Lecture 10/11/12

1 Overview

Last time, we showed that since for a one to one random destination routing problem, that with high probability the congestion is $O(\log n)$ (actually $O(\log N/\log\log N)$), and therefore any "greedy" routing algorithm will certainly route in $O(\log^2 N/\log\log N)$ time. We used "infinite" queues at each node.

We will now show that this could be improved to the $O(\log N)$ time by using the notion of path congestion rather than edge congestion.

2 Scheduling.

In the previous lecture, we showed how to select routes that are good in the sense of congestion. We review whether that is sufficient, whether we can actually schedule packet movements to solve the problem within the congestion bounds. We refer to this as the scheduling problem.

In this section, we will examine the details of the routing a bit more closely. Notice, though that at each switch, two packets could come in, and both could be destined for the output, packets will build up at the switch.

Thus, we may need to store packets at internal switches. For now, we wish to avoid backups, so will allow ourselves to do so.

In particular, let's assume that the packets are routed FIFO, and each switch has sufficient queues so that any packet that is destined for it, can be transmitted. That is, the only reason a packet is not forwarded is that another packet is using the exit edge at the switch where the packet currently is.

Clearly, we should be able to route any permutation in $O(\log^2 N/\log\log N)$ time on a Benes with high probability. You can't possibly wait more than $O(\log N/\log\log N)$ setps at any edge.

But we can get better bound. First, we define a concept called path congestion.

Definition 1 *The path congestion of a path, P in a routing is the sum over the edges in the path of the other packets that use the edge. The path congestion of the routing is the maximum over the paths of their path congestion.*

Now, if the path congestion is P_c, each packet only "sees" P_c packets and thus should only be delayed P_c times. That is, in each step either the packet moves or someone else moves. Thus, the routing time is at most $D + P_c$, where D is the total length of a packet's path, and P_c is the total movement on the packet's path.

What is the path congestion for permutation routing? Well, here we do two steps. We first bound the number of packets that touch the path at all by $O(\log N)$? Then, we need

to say that they don't use the path for too long. That is, for example, each packet that touches the path doesn't stay on the path for say $\Omega(\log N)$ steps.

The, first step is easy. That is, for each packet, X_p we have a $0-1$ variable that indicates whether it hits the path. For packets in the same subbutterfly at level i, the probability is $1/2^i$. There are around 2^i such packets. Thus, one gets the expectation of the sum of these random variables, which is the path congestion, is around $O(\log N)$. We can once again use a Chernoff bound to show that with probability $1 - 1/n^2$ this random variable has a value below $O(\log N)$. This time we use the full power of the Chernoff bound; i.e., the probabilities of hitting the path are different and thus the random variables that we are adding are not independently distributed.

The second step is to show the number of hits is $O(\log N)$. We can assume that we start with at most $K = O(\log N)$ packets hitting the path. For each packet, the probability that it hits the path again is $1/2$. The probability it hits it yet again is $1/2$, and so on. This is an exponentially distributed variable. And the path congestion just the sum of exponentially distributed variables with mean 1. There is surely some bound that bounds the probability that this sum is large. For the sake of completeness, we will do it differently and break it down as follows.

Each of the K random variables can be viewed as a sequence of coin tosses. The random variable is l if the coin tosses come up with heads l times in a row.

Thus, the sequence $HHTHHHTHHHT$ means that packet 0 stayed on the path two times after the first, packet 1 hit three times, packet 3 three times. The length of the sequence is the total length is the path congestion.

For the number of "heads" in the sequence to be k (we know the number of tails is K), an upperbound on the probability is the sum over choices of $\{l_1, \ldots, l_K\}$ where the $\sum_i l_i = k$ (that is packet i hits the path l_i times) of

$$\pi_{v \in \{l_1, l_2, \ldots, l_K\}} (1/2)^v = \left(\frac{1}{2}\right)^k$$

The number of terms choices of $\{l_1, \ldots, l_K\}$ is $\binom{k+K}{K} \leq ((k+K)e/K)^K$. Let's assume that $k = cK$ for some value of c. An upper bound on the probability is

$$((c+1)e)^K (1/2)^{cK}.$$

It can be shown that for $K = O(\log N)$, that c can be a constant and the probability will be less than $1/N^2$

This is also $O(\log N)$ with high probability.

REMARK 1 As someone asked in some class. This is a bit strange in that the second depends on the first. A bit more formally, we first bound the probability of event A, which is the event that more than say $4 \log N$ packets see the path at all. Then, we bound the event, B, that the total congestion on the path is too high given that not too many packets hit the path. (I.e. the packets don't overstay their welcome.) That is, we bound $Pr[B|\overline{A}]$. Thus, using $Pr[B] \leq Pr[A] + Pr[B|\overline{A}]$, we have a bound on $Pr[B]$.

