Algorithmic aspects of communication networks	**Lecture 7**
OR 739 Spring 1997	**February 5, 1997**

Many of the algorithms we've discussed so far are provably optimal, in the sense that one can prove (sometimes unpleasant) theorems which show that the algorithm comes within a constant factor of reaching the lower bound which an algorithm can achieve for that problem.

Today, we prove such a theorem for the problem of congestion control. This proof is not like an NP-hardness proof; the problem is not that the problem is provably difficult, but that it's impossible to do better because of the on-line nature of the (s,t) pairs' arrival.

For review, here's the setting for this problem, which is slightly different from the case we discussed in class, in that the graph is directed:

- Given a directed graph $G = (V, E)$.
- (s,t) pairs arrive on-line.
- Must always route every pair via a path in G.
- Objective function: minimize the maximum number of paths using an edge. That is, if $f(e)$ = the number of paths using edge e, then we want to minimize

$$\lambda = \max_{e \in E} f(e)$$

What are possible limitations on our algorithms which ensure that they might be reasonable?

- Reasonable amount of work: The algorithm can't solve NP-hard problems.
- Reasonable amount of knowledge: The algorithm can't know what the future pairs may be.

Algorithms are compared for competitiveness against

$$\lambda^* = \min \max_{e \in E} f(e),$$

where λ^* is the off-line optimal value of λ.

This is the theorem we are to prove:

Theorem 1 *For any on-line algorithm A which cannot predict the future, there exists a graph, G, and a finite sequence of (s,t) pairs such that*

$$\lambda_A \geq (c \lg n) \cdot \lambda^*.$$

In fact, we find a sequence where $\lambda^* = 1$ and $\lambda_A \geq c \lg n$.

Here's the general idea: Start with a perfectly matched bipartite graph, and connect each edge in the graph to a sink and a source, as in Figure 1.

This is a graph with nodes $\{t, v_1, w_1, v_2, w_2, \ldots, v_{2^k}, w_{2^k}, s_{00}\}$, and with edges from every $v_i \rightarrow w_i$, from $s \rightarrow v_i$, and from $w_i \rightarrow t$, for $i = 1, \ldots, 2^k$.

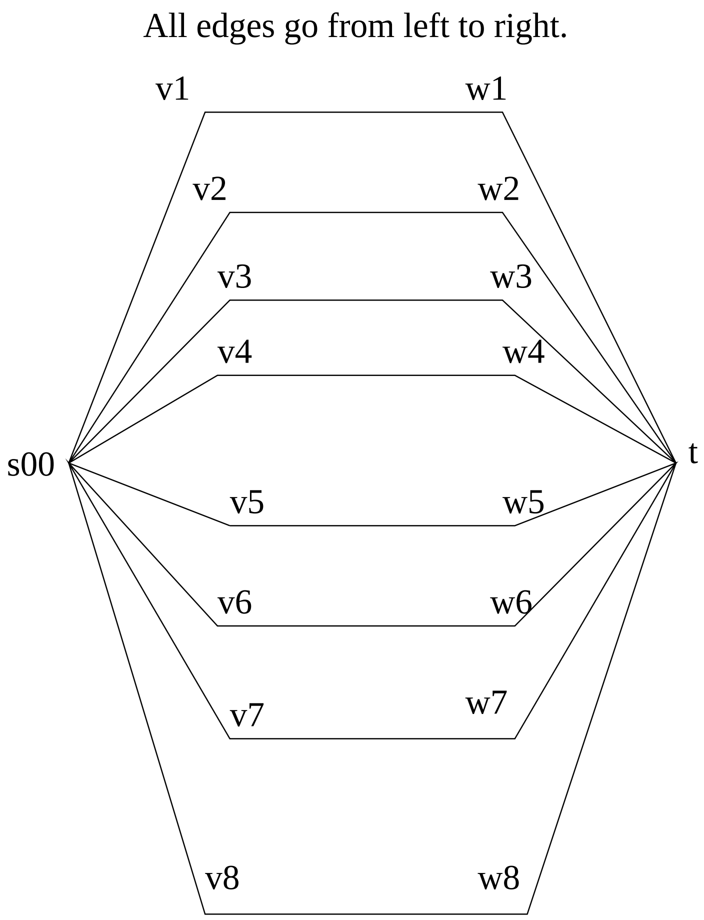

Figure 1: First attempt at the counterexample graph

We now create an $(s - t)$ pair sequence that makes the network congested. To do this, link (s_{00}, t), 2^{k-1} times, which uses some subset of the $v_i \to w_i$ paths. We now want to force the network to only reuse the edges from the $v_i \to w_i$ that have already been used.

