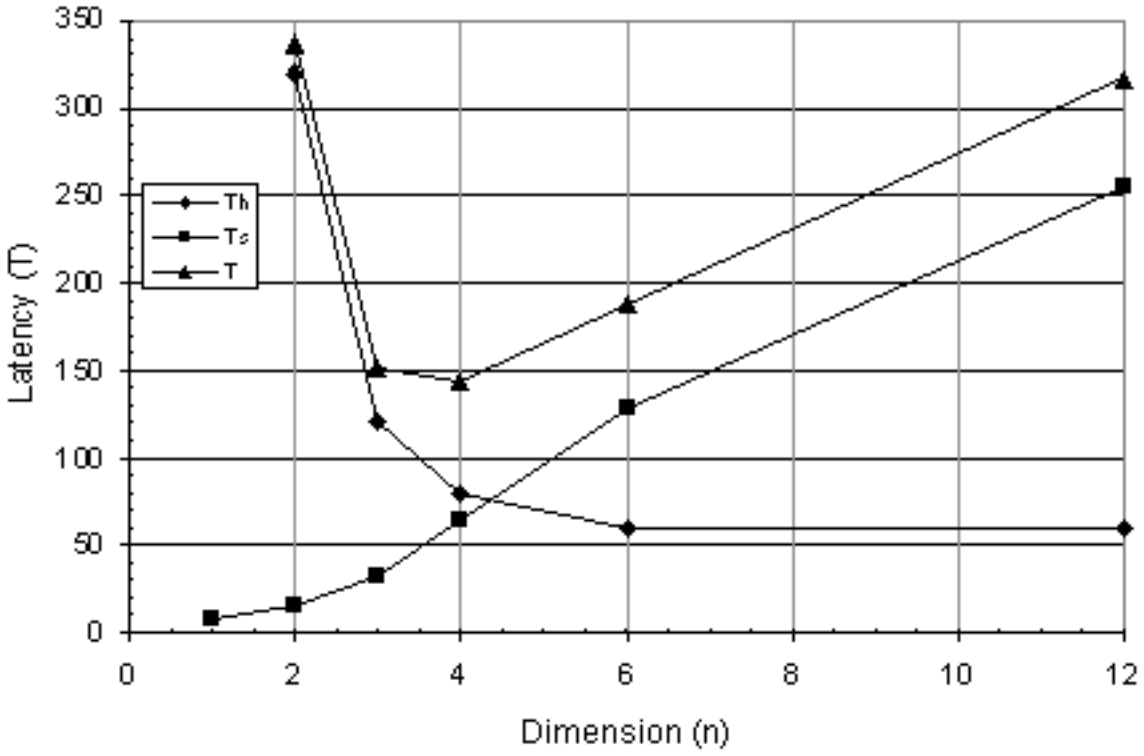

Figure 8: latency vs. dimension

4.3 Path Diversity

Suppose we have a 2D torus, and we want to go from node a to node b. If the distance between them is h_x in the x direction and h_y in the y direction, then how many minimal length paths are there from a to b?

Represent the path from a to b as a binary number with $h_x + h_y$ bits, where each 0 bit indicates a step in x and each 1 a step in y. Then a minimal length path is represented by a number with h_y 1's and h_x 0's. Now the number of minimal paths is simply the number of unique ways to arrange h_y 1's in a number with $h_x + h_y$ bits:

$$|R_{xy}| = \binom{h_x + h_y}{h_y}$$

Certain traffic patterns require non-minimal routing in order to keep the network load balanced. For example, suppose you have a k-node ring, and each node sends to the node that is $k/2 - 1$ away in the counter-clockwise direction. With this "tornado traffic," the average load per channel is four times worse than the $k/8$ seen with random traffic. First, we are sending traffic in one direction instead of two. Second, we are traversing

