University of Tokyo: Advanced Algorithms **Summer 2010**

Lecture 9 — 17 June

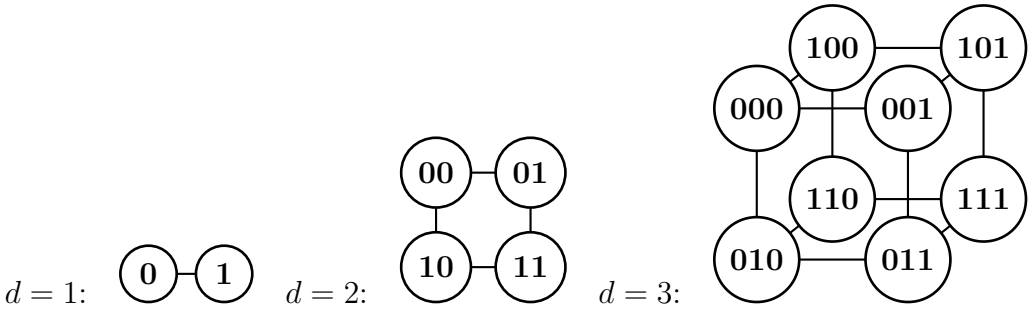
Lecturer: Christian Sommer *Scribe: Adrian Klein*

9.1 Permutation Routing, Part 1/2

In the next two lectures we consider *routing* problems, where several computers communicate with each other. We can represent the computers as nodes and their communication links (wires) as edges in a graph. We assume that time passes in discrete time steps. Each node can send and receive at the same time, but at each step *at most one* packet can be transmitted per edge (wire). The graph (network) we consider in this lecture is the *d–dimensional hypercube*.

9.1.1 Hypercube

A d–dimensional hypercube can be imagined as the d–dimensional version of a square ($d = 2$) or a cube ($d = 3$).



A hypercube of dimension d has $n = 2^d$ nodes (see example above for $d \in \{1, 2, 3\}$). We can construct any hypercube recursively by starting with a hypercube with dimension 1, and then iteratively combining two hypercubes of dimension $d - 1$ by connecting corresponding nodes until we have the desired dimension.

Note In the d–dimensional hypercube, all the distances (number of edges between two nodes) are at most d.

9.1.2 Permutation Routing Problem

Each computer initially has exactly one packet to be routed to its destination. Furthermore, each computer shall receive exactly one packet. The problem thus corresponds to a permutation of the address space of the computers. An example problem, denoting the initial source with i and its destination with $\pi(i)$, could look like the following (Table 9.1):

Table 9.1. An example of a permutation routing problem

i	000	001	010	011	100	101	110	111
$\pi(i)$	011	100	101	001	000	111	010	110

9.1.3 The Bit-Fixing Algorithm

A natural algorithm solving this problem is the *bit-fixing* algorithm: at each step exactly one bit of the address is "fixed" (by sending the packet along an edge to another node), starting from the leftmost. For instance, a packet routed from node (1011) to node (0000) would first pass through node (0011) and then through node (0001) before reaching its final destination.

Question How many steps does the bit-fixing algorithm require to send n packets? The issue is the congestion that occurs when we send more than one packet through the same wire at the same time. This congestion occurs even though there are no *critical* edges (edges with significantly higher load than others) in a hypercube.

Worst-Case Analysis

We analyze the number of steps required by the bit-fixing algorithm for a worst-case instance. We let an adversary choose a permutation π (which our algorithm is not allowed to look at). We prove that, for such a worst-case permutation, bit fixing requires at least $\Omega(\sqrt{n}/d)$ rounds. First, we try to find a permutation for which bit fixing requires many rounds. As each path has length at most d, we need to find a permutation for which the congestion is really high. Let us assume that d is even. We split each address into two binary vectors of length $d/2$. The permutation π is defined as $\pi(ab) = ba$. For example, routing from the binary representation of 8913 (ab) to the binary representation of 53538 (ba) has the following steps (Table 9.2):

Table 9.2. Example Routing Steps for $ab = 8913$

a	b
00 10 00 10	11 01 00 01
10 10 00 10	11 01 00 01
11 10 00 10	11 01 00 01
⋮	⋮
11 01 00 01	11 01 00 01
⋮	⋮
11 01 00 01	00 10 00 10
b	a

Note that there is a node whose address is of the form aa, meaning that the left part equals the right part. The bit-fixing algorithm uses such a node aa for any pair $ab \rightsquigarrow ba$.

There are at most $2^{d/2} = \sqrt{n}$ many nodes of the form aa but n packets to be routed through these nodes. Each one of these nodes has d links and, by definition, there can be at most one packet per link at any time step. Routing n packets through $\sqrt{n}$ nodes, sending at most d packets at any given time step, we obtain the lower bound of $\Omega(\sqrt{n}/d)$ rounds.

