Principles of Distributed Computing
Roger Wattenhofer

9 Routing Strikes Back

9.1 Butterfly

Let's first assume that all the sources are on level 0, all destinations are on level d of a d-dimensional butterfly.

Algorithm 9.1 (Greedy Butterfly Routing) *The unique path from a source on level 0 to a destination on level d with d hops is the greedy path. In the greedy butterfly routing algorithm each packet is constrained to follow its greedy path.*

Remark:

- In the bit-reversal permutation routing problem, the destination of a packet is the bit-reversed address of the source. With $d = 3$ you can see that both source $(000, 0)$ and source $(001, 0)$ route through edge $(000, 1..2)$. Will the contention grow with higher dimension? Yes! Choose an odd d, then all the sources $(0 \ldots 0 b_{(d+1)/2} \ldots b_{d-1}, 0)$ will route through edge $(00..0, (d-1)/2...(d+1)/2)$. You can choose the bits b_i arbitrarily. There are $2^{(d+1)/2}$ bit combinations, which is $\sqrt{n/2}$ for $n = 2^d$ sources.

- On the good side, this contention is also a guaranteed time bound, as the following theorem shows.

Theorem 9.2 (Analysis) *The greedy butterfly algorithm terminates in $O(\sqrt{n})$ steps.*

Proof. For simplicity we assume that d is odd. An edge on level l (from a node on level l to a node on level $l + 1$) has at most 2^l sources, and at most 2^{d-l-1} destinations. Therefore the number of paths through an edge on level l is bounded by $n_l = 2^{\min(l,d-l-1)}$. A packet can therefore be delayed at most $n_l - 1$ times on level l. Summing up over all levels, a packet is delayed at most

$$\sum_{l=0}^{d-1} n_l = \sum_{l=0}^{(d-1)/2} n_l + \sum_{l=(d+1)/2}^{d-1} n_l = \sum_{l=0}^{(d-1)/2} 2^l + \sum_{l=0}^{(d-3)/2} 2^l < 3 \cdot 2^{(d-1)/2} = O(\sqrt{n}).$$

steps. $\sqcap$

Remarks:

- The bit-reversed routing is therefore asymptotically a worst-case example.

- However, one that requires square-root queues. When being limited to constant queue sizes the greedy algorithm can be forced to use $\Theta(n)$ steps for some permutations.

- A routing problem where all the sources are on level 0 and all the destinations are on level d is called an end-to-end routing problem. Surprisingly, solving an arbitrary routing problem on a butterfly (or any hypercubic network) is often not harder.

- In the next section we show that there is general square-root lower bound for “greedy” algorithms for any constant-degree graph. (In other words, our optimal greedy mesh routing algorithm of Chapter 4 was only possible because the mesh has such a bad diameter...)

9.2 Oblivious Routing

Definition 9.3 (Oblivious) *A routing algorithm is oblivious if the path taken by each packet depends only on source and destination of the packet (and not on other packets, or the congestion encountered).*

Theorem 9.4 (Lower Bound) *Let G be a graph with n nodes and (maximum) degree d. Let A be any oblivious routing algorithm. Then there is a one-to-one routing problem for which A will need at least $\sqrt{n}/2d$ steps.*

Proof. Since A is oblivious, the path from node u to node v is $P_{u,v}$; A can be specified by n^2 paths. We must find k one-to-one paths that all use the same edge e. Then we can proof that A takes at least $k/2$ steps.

Let's look at the $n-1$ paths to destination node v. For any integer k let $S_k(v)$ be the set of edges in G where k or more of these paths pass through them. Also, let $S_k^*(v)$ be the nodes incident to $S_k(v)$. Since there are two nodes incident to each edge $|S_k^*(v)| \leq 2|S_k(v)|$. In the following we assume that $k \leq (n-1)/d$; then $v \in S_k^*(v)$, hence $|S_k^*(v)| > 0$.

We have

$$n - |S_k^*(v)| \leq (k-1)(d-1)|S_k^*(v)|$$

because every node u not in $S_k^*(v)$ is a start of a path $P_{u,v}$ that enters $S_k^*(v)$ from outside. In particular, for any node $u \notin S_k^*(v)$ there is an edge (w, w') in $P_{u,v}$ that enters $S_k^*(v)$. Since the edge $(w, w') \notin S_k(v)$, there are at most $(k-1)$ starting nodes u for edge (w, w'). Also there are at most $(d-1)$ edges adjacent to w' that are not in $S_k(v)$. We get

$$n \leq (k-1)(d-1)|S_k^*(v)| + |S_k^*(v)| \leq 2[1 + (k-1)(d-1)]|S_k(v)| \leq 2kd|S_k(v)|$$

Thus, $|S_k(v)| \geq \frac{n}{2kd}$. We set $k = \sqrt{n}/d$, and sum over all n nodes:

$$\sum_{v \in V} |S_k(v)| \geq \frac{n^2}{2kd} = \frac{n^{3/2}}{2}$$

Since there are at most $nd/2$ edges in G, this means that there is an edge e for at least

$$\frac{n^{3/2}/2}{nd/2} = \sqrt{n}/d = k$$

different values of v.