Suppose, in other words, that the edges which were used were only those on the "upper" half of the network. Then we'd like to penalize the algorithm for not using the bottom of the network by making further connections use the the top half of the network only.

However, that's unrealistic, since the entire graph has to be pre-constructed; we're trying to start off with a set graph and only choose the pairs maliciously.

What we can do, though, is to choose which half (top or bottom) of the graph was more used, and make sure that future routings only use that half, which will still guarantee congestion. Suppose, then, that we have two more sources, s_{10} and s_{11}, where $s_{10} \to v_i$ for $i = 1, \ldots, 2^{k-1}$ and $s_{11} \to w_i$ for $i = (2^{k-1}+1), \ldots, 2^k$. See which half was more used by the first 2^{k-1} connections, and if it was the top, route 2^{k-2} connections from s_{10} to t; if it was the bottom, route 2^{k-2} connections from s_{11} to t.

And then recurse, for increasingly smaller sets pointed to by the s_{ij}, always choosing the most congested segment.

To be more formal, there are k levels of the source nodes, s_{ij}, where $i = 0 \ldots k - 1$ and $j = 0 \ldots 2^i - 1$. The source node s_{ij} is connected to v_r iff $r \in j(2^{k-i}) + 1, \ldots, (j+1)2^{k-i}$. Thus every source s_{ij} is joined to 2^{k-i} of the v_r. The graph for when $k = 3$ is in Figure 2.

We route the following pairs:

$j = 0$.

For $i = 0, \ldots, k - 1$:

- We request (s_{ij}, t), 2^{k-i-1} times.
- We then select, of the source nodes $s_{(i+1)j}$ at the next level, the one whose neighbors are already most congested. That is, the j where

$$\sum_{r=j(2^{k-i})+1}^{(j+1)2^{k-i}} f(v_r, w_r)$$

  is maximal, and set j equal to that index.

The algorithm is as malicious an adversary as possible; it watches the algorithm and requests paths exactly where there's lots of congestion.

We simply need to prove these two facts, and the proof will be complete:

Lemma 1 $\lambda^* = 1$, *and* $\lambda_A \geq c \lg n$ *(in fact,* $\lambda_A \geq k/2$*).*

Then, the theorem is immediate.

Proof: If we had actually routed all the pairs in exactly the reverse sequence, then every time an (s, t) pair appeared, there'd be at least one uncongested path for each pair to directly go across, so the congestion would stay constant as one. That is, let i go backwards from $k - 1$ to 0, and:

- When $i = k - 1$, there is exactly one path; route it.
- When $i = k - 2$, there's 2 requests. However, there's 4 connections connecting the source node to t, so there is still room for disjoint paths to connect them.

