9.5 Application #1: Oblivious Routing on the Hypercube

Now we return to fourth application mentioned at the beginning of the chapter. (The first two applications have already been considered above, the third will be covered as a homework problem.)

The setting is the following: we are given the d-dimensional hypercube Q_d, with $n = 2^d$ vertices. We have $n = 2^d$ vertices, each labeled with a d-bit vector. Each vertex i has a single packet (which we also call packet i), destined for vertex $\pi(i)$, where π is a permutation on the nodes $[n]$.

Packets move in synchronous rounds. Each edge is bi-directed, and at most one packet can cross each directed edge in each round. Moreover, each packet can cross at most one edge per round. So if $uv \in E(Q_d)$, one packet can cross from u to v, and one from v to u, in a round. Each edge e has an associated waiting queue W_e; so each node has d queues, one for each edge leaving it. If several packets want to cross an edge e in the same round, only one can cross; the rest wait in the queue W_e and try again the next round. We assume the queues are allowed to grow to arbitrary size (though one can also show queue length bounds in the algorithm below). The goal is to get a simple routing scheme that delivers the packets in $O(d)$ rounds, no matter what permutation π needs to be routed.

One natural proposal is the *bit-fixing routing* scheme: each packet i looks at its current position u, finds the first bit position where u differs from $\pi(i)$, and flips the bit (which corresponds to traversing an edge out of u). For example:

$$0001010 \to 1001010 \to 1101010 \to 1100010 \to 1100011.$$

However, this proposal can create "congestion hotspots" in the network, and therefore delay some packets by $2^{\Omega(d)}$. In fact, it turns out any deterministic *oblivious* strategy (that does not depend on the actual sources and destinations) must have a delay of $\Omega(\sqrt{2^d/d})$ rounds.

Suppose we choose a permutation π such that

$$\pi(\mathbf{w0}) = \mathbf{0w},$$

where $\mathbf{w}, \mathbf{0} \in \{0,1\}^{d/2}$. All these $2^{d/2}$ packets have to pass through the all-zeros node in the bit-fixing routing scheme; since this node can send out at most d packets at each timestep, need at least $2^{d/2}/d$ rounds.

9.5.1 A Randomized Algorithm...

Here's a lovely randomized strategy, due to Les Valiant, and to Valiant and Brebner. It requires no centralized control, and is optimal in the sense of requiring $O(d)$ rounds (with high probability) on any permutation π.

Valiant (1982)

> Each node i picks a randomized midpoint R_i independently and uniformly from $[n]$: it sends its packet to R_i. Then after $5d$ rounds have elapsed, the packets proceed to their final destinations $\pi(i)$. All routing is done using bit-fixing.

9.5.2 ... and its Analysis

Theorem 9.16. *The random midpoint algorithm above succeeds in delivering the packets in at most* $10d$ *rounds, with probability at least* $1 - \frac{2}{n}$.

Proof. We only prove that all packets reach their midpoints by time $5d$, with high probability. The argument for the second phase is then identical. Let P_i be the bit-fixing path from i to the midpoint R_i, and define

$$S_i := \{j \neq i \mid \text{ path } P_j \text{ shares an edge with } P_i\}.$$

Claim 9.17. Any two paths P_i and P_j intersect in one contiguous segment.

Proof. (Exercise.) This is where using a consistent routing strategy like bit-fixing helps. □

Claim 9.18. Packet i reaches midpoint R_i by time at most $|P_i| + |S_i|$.

Proof. Consider the path $P_i = \langle e_1, e_2, \dots, e_\ell \rangle$ taken by packet i. If S_i were empty, clearly packet i would reach its destination in time $|P_i|$; we now show how to charge each timestep that packet i is delayed to a distinct packet in S_i. For that, we first define the notion of *lag*. For any edge $e_k \in P_i$, we say every packet in W_{e_k} at the beginning of timestep t has lag $t - k$. Note that all packets in the same queue at the same time have the same lag. Now:

Observe: the lags are defined for packets in S_i according to the numbering of edges in P_i, not the numbering of their own paths.

1. Each packet j in $S_i \cup \{i\}$ either reaches its destination on P_i or it leaves P_i (forever, by Claim 9.17) after traversing some last edge $e_k \in P_i$. Call this traversal of e_k the *final traversal* for packet j, and call its lag value just before this final traversal its *final lag*.

