Routing in Hypercube Networks

Scribe: Keshav Kunal

February 18, 2003

Hypercube networks is a family of networks that have 2^n nodes for $n \in 0, 1, 2 \ldots$ such that a network of order n *i.e.* H_n is constructed recursively from two H_{n-1} networks with their corresponding nodes joined. H_0 is simply a node with no edges. Clearly, each node in H_n with $2^n = N$ nodes, has degree n and hence the number of edges $= \frac{N \cdot n}{2} \sim N \lg N$.

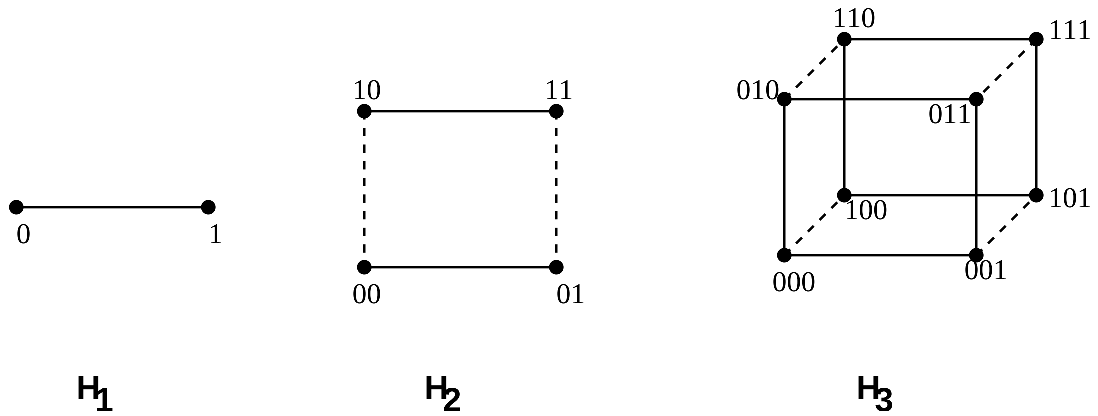

Figure 1: Hypercube Networks for $n = 1, 2$ and 3. Dotted lines are used to show the recursive construction from smaller hypercubes.

We use a n bit boolean vector representation to label nodes of H_n. First label both H_{n-1} networks recursively and add prefix '0' to one and '1' to another (see Fig.1). It is easy to see that by our construction there is an edge from node i to j if and only if their bit vectors differ in exactly one position.

We consider a a *permutation routing problem* on a network of N parallel processors, where each processor initially has one packet destined for some processor in the network such that each processor is the destination of exactly one packet(so, after the routing is complete, the packets have been permuted). We model the network as a hypercube network H_n with the following assumptions:

1. Each node represents a unique processor and edges represent comunication links between the processors.

2. Each edge can carry a unit message or *packet* in a step.

3. Each edge is bidirectional and can carry messages in either direction in the same step. [1]
4. A packet takes unit time, which we call a *round* to traverse an edge.
5. A processor can receive/send packets through any of the n edges simultaneously.
6. A route for a packet is a sequence of edges it can follow from its source to its destination.
7. In following a route, a packet may occasionally have to wait at an intermiediate node because the next edge on its route is *busy* transmitting another packet. We assume that each edge maintains a queue for packets waiting to leave via that edge.

A routing algorithm must specify the route for each packet and also a queueing discipline to resolve conflicts between packets that simultaneously wish to follow the same edge out of a node. We consider only algorithms where the source of each packet determines the complete path, which we call *oblivious algorithms*. We want to determine the minimum number of rounds required to route an arbitrary permutation request.

First we establish some trivial lower bounds. Define *traffic* on an edge e as the maximum number of packets that have to traverse e. Maximum traffic on any edge, c, serves as a lower bound on the number of rounds. Length of the longest path(measured in number of edges to be traversed), d, between all source-destination pairs also serves as a lower bound because that particular packet will take that many rounds atleast. Leighton,Maggs and Rao (see [1]) showed that for any levelled network there exists a schedule that takes $O(d+c)$ time and gave a randomized schedule that terminates in $O(c+d+\log N)$ rounds.

The following theorem, which we state without proof gives a limit on the performance of deterministic oblivious algorithms.

Theorem 1 *For any deterministic oblivious permutation routing algorithm on a network of N nodes each of out-degree d, there is an instance of permutation routing requiring $Omega(\sqrt{\frac{N}{d}}$ rounds.*

Using Theorem 1 on a hypercube network H_n, we know that for every deterministic routing algorithm, there is an instance of permutation that requires $\Omega(\sqrt{\frac{N}{n}})$ steps. We will give a randomized oblivious routing algorithm, as given in [3], such that it takes considerably less number of rounds.

First consider the bit-fixing routing strategy. Given that the source S_x and destination D_x for a packet x are n-bit vectors, scan the bits od D_x from left to right and compare it with the current location of x. Send the packet out of the current node along the edge corresponding to the left-most bit in which the current position and D_x differ. So, for instance in going from (1111) to (0100) in H_4, the packte would pass through (0111) and (0101). We first make a simple observation about any routing strategy and queueing discipline.

Observation 1 *If two packets are not in the same queue in the time interval $[t, t+1]$* i.e. *round $t+1$, then they can not delay each other in that round.*

[1] Or equivalently assume that each step is divided into two phases and in each pahse routing is allowed in a particular direction only.