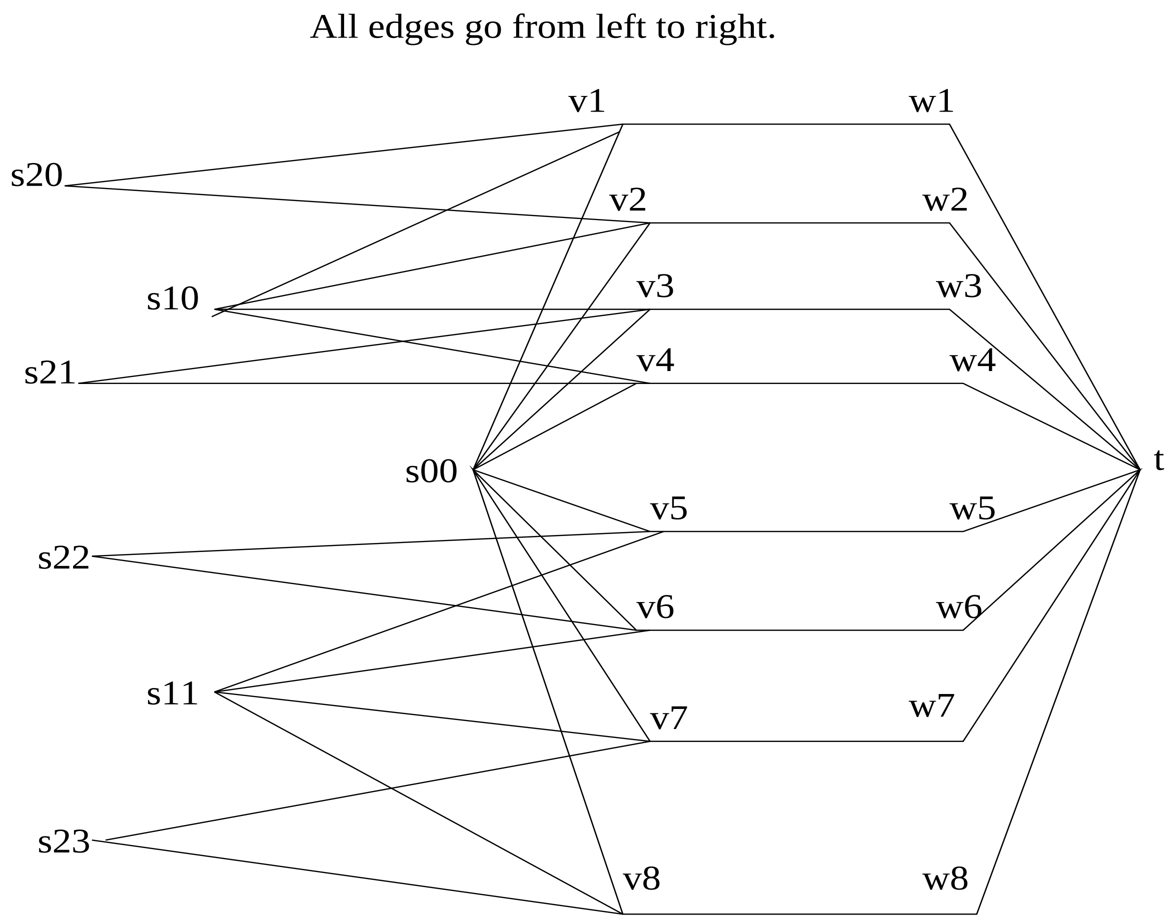

Figure 2: The counterexample graph

And, in general, a greedy routing will give each request its own path, and if we had used these routings on the original sequence (in forward order), we would still have had $\lambda^* = 1$. ■

Now, we prove that $\lambda_A \geq k/2$, via induction on i.

Assume that it's true that before iteration i that the average congestion of neighbors of the selected source node s_{ij} is at least $i/2$. This is clearly true for the base case when $i = 0$.

Induction step: Before the new pairs are routed, there's

$$\sum_{v_r \in N} f(v_r, w_r) \geq i/2 \cdot 2^{k-i}$$

pairs routed across the (v_r, w_r) edges connected to the neigbors N of s_{ij}.

After the new pairs are routed, there's at least

$$i/2 \cdot 2^{k-i} + 2^{k-i-1}$$

edges across it, so now we have at least

$$i/2 \cdot 2^{k-i} + \frac{1}{2} 2^{k-i} = \frac{1}{2}(i+1)2^{k-i}$$

paths across it. Now we select the most congested half of these edges to pick the next j, so we certainly still have average congestion $\frac{i+1}{2}$. So the induction holds and the lemma is true.

Actually, it's clear that before the last step (when $i = k - 1$), the maximum congestion is at least $\frac{k-1}{2}$, and in the last step, we definitely choose the most congested (v_r, w_r) edge and route across it, increasing the maximum congestion by 1, not $1/2$. So we actually have that

$$\lambda_A \geq k/2 + 1/2,$$

and the proof of the lemma is complete. ■

So we see that $\lambda^* = 1$ and $\lambda_A \geq k/2$. To finish the theorem, we need to relate the number of nodes to k. But G has 1 sink node t, 2^k nodes v_r, 2^k nodes w_r, and $1 + 2 + 4 + \cdots + 2^{k-1}$ nodes of the form s_{ij}. This totals $n = 3 \cdot 2^k$ nodes.

So $\lg n = k + \lg 3$, and we see that $\lambda_A \geq c \lg n \cdot \lambda^*$, as desired. ■

Having finished this proof, some notes seem in order:

- This proof would have been much easier if we were in the unrealistic setting where we could construct the bad graph G on line along with the sequence of bad (s, t) pairs.

- The adversary we're facing is unrealistically powerful, in that it can see exactly where we've routed (or where we're going to route) the pairs. What happens if the algorithm being run is randomized? If we allow the adversary to see the algorithm, but require it to output the graph and (s, t) pairs before the algorithm runs, can we still guarantee bad congestion?

  It turns out that the answer is yes; we can compute which regions will have high expected congestion and target those areas with our (s, t) pairs; this results in a lower bound on the expectation of λ_A which is the same as the one proved.

- This proof is in the special case of disjoint paths, but does generalize to the case where the minimum congestion is much bigger than 1; we just request each (s,t) pair a factor of L times more frequently. The exact same arguments as before show that

$$\lambda^* \leq L,$$

but

$$\lambda_A \geq cL \lg n,$$

giving the same competitiveness ratio as before.