Principles of Distributed Computing
Roger Wattenhofer

4 Routing

4.1 Array

(Routing is important for any distributed system. This chapter is only an introduction into routing; we will see other facets of routing in a next chapter.)

Definition 4.1 (Routing) *We are given a graph and a set of routing requests, each defined by a source and a destination node.*

Definition 4.2 (One-to-one, Permutation) *In a one-to-one routing problem, each node is the source of at most one packet and each node is the destination of at most one packet. In a permutation routing problem, each node is the source of exactly one packet and each node is the destination of exactly one packet.*

Remark:

- Permutation routing is a special case of one-to-one routing.

Definition 4.3 (Store and Forward Routing) *The network is synchronous. In each step, at most two packets (one in each direction) can be sent over each link.*

Remark:

- If two packets want to follow the same link, then one is queued (stored) at the sending node. This is known as contention.

Algorithm 4.4 (Greedy on Array) *An array is a linked list of n nodes; that is, node i is connected with nodes $i-1$ and $i+1$, for $i = 2, \ldots, n-1$. With the greedy algorithm, each node injects its packet at time 0. At each step, each packet that still needs to move rightward or leftward does so.*

Theorem 4.5 (Analysis) *The greedy algorithm terminates in $n-1$ steps.*

Proof. By induction two packets will never contend for the same link. Then each packet arrives at its destination in d steps, where d is the distance between source and destination. □

Remarks:

- Unfortunately, only the array (or the ring) allows such a simple contention-free analysis. Already in a tree (with nodes of degree 3 or more) there might be two packets arriving at the same step at the same node, both want to leave on the same link, and one needs to be queued. In a "Mercedes-Benz" graph $\Omega(n)$ packets might need to be queued. We will study this problem in the next section.
- There are many strategies for scheduling packets contending for the same edge (e.g. "farthest goes first"); these queuing strategies have a substantial impact on the performance of the algorithm.

4.2 Mesh

Algorithm 4.6 (Greedy on Mesh) *A mesh (a.k.a. grid, matrix) is a two-dimensional array with m columns and m rows ($n = m^2$). Packets are routed to their correct column (on the row in greedy array style), and then to their correct row. The farthest packet will be given priority.*

Theorem 4.7 (Analysis) *The greedy algorithm terminates in $2m - 2$ steps.*

Proof. First note that packets in the first phase of the algorithm do not interfere with packets in the second phase of the algorithm. With Theorem 4.5 each packet arrives at its correct column in $m - 1$ steps. (Some packets may arrive at their turning node earlier, and already start the second phase; we will not need this in the analysis.) We need the following Lemma for the second phase of the algorithm.

Lemma 4.8 (Many-to-One on Array, Lemma 1.5 in Leighton Section 1.7) *We are given an array with n nodes. Each node is a destination for at most one packet (but may be the source of many). If edge contention is resolved by farthest-to-go (FTG), the algorithm terminates in $n - 1$ steps.*

Proof. [Leighton Section 1.7 Lemma 1.5] Leftward moving packets and rightward moving packets never interfere; so we can restrict ourselves to rightward moving packets. We name the packets with their destination node. Since the queuing strategy is FTG, packet i can only be stopped by packets $j > i$. Note that a packet i may be contending with the same packet j several times. However, packet i will either find its destination "among" the higher packets, or directly after the last of the higher packets. More formally, after k steps, packets $j, j + 1, \ldots, n$ do not need links $1, \ldots, l$ anymore, with $k = n - j + l$. Proof by induction: Packet n has the highest priority: After k steps it has escaped the first k links. Packet $n - 1$ can therefore use link l in step $l + 1$, and so on. Packet i not needing link i in step $k = n$ means that packet i has arrived at its destination node i in step $n - 1$ or earlier. □

Lemma 4.8 completes the proof. □

Remarks:

- A $2m - 2$ time bound is the best we can hope for, since the distance between the two farthest nodes in the mesh is exactly $2m - 2$.
- One thing still bugs us: The greedy algorithm might need queues in the order of m. And queues are expensive! In the next section, we try to bring the queue size down!