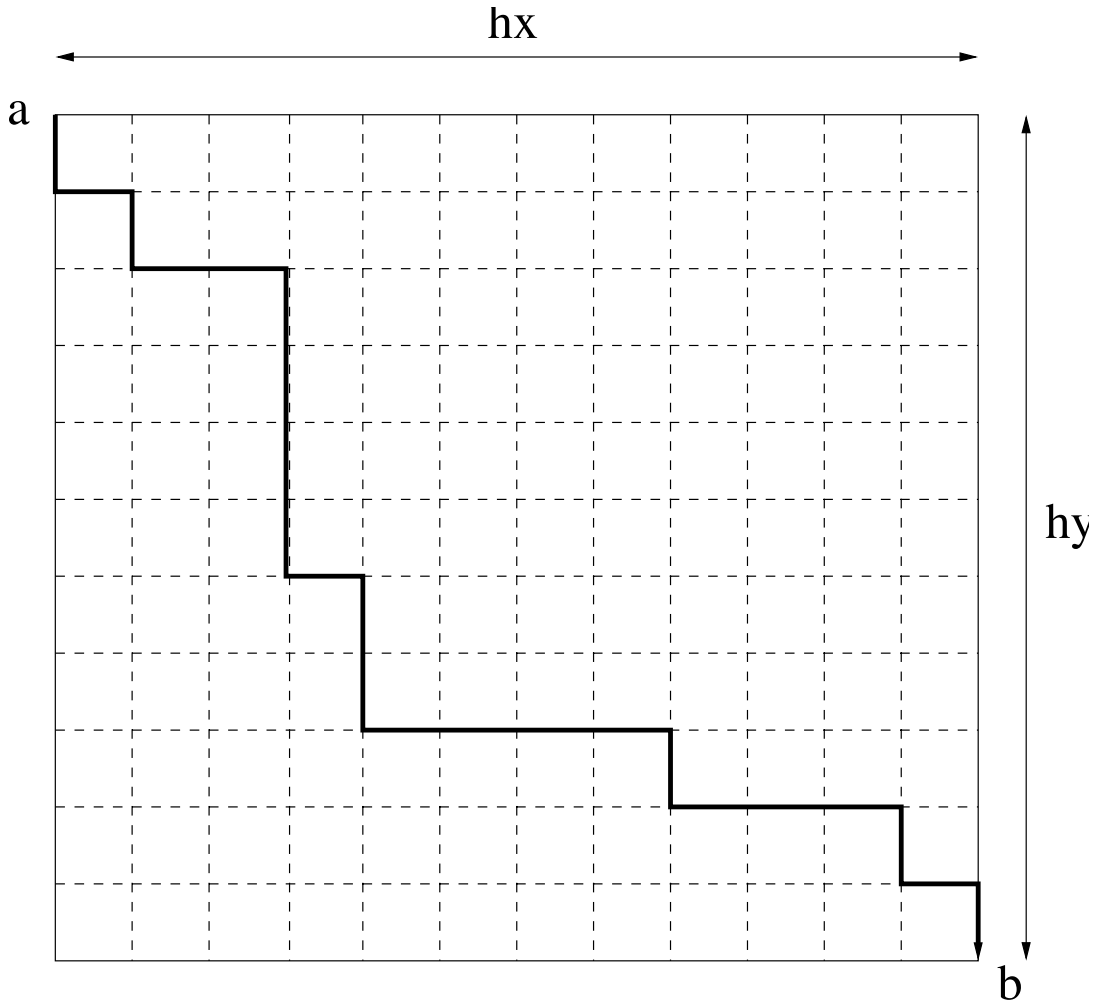

Figure 9: Paths from a to b

double the average number of channels with each route. To get back the first factor of 2, we use non-minimal routing, sending a fraction of the total traffic both ways around the ring. For a node that is h steps away, send h/k around the long way and $\frac{k-h}{k}$ around the short way. This results in a load per channel of $k/4$, recovering a factor of two.

Considering these additional, non-minimal paths results in a new equation for path diversity, with extra terms to account for the paths that take the long way in x, the long way in y, and the long way in both directions:

$$|R'_{ab}| = \binom{h_x + h_y}{h_x} + \binom{k - h_x + h_y}{k - h_x} + \binom{h_x + k - h_y}{h_x} + \binom{2k - h_x - h_y}{k - h_x}$$

This formula does not take all non-minimal paths into account. For example, paths with loops and meanders are not considered. Even so, this restricted version of non-minimal

h_x	h_y	h_z	$\|R_{ab}\|$	$\|R'_{ab}\|$
0	0	1	1	5,730,664
0	0	2	1	1,766,242
0	0	3	1	841,178
0	1	1	2	2,029,426
0	1	2	3	669,383
0	1	3	4	338,136
0	2	2	6	239,296
0	2	3	10	129,450
0	3	3	20	74,000
1	1	1	6	765,204
1	1	2	12	271,782
1	1	3	20	146,200
1	2	2	30	105,576
1	2	3	60	61,160
1	3	3	140	37,520
2	2	2	90	45,450
2	2	3	210	28,560
2	3	3	560	19,040
3	3	3	1680	13,440

Table 3: non-minimal paths in a 6-ary 3-cube

routing results in a very large number of routes. To illustrate this, the table below computes the number of non-minimal paths between nodes in a 6-ary 3-cube.

With this number of possible routes (over 10,000 at a minimum), the challenge is to find a good subset that can be stored in a routing table.