Thus, the total number of hits is at most $O(\log N)$ with high probabiliy, and thus greedy routing on the butterfly routes in time $O(\log N)$.

3 Loaded Butterflies

.

The analysis of the previous lecture is optimal to within a constant factor since one cannot route in fewer than $\log N$ steps, and it routes in $O(\log N)$ steps. But is it efficient? Could one route more packets in this time? Well, the average congestion on each edge is $O(1)$, so most edges are idle over most of the $\Theta(\log n)$ time units. There should perhaps have been some pipelining. Can we use them to route other packets.

For example, what if $\log N$ packets started at each input and were destined for each output? Can this be accomplished in $O(\log N)$ time? This, we will show (to borrow a phrase from Yoda; a wrinkly guy from an old movie.)

In the previous section, we showed how to select routes that are good in the sense of congestion. Here we will view the scheduling problem more generatlly. In particular, we assume that we have a set of messages routed in a network (perhaps the Benes network perhaps another) that gives congestion at most C. Furthermore, we assume that the network has a special structure (like the Benes network) where packets flow monotonically between levels of a network. For example, in the Benes network each packet flows from one level to the next, the depth D of the network is $\log N$. We call these levelled networks.

We will use an algorithm that assigns random priorities to each packet. That is, if two (or more) packets need to use the same edge during a timestep, the highest priority packet is sent. Note this allows packets to jump over other packets in queues.

First, think of a packet's progress. It either moves or is delayed. This time it is delayed by a higher priority packet. So, if the packet is delayed L times it must have met up with L higher priority packets. We will write down this thing as a "delay sequence" which we define as follows with respect to a priority function $r(\cdot)$ on the packets.

Definition 2 *A priority respecting delay sequence for a packet routing is a sequence of pairs (p_i, e_i) where p_i is a packet and e_i is an edge in the network where packet p_i uses edge e_i and either*

1. *$p_i = p_{i+1}$ and $e_i \neq e_{i+1}$ and e_{i+1}, e_i form a path in the network, or*
2. *$p_i \neq p_{i+1}$ and $e_i = e_{i+1}$, and $r(p_i) > r(p_{i+1})$.*

Lemma 1
If a packet p suffers a delay of L in a priority based routing run, one can construct a priority respecting delay sequence of length $D + L$ where D is the depth of the network.

We begin with the packet, (p, e), where e is the last edge that the packet traversed. Now, if the packet was delayed in the previous step, that would mean that it waited to use edge e. This would mean some other higher priority packet p' used e. Thus, we add (p', e) to the delay sequence. Otherwise, the packet must have used another edge e' in the previous step, and we add (p, e') to the delay sequence. We continue to extend the delay sequence in this manner. Note, we change the packet that we follow backwards. When we switch edges we go backwards in the network. The ith iteration corresponds to the $T - i$th step of operation of the network. Since $T = D + L$, we can continue for this many steps.

Notice, further that the number of times we switch packets is L and the number of times we switch edges is D.

Now, we can prove the following theorem.

THEOREM 2
For any levelled network of bounded degree and of depth D, the priority routing algorithm routes in time $O(C + D + \log N)$ time.

PROOF:

We will show that the probability of a very long, i.e., $>> C + D + \log N$, delay sequence is very low. Here, the probability space is defined by the choice of the random priorities.

We assume that a packet arrives at time t, and note that there will be a delay sequence of length t.

The probability that a delay sequence of length t is sorted is at most

$$\frac{1}{(t-D)!}.$$

For convenience, we define $L = (t - D)$.

Here, we assume that the priorities are chosen in a large enough universe so that there are no ties.

By choosing packets from a range larger than N^4. priorities from before one can conclude that each packet has a different priority, with probability at least $1/N^2$.

The calculation then follows by noting that the number of packets is $L = t - D$ and they must be sorted. Then we observe, that there are $L!$ possible orders for the priorities of these packets. (Formally, we are computing a probability conditioned on the event that no 2 packets have the same priority.)

What is the number of delay sequences of length t? Well each delay sequence consists of a path of length D in the network, a choice of whether to switch packets or edges at each point, and a selection of packets on $L = t - D$ edges on the path. The number of paths is at most

$$Nd^D$$

, where d is the maximum in-degree in the network, N starting points, and d choices of edges to switch in each step. The number of choices for switching packets is

$$\binom{t}{D}.$$

The number of choices of packets at each place where packets are chosen is C. Overall this amounts to C^L possibilities. All together the total number of delay sequences is bounded by

$$Nd^D \binom{t}{D} C^L.$$

By the union bound, the probability that there is any sorted delay sequence given a randomly chosen set of priorities at at most

$$Nd^D \binom{t}{D} C^L \frac{1}{L!}.$$

It can be shown that for some $L = \Omega(D + C + \log N)$, this probability is at most $1/N$. (You can assume the degree d of the network is 2 for simplicity.)