2. Suppose packet i traverses the last edge e_ℓ on its path and reaches its destination at timestep T. Since it has lag $T - \ell = T - |P_i|$ just before it traverses the edge, it reaches the destination at time $|P_i|$ plus its final lag. So it suffices to show that i's final lag is at most $|S_i|$.

3. The initial lag (at time $t = 1$) of this packet i is $(1 - 1) = 0$, since it belongs to queue W_{e_1} at the very beginning. The lag of this packet never decreases over time as it makes its way along the path. Indeed, if it is in W_{e_k} at the beginning of some timestep t, and it traverses the edge, it now belongs to $w_{e_{k+1}}$ at the start of timestep $t + 1$, and its new lag is $(t + 1) - (k + 1) = t - k$ and therefore unchanged.

4. Else suppose packet i's lag increases from some value L to $L + 1$ at some timestep. This is because $i \in W_{e_k}$ for some k at the beginning

of time $t = L + k$, but some other packet $j \in S_i$ from queue W_{e_k} was sent across the edge e_k at this timestep. In this case, imagine packet i gives packet j a *token* numbered L. So there is a single token generated for each increase in i's lag, each with a different number.

5. We show (in the next bullet point) how to maintain the invariant that at the beginning of each time, any token numbered L still on the path P_i is carried by some packet in S_i with current lag L. This implies that when a packet in S_i makes its final traversal and it has some final lag L', it is either carrying a single token numbered L' at that time or no token at all. Since each token is carried by some packet, this means there can be at most $|S_i|$ tokens overall, and hence i's final lag value is at most $|S_i|$.

6. To ensure the invariant, note that when j got the token numbered L from i, packet j had lag value L. Now as long as j does not get delayed as it proceeds along the path, its lag remains L (and it keeps the token). If it does get delayed, say while waiting in queue $W_{e_{k'}}$ while some other packet j' (having the same lag value L, because they were sharing the same queue) traverses the edge $e_{k'}$, packet j gives its token numbered L to this j'. This maintains the invariant.

□

Finally, we bound the size of S_i by a concentration bound. Since R_i is chosen uniformly at random from $\{0,1\}^d$, the labels of i and R_i differ in $d/2$ bits in expectation. Hence P_i has expected length $d/2$. There are $d2^d = dn$ (directed) edges, and all $n = 2^d$ paths behave symmetrically, so the expected number of paths P_j using any edge e is $\frac{n \cdot d/2}{dn} = 1/2$.

Claim 9.19. $\Pr[|S_i| \geq 4d] \leq e^{-2d}$.

Proof. If X_{ij} is the indicator of the event that P_i and P_j intersect, then $|S_i| = \sum_{j \neq i} X_{ij}$, i.e., it is a sum of a collection of independent $\{0,1\}$-valued random variables. Now conditioned on any choice of P_i (which is of length at most d), the expected number of paths using each edge in it is at most $1/2$, so the conditional expectation of S_i is at most $d/2$. Since this holds for any choice of P_i, the unconditional expectation $\mu = \mathbb{E}[S_i]$ is also at most $d/2$.

Now apply the Chernoff bound to S_i with $\lambda = 4d - \mu$ and $\mu \leq d/2$ to get

$$\Pr[|S_i| \geq 4d] \leq \exp\left\{-\frac{(4d-\mu)^2}{2\mu + (4d-\mu)}\right\} \leq e^{-2d}.$$

Note that we could apply the bound even though the variables X_{ij} were not i.i.d., and moreover we did not need estimates for $\mathbb{E}[X_{ij}]$, just an upper bound for their expected sum. $\square$

Now applying a union bound over all $n = 2^d$ packets i means that all n packets reach their midpoints within $d + 4d$ steps with probability $1 - 2^d \cdot e^{-2d} \geq 1 - e^{-d} \geq 1 - 1/n$. Similarly, the second phase has a probability at most $1/n$ of failing to complete in $5d$ steps, completing the proof. $\square$

A different strategy would be to let each packet pick a random permutation and fix the bits according to that permutation. Sadly, this approach gives delay $2^{\Omega(d)}$. This is true even if each node picks its permutation independently. One bad example appears in Valiant's original paper (see Section 5 "The Necessity for Phase A") and shows that you can fix a permutation that "gangs up" on some node, even if the bit-fixing order is random.

9.6 *Application #2: Graph Sparsification*

9.7 *Application #3: The Power of Two Choices*