Since edge e is in at least k different paths in each set $S_k(v)$ we can construct a one-to-one permutation problem where edge e is used $\sqrt{n}/d$ times (directed: $\sqrt{n}/2d$ contention). $\sqcap$

Remarks:

- In fact, as many as $(\sqrt{n}/d)!$ one-to-one routing problems can be constructed with this method.

- The proof can be extended to the case where the one-to-one routing problem consists of R route requests. The lower bound is then $\Omega(\frac{R}{d\sqrt{n}})$.
- There is a node that needs to route $\Omega(\sqrt{n/d})$ packets.
- The lower bound can be extended to randomized oblivious algorithms... however, if we are allowed to use randomization, the lower bound gets much weaker. In fact, one can use Valiant's trick also in the butterfly: In a first phase, we route each packet on the greedy path to a random destination on level d, in the second phase on the same row back to level 0, and in a third phase on the greedy path to the destination. This way we can escape the bad one-to-one problems with high probability. (There are much more good one-to-one problems than bad one-to-one problems.) One can show that with this trick one can route any one-to-one end-to-end routing problem in asymptotically optimal $O(\log n)$ time (with high probability).
- If a randomized algorithm fails (takes too long), simply re-run it. It will be likely to succeed then. On the other hand, if a deterministic algorithm fails in some rare instance, re-running it will not help!

9.3 Offline Routing

There are a variety of other aspects in routing. In this section we study one of them to gain further insights.

Definition 9.5 (Offline Routing) *We are given a routing problem (graph and set of routing requests). An offline routing algorithm is a (not distributed) algorithm that sees the whole input (the routing problem).*

Remarks:

- Offline routing is worth being studied because the same communication pattern might appear whenever you run your (important!) (parallel) algorithm.
- In offline routing, path selection and scheduling can be studied independently.

Definition 9.6 (Path Selection) *We are given a routing problem (a graph and a set of routing requests). A path selection algorithm selects a path (a route) for each request.*

Remarks:

- Path selection is efficient if the paths are "short" and do not interfere if they do not need to. Formally, this can be defined by congestion and dilation (see below).
- For some routing problems, path selection is easy. If the graph is a tree, for example, the best path between two nodes is the direct path. (Every route from a source to a destination includes at least all the links of the shortest path.)

Definition 9.7 (Dilation, Congestion) *The dilation of a path selection is the length of a maximum path. The contention of an edge is the number of paths that use the edge. The congestion of a path selection is the load of a most contended edge.*

Remarks:

- A path selection should minimize congestion and dilation.
- Networking researchers have defined the "flow number" which is defined as the minimum $\max$(congestion, dilation) over all possible path selections.
- Alternatively, congestion can be defined with directed edges, or nodes.

Definition 9.8 (Scheduling) *We are given a set of source-destination paths. A scheduling algorithm specifies which messages traverse which link at which time step (for an appropriate model).*

Remark: The most popular model is store-and-forward (with small queues). Other popular models have no queues at all: e.g. hot-potato routing or direct routing (where the source might delay the injection of a packet; once a packet is injected however, it will go to the destination without stop.)

Lemma 9.9 (Lower Bound) *Scheduling takes at least $\Omega(C + D)$ steps, where C is the congestion and D is the dilation.*

Remark: We aim for algorithms that are competitive with the lower bound. (As opposed to algorithms that finish in $O(f(n))$ time; $C + D$ and n are generally not comparable.)

Algorithm 9.10 (Direct Tree Routing) *We are given a tree, and a set of routing requests. (Since the graph is a tree each route request will take the direct path between source and destination; in other words, path selection is trivial.) Choose an arbitrary root r. Now sort all packets using the following order (breaking ties arbitrarily): packet p comes before packet q if the path of p reaches a node closer to r then the path of q. Now scan all packets in this order, and for each packet greedily assign its injection time to be the first that does not cause a conflict with any previous packet.*

Theorem 9.11 (Analysis) *Algorithm 9.10 terminates in $2C + D$ steps.*

Proof. A packet p first goes up, then down the tree; thus turning at node u. Let e_u and e_d be the "up" resp. "down" edge on the path adjacent to u. The injection time of packet p is only delayed by packets that traverse e_u or e_d (if it contends with a packet q on another edge, and packet q has not a lower order, then it contends also on e_u or e_q). Since congestion is C, there are at most $2C - 2$ many packets q. Thus the algorithm terminates after $2C + D$ steps. $\sqcap$

Remark: [Leighton, Maggs, Rao 1988] have shown that the existence of a $O(C + D)$ schedule for any routing problem (on any graph!) using the Lovasz Local Lemma. Later the result was made more accessible by [Leighton, Maggs, Richa 1996] and others. Still it is too hard for this course...