4.3 Routing in the Mesh with Small Queues

(First we look at a slightly simpler problem.)

Definition 4.9 (Random Destination Routing) *In a random destination routing problem, each node is the source of at most one packet with destination chosen uniformly at random.*

Remarks:

- Random destination routing is not one-to-one routing. In the worst case, a node can be destination for all n packets, but this case is very unlikely (with probability $1/n^{n-1}$)
- We study algorithm 4.6, but this time in the random destination model. Studying the random destination model will give us a deeper understanding of routing... and distributed computing in general!

Theorem 4.10 (Random destination analysis of algorithm 4.6) *If destinations are chosen at random the maximum queue size is $O(\log n / \log\log n)$ with high probability. (With high probability means with probability at least $1 - O(1/n)$.)*

Proof. We can restrict ourselves to column edges because there will not be any contention at row edges. Let us consider the queue for a north-bound column edge. In each step, there might be three packets arriving (from south, east, west). Since each arriving south packet will be forwarded north (or consumed when the node is the destination), the queue size can only grow from east or west packets – packets that are "turning" at the node. Hence the queue size of a node is always bounded by the number of packets turning at the node. A packet only turns at a node u when it is originated at a node in the same row as u (there are only m nodes in the row). Packets have random destinations, so the probability to turn for each of these packets is $1/m$ only. Thus the probability P that r or more packets turn in some particular node u is at most

$$P \leq \binom{m}{r}\left(\frac{1}{m}\right)^r$$

(The factor $(1-1/m)^{m-r}$ is not present because the event "exactly r" includes the event "more than r" already.) Using

$$\binom{x}{y} < \left(\frac{xe}{y}\right)^y, \text{ for } 0 < y < x$$

we directly get

$$P < \left(\frac{me}{r}\right)^r \left(\frac{1}{m}\right)^r = \left(\frac{e}{r}\right)^r$$

Hence most queues do not grow larger than $O(1)$. Also, when we choose $r := \frac{e \log n}{\log\log n}$ we can show $P = o(1/n^2)$. The probability that any of the $4n$ queues ever exceeds r is less than $1-(1-P)^{4n} = o(1/n)$. ⊓

Remarks:

- OK. We got a bound on the queue size. Now what about time complexity?!? The same analysis as for one-to-one routing applies. The probability that a node sees "many" packets in phase 2 is small... it can be shown that the algorithm terminates in $O(m)$ time with high probability.
- In fact, maximum queue sizes are likely to be a lot less than logarithmic. The reason is the following: Though $\Theta(\log n / \log\log n)$ packets might turn at some node, these turning packets are likely to be spread in time. Early arriving packets might use gaps and do not conflict with late arriving packets. With a much more elaborate method (using the so-called "wide-channel" model) one can show that there will never be more than four(!) packets in any queue (with high probability only, of course).

- Unfortunately, the above analysis only works for random destination problems. Question: Can we devise an algorithm that uses small queues only but for any one-to-one routing problem? Answer: Yes, we can! In the simplest form we can use a clever trick invented by Leslie Valiant: Instead of routing the packets directly on their row-column path, we route each packet to a randomly chosen intermediate node (on the row-column path), and from there to the destination (again on the row-column path). Valiant's trick routes all packets in $O(m)$ time (with high probability) and only needs queues of size $O(\log n)$. Instead of choosing a random intermediate node one can choose a random node that is more or less in the direction of the destination, solving any one-to-one routing problem in $2m + O(\log n)$ time with only constant-size queues. You don't wanna know the details...
- What about no queues at all?!?