□

This result was first proven by Alelunias and Upfal. This presentation was based on Ranade's algorithm, which was invented to solve the problem for routing with bounded queues.

We proceed to use it thusly. (Is that even a word?)

4 Bounded Queues

What about bounded queues? How do we solve this problem? Well, Ranade invented the random priority algorithm to solve precisely this problem. There is one additional condition he enforces.

Ranade's sorting condition. The packets that use a switch must emerge in sorted order in terms of the priorities.

This seems a global condition. That is, before one can send a message out a link, one needs to know if anyone with better priority is ever going to use the switch!! How can we implement this?

Ranade used "ghost packets", whenever one sends a packet of priority γ out one edge, one should send a "ghost packet" of priority γ out all the other edges. We modify the switch design to only send one packet out from a switch.

Consider the first level. We start by sending out a packet on one edge, and we send ghost packets out the other. Moreover, when we are done sending packets, we send end of stream packet (EOS) along all links, forever. Note, that we send a packet on every edge in every period.

For internal nodes, it gets a bound on the priorities from the incoming packets. By induction, each input has packets with worse and worse priorities over time. Thus, one sends a packet from one input, when the other inputs verify that no better packet it to come. That is, one can always send the best packet at all the inputs and maintain the condition that the packets are sent in sorted order.

Here, we use the idea of the levelled network of depth D. After D steps everyone has at least a ghost packet on the inputs.

Finally, there is a bounded buffer, say of size 5 at each edge, so a switch may not be allowed to send a packet at all!

So, to review, the algorithm at each node is to send the best priority packet from all its inputs, *if* there is room at the next switch. Moreover, it sends a ghost copy of the packet on all edges.

Notice, that one never needs to have a ghost packet being internal to a queue. If a packet arrives after it, it has worse priority and can simply destroy the ghost packet. Moreover, if it does wait it simply informs the previous packet in the queue of its lower bound. (Thus, each packet carries a lower bound on successive pakcets and a priority.)

Now, we can state a generalization of Ranade's theorem.

THEOREM 3
For any bounded degree, levelled network of depth D, one can route any set of paths with congestion C, in time $O(C + D + \log N)$ using switches with constant sized buffers.

Now, the concept of the delay sequence needs to be modified. A packet p can be delayed by a packet that uses a node that it wants to, or by the fact that a queue is full ahead of it, or by the fact that it is waiting to get a signal that its priority is high enough. Note that the ghost packet, never waits.

Now, we will define a definition of delay sequence that corresponds to this situation.

DEFINITION 3 *Given a set of packets with priority function $r(\cdot)$ a sorted delay q-bounded buffer delay sequence is a sequence $\{(p_1, S_1, s_1), (p_2, S_i, s_2), \ldots, \}$ where p_i uses switch s_i and S_i is an ordered set of packets that use s_i,*

- *$p_i = p_{i+1}$ and $s_i \neq s_{i+1}$ and s_{i+1} is an immediate predecessor of s_i in the network, and S_i is empty,*
- *$p_i \neq p_{i+1}$ and $s_i \neq s_{i+1}$ and s_{i+1} is a predecessor of s_i in the network, and S_i is empty,*
- *$p_i \neq p_{i+1}$, $s_i = s_{i+1}$, and S_i is empty, and $r(p_i) < r(p_{i+1})$.*
- *$p_i \neq p_{i+1}$, s_{i+1} is an (immediate) successor of edge of s_i, and $S_i = p^1, p^2.., p^q$, with $p^q = p_{i+1}$ and $r(p_i) < r(p^1) \cdots < r(p^q)$. We call this a forward step.*

Furthermore, if p appears either as p_i or in S_i then p is a real packet whose path uses switch i.

The fourth condition is new and models waiting for a packet in a queue up ahead.

We now consider the notion of the lag of a packet. That is at time t if a packet is at level ℓ in the levelled network, we define the lag to be $t - \ell$.

LEMMA 4
If a packet has lag L, one can construct a bounded queue, sorted delay sequence with $(L - f) + f(q)$ distint packets appearing either at p_i or in S_i, where f is the number of forward edges in the delay sequence in a network with queuesize q.

PROOF:

Question 1: Prove this lemma.

□

Again, we count the number of possible delay sequences and then multiply by the probability that each is sorted to upper bound the probability that any packet has lag L.

The number of delay sequences is at most ??.

Question 2: Give an upper bound on the number of delay sequences. Use the fact that we trace out a path by going backward or by going forward, and the fact that when we move backward we get no new packets.

Again, the probability that it is sorted is ??

Question 3: What is the probability that it is sorted? Use the fact that there are $L - f + (q)f$ is the number of packets and there is one sorted order.

Question 4: (Optional, do not turn in.) Show that there is some value of $L = O(C + D + \log N)$ which can ensure that the probability is less than $1/N^2$. (You may only do the case where $C >> D$ and for q to be a sufficiently large constant.)