Claim 1 *For any path $\Pi = x_1, x_2 \dots x_t, x_i \in$ set of nodes, if there are T packets which intersect the path(assume simple intersections* i.e. *once a packet leaves the path Π it does not enter Π again), then the maximum delay encountered by any packet in $\Pi \leq T - 1$.*

Proof: It is easy to see that if we can prove that a packet does not get delayed by another packet more than once, the claim is true. But we want to prove our claim irrespective of the queueing discipline. Suppose x has the largest delay among the T packets. We will use the following queueing policy to count the delay encountered by x:
If x is at node x_i in round $t+1$, then $\forall j > i$ and $\forall t' > t$, we will give priority to a packet starting (entering Π) at $x_{j'}, j' < j$.
Note that our policy is applied to queues not containing x and *ahead* of it.It does not affect the total delay encountered by x.

Observation 2 *Once a packet y gets* ahead *of x in Π, due to our queueing policy y and x will never be in the same queue again.*

This observation follows from the fact that y will never have to wait at any queue after it gets ahead of x. So, a packet will not delay x more than once. Since there are only $T-1$ packets apart from x, it can not be delayed by more than $T-1$ rounds. □

The above lemma used an assumption that a packet intersects a path only once. This is true for our bit-fixing routing strategy because once two paths separate, they will never meet again. Let's assume when the paths separated, the edges corresponded to *correcting* bits i and j, $i < j$. They can never meet again because at all future intermediate nodes they will differ in the i^{th} bit atleast. We now give a scheme that routes a packet using the bit-fixing algorithm in two stages, independently of all other packets, in the following manner. For a packet x that has to be routed from S_x to D_x :

i Route x from S_x to R_x which is a randomly selected node.

ii Route x from R_x to D_x.

We will first analyse the delays in first stage when a packet is routed to a randomly selected destination. Let $T(e_i)$ denote the traffic along edge e_i, which we earlier defined as the number of packets traversing that edge. Let Δ denote the delay encountered by a packet X_i whose path consists of edges $e_1, e_2 \dots e_k$, $k \leq n$. Let the random variable $H_{ij} = 1$ if paths of packets X_i and X_j intersect and 0 otherwise.

$$\begin{aligned}
\Delta &\leq \Sigma_{j=1}^{N} H_{ij} \\
&\leq \Sigma_{i=1}^{k} T(e_i) \\
E[\Delta] &\leq E[\Sigma_{i=1}^{k} T(e_i)] \\
&\leq \Sigma_{i=1}^{k} E[T(e_i)] \qquad (1)
\end{aligned}$$

Claim 2 $E[T(e_i)] = E[T(e_j)] = 1/2$

Proof: Consider an edge e_i which goes from node v_1 to v_2 such that the nodes differ in the k^{th} bit in the bit-vector representation. The possible source nodes which could send packet to v_1 can only differ in the first $k-1$ bits because they would have corrected errors only in those bits before using the edge e_i. So, the number of nodes which could route their packet along the edge is 2^{k-1}. For each such node, since the destination is chosen among the $N = 2^n$ nodes with equal probability, the probability that the destination has first k bits common with v_2 is $\frac{1}{2^k}$ (only then will they route along the edge e_i.) So, the expected traffic on edge $e_i = \frac{2^{k-1}}{2^k} = \frac{1}{2}$. Since this analysis was done for any general edge, the claim follows. □

An alternative proof of the above claim is that the expected number of edges traversed by each packet is $n/2$, so that the expected total number of edges traversed is $Nn/2$. Note that the number of edges in the hypercube is Nn. Due to symmetry, expected traffic on each edge will be same and hence $E[T(e_i)] = 1/2$ for any edge e_i.

Substituting the value in Eqn. 1 we get $E[\Delta] \leq \frac{n}{2}$. Since the routes for packets are chosen independently at random, the random variables H_{ij} are independent Poisson trials for $j \neq i$. Using Chernoff bounds, we get

$$\text{Prob.}(\Delta \geq c\alpha n) \leq \frac{1}{2^{n\alpha}} = \frac{1}{N^\alpha}$$

Note that we can not apply Chernoff bounds to $\Sigma_{i=1}^{k} E[T(e_i)]$ since the random variables $T(e_i)$ are not independent(and infact are not Poisson trials). We have used $\Sigma_{i=1}^{k} T(e_i)$ only to obtain an upper bound on $E[\Sigma_{j=1}^{N} H_{ij}]$ and then applied Chernoff bound to $\Sigma_{j=1}^{N} H_{ij}$.

We have shown that the maximum delay encountered by any packet in first stage is bounded by cn for constant c, with high probability. Note that the routing scheme for second stage can be viewed as the scheme for first stage *run backwards*. The same analysis carries through for it also, thus doubling the expected delays encountered by a packet. Thus we have achieved a significant improveement over the delays in any deterministic algorithm.

In our analysis we never tried to bound the queue length at any edge. We would like the maximum size of any queue to be constant while still maintaining the linear time(expected) routing scheme. Ranade's algorithm (see [2]) solves this problem, though we will not cover it in our course.

References

[1] Tom Leighton, Bruce Maggs, and Satish Rao. Universal packet routing algorithms (extended abstract). In *29th Annual Symposium on Foundations of Computer Science*, pages 256–269, White Plains, New York, 24–26 October 1988. IEEE.

[2] Abhiram G. Ranade. How to emulate shared memory. *Journal of Computer and System Sciences*, 42(3):307–326, June 1991.

[3] L. G. Valiant and G. J. Brebner. Universal schemes for parallel communication. In *Thirteenth Annual Symposium on Theory of Computation*, pages 263–277, Milwaukee, Wisconsin, 11–13 May 1981. ACM.