4.4 Hot-Potato Routing

Definition 4.11 (Hot-Potato Routing) *Like the store-and-forward model the hot-potato model is synchronous and at most two packets (one in each direction) can be sent over a link. However, contending packets cannot be stored; instead all but one contending packet must be sent over a "wrong link" (known as deflection) immediately, since the hot-potato model does not allow queuing.*

Remarks:

- Don't burn your fingers with "hot-potato" packets. If you get one you better forward it directly!
- A node with degree δ receives up to δ packets at the beginning of each step – since the node has δ links, it can forward all of them, but unfortunately not all in the right direction.
- Hot-potato routing is easier to implement, especially on light-based networks, where you don't want to convert photons into electrons and then back again. There are a couple of parallel machines that use the hot-potato paradigm to simplify and speed up routing.
- How bad does hot-potato routing get (in the random or the one-to-one model)? How bad can greedy hot-potato routing (greedy: whenever there is no contention you must send a packet into the right direction) get in a worst case?

Algorithm 4.12 (Greedy Hot-Potato Routing on a Mesh) *Packets move greedy towards their destination (any good link is fine if there is more than one). If a packet gets deflected, it gets excited with probability p (we set $p = \Theta(1/m)$). An excited packet has higher priority. When being excited, a packet tries to reach the destination on the row-column path. If two excited packets contend, then the one that wants to exit the opposite link is given priority. If an excited packet fails to take its desired link it becomes normal again.*

Theorem 4.13 (Analysis) *A packet will reach its destination in $O(m)$ expected time.*

Proof. [Sketch, full proof in Busch et al., SODA 2000] An excited packet can only be deflected at its start node (after becoming excited), and when trying to turn. In both cases, the probability to fail is

only constant since other excited packets need to be at the same node at exactly the right instant. Thus the probability that an excited packets finds to its destination is constant, and therefore a packet needs to "try" (to become excited) only constantly often. Since a packet tries every p'th time it gets deflected, in only gets deflected $O(1/p) = O(m)$ times in expectation. Since each time it does not get deflected, it gets closer to its destination, it will arrive at the destination in $O(m)$ expected time. $\sqcap$

Remarks:

- It seems that at least in expectation having no memory at all does not harm the time bounds much.
- It is conjectured that one-to-one routing can be shown to have time complexity $O(m)$ for this greedy hot-potato routing algorithm. However, the best known bound needs an additional logarithmic factor.

4.5 More Models

Routing comes in many flavors. We mention some of them in this section for the sake of completeness.

Store-and-forward and hot-potato routing are variants of packet-switching. In the circuit-switching model, the entire path from source to destination must be locked such that a stream of packets can be transmitted.

A packet-switching variant where more than one packet needs to be sent from source to destination in a stream is known as wormhole routing.

Static routing is when all the packets to be routed are injected at time 0. Instead, in dynamic routing, nodes may inject new packets constantly (at a certain rate). Not much is known for dynamic routing.

Instead of having a single source and a single destination for each packet as in one-to-one routing, researchers have studied many-to-one routing, where a node may be destination for many sources. The problem of many-to-one routing is that there might be congested areas in the network (areas with nodes that are destinations of many packets). Packets that can be routed around such a congested area should do that, or they increase the congestion even more. Such an algorithm was studied by Busch et al. at STOC 2000.

Also one-to-many routing (multicasting) was considered, where a source needs to send the same packet to many destinations. In one-to-many routing, packets can be duplicated whenever needed.

Nobody knows the topology of the Internet (and it is certainly not an array or a mesh!). The problem is to find short paths without storing huge routing tables at each node. There are several forms of routing (e.g. compact routing, interval routing) that study the trade-off between routing table size and quality of routing.

Also, researchers started studying the effects of mixing various queuing strategies in one network. This area of research is known as adversarial queuing theory.

And last not least there are several special networks. A mobile ad-hoc network, for example, consists of mobile nodes equipped with a wireless communication device. In such a networks nodes can only communicate when they are within transmission range. Since the network is mobile (dynamic), and since the nodes are considered to be simple, a variety of new problems arise.