Furthermore, one can prove the same lower bound for any deterministic oblivious routing strategy [KKT91].

9.1.4 Randomized Algorithm

We use a randomized algorithm to "protect" ourselves against worst-case instances. The idea is to choose for each node i a random intermediate node $\sigma(i)$, where we first send the packet to, and from which we then forward the packet to the final target. The routing procedure for the packet at node i works as follows.

1. send packet to $\sigma(i)$ and wait until time $7d$
2. send packet to $\pi(i)$

Theorem 9.1. *With probability at least $1-1/n$ every packet reaches its destination in time at most $14d$.*

The packet i is first routed to $\sigma(i)$ and then forwarded on to $\pi(i)$. Let $g_i = (e_1, e_2, ..., e_k)$ denote the path (sequence of edges) of a packet v_i routed from i to $\sigma(i)$ (the first part of the route). We know that k is at most d (path length of the hypercube). Let S_i denote the set of packets except i also using one of these edges. These packets could potentially interfere or cause congestion. Note that each packet in S_i can cause at most one time step of delay.

Claim 1. *The delay of packet i is at most $|S_i|$.*

If we can bound the size of S_i then we can bound the time required for packet i to arrive at $\sigma(i)$. We define random variables to count the number of packets in S_i.

$$\text{Random variable (RV): } H_{ij} = \begin{cases} 1 & \text{if packet } j \in S_i \\ 0 & \text{otherwise} \end{cases}$$

We then have $|S_i| = \sum_{j=1}^{n} H_{ij}$ and, consequently, $\mathsf{E}[|S_i|] = \mathsf{E}[\sum_{j=1}^{n} H_{ij}]$.

For an edge e, let the random variable R_e denote the number of routes through e.

Claim 2. $E[R_e] = 1/2$.

Proof: First, note that R_e is the same for all e, since everything is symmetric in a hypercube. The expected path length is $d/2$, the expected total path length is $nd/2$. The number of directed edges in a hypercube is nd. Therefore, $\mathsf{E}[R_e] = \frac{\frac{nd}{2}}{nd} = 1/2$. □

Summing up the R_e values over all the edges of a path $(e_1, e_2, ..., e_k)$, we obtain that

$$\mathsf{E}[|S_i|] = \mathsf{E}[\sum_{j=1}^{n} H_{ij}] \leq \mathsf{E}[\sum_{l=1}^{k} R_{e_l}] = \sum_{l=1}^{k} \mathsf{E}[R_{e_l}] = k/2 \leq d/2.$$

The expected time until packet i arrives at $\sigma(i)$ is at most $d+d/2$. Next, we wish to bound the probability that *none* of the packets arrives too late. We claim that the probability that $Pr[\exists i. |S_i| > 6d]$ is low. To prove this claim, we use *tail inequalities*. Note that the R_e's are not independent. While being convenient to compute expectations, we may not use the R_e's in tail inequalities that require independency of the variables.

Markov Inequality For a random variable $X > 0$ and for a parameter $t > 0$ we have

$$\mathsf{Pr}[X \geq t] \leq \mathsf{E}[X]/t.$$

Proof

$$\mathsf{E}[X] = \sum_{x} x\mathsf{Pr}[X = x] \geq \sum_{x \geq t} x\mathsf{Pr}[X = x] \geq t \sum_{x \geq t} \mathsf{Pr}[X = x] = t\mathsf{Pr}[X \geq t].$$

Chernoff Inequality Let random variables $X_i > 0$ independent. Then we have the following statement on their sum $X := X_1 + \ldots X_n$. Let $\mu = \mathsf{E}[X]$. For any $\epsilon > 0$,

$$\mathsf{Pr}[X > (1+\epsilon)\mu] < \left(\frac{e^{\epsilon}}{(1+\epsilon)^{1+\epsilon}}\right)^{\mu}.$$

Note that the X_i must be independent. We now bound $\mathsf{Pr}[|S_i| > 6d]$ using $|S_i| = \sum_{j=1}^{n} H_{ij}$, the Chernoff inequality, and the fact that the H_{ij} are independent. We may also use the fact that $\mathsf{E}[|S_i|] \leq d/2$. We have that $\mathsf{Pr}[|S_i| > 6d] < 2^{-6d}$. Therefore, for 2^d packets, the probability for one of them to require more than $d+6d$ steps is at most 2^{-5d} (using the union bound). Including the second phase, the overall time is at most $7d + 7d$.

Bibliography

[KKT91] Christos Kaklamanis, Danny Krizanc, and Thanasis Tsantilas. Tight bounds for oblivious routing in the hypercube. *Theory of Computing Systems*, 24(1):223–232, 1991.