Designing Networks for Selfish Users is Hard

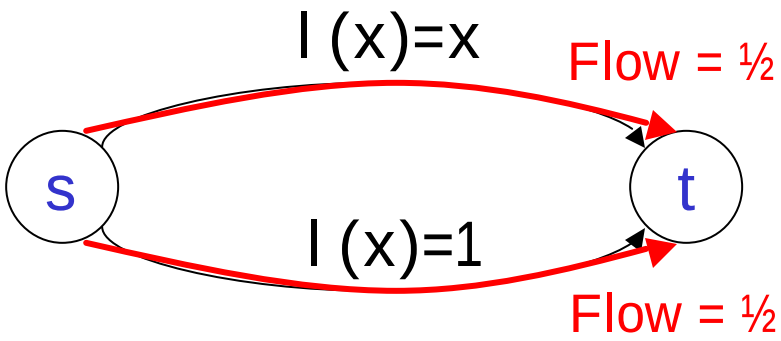
Tim Roughgarden
Cornell University

Traffic in Congested Networks

The Model:

- A directed graph $G = (V,E)$
- A source s and a sink t
- A rate r of traffic from s to t
- For each edge e, a latency function $l_e(\bullet)$

Example: $(r=1)$



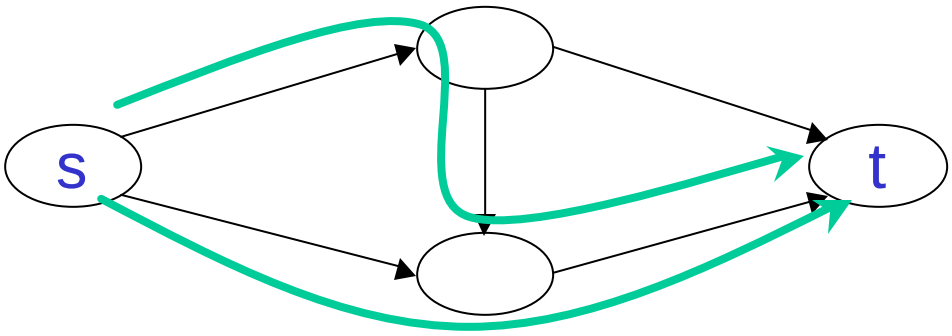
Flow Paths

Traffic and Flows:

- f_P = amount of traffic routed on s-t path P
- flow vector f ⇔ routing of traffic

Path Latency:

- latency of path P w.r.t. flow f = sum of latencies of edges on P

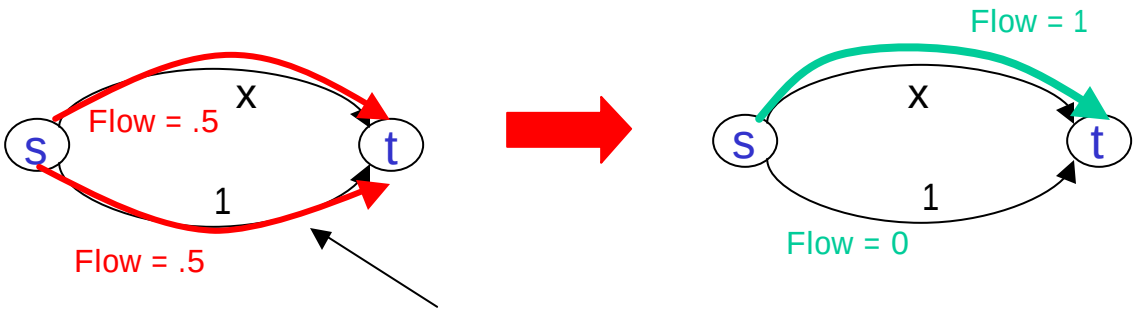


Flows as Selfish Traffic

- flow = routes of many noncooperative agents
- Examples:
  - cars in a highway system
  - packets in a network
- agents are selfish
  - will seek out path with minimum-possible latency

Flows at Nash Equilibrium

Def: A flow is at Nash equilibrium (is a Nash flow) if no agent can improve its latency by changing its path



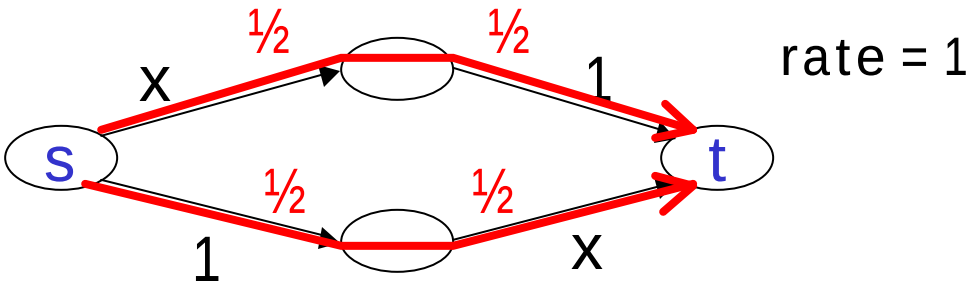
Assumption: edge latency functions are continuous, nondecreasing

Lemma: f is a Nash flow $\Leftrightarrow$ all flow on minimum-latency paths (w.r.t. f)

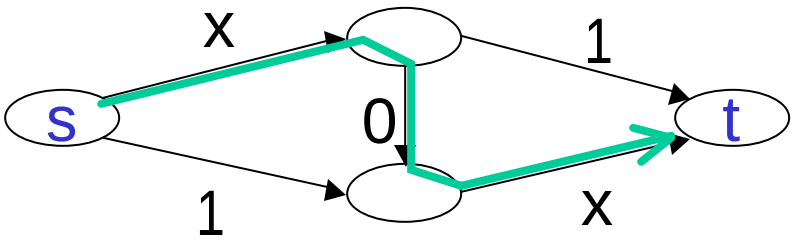
Fact: have existence, uniqueness

Braess's Paradox

Better network, worse Nash flow:



latency = 1.5



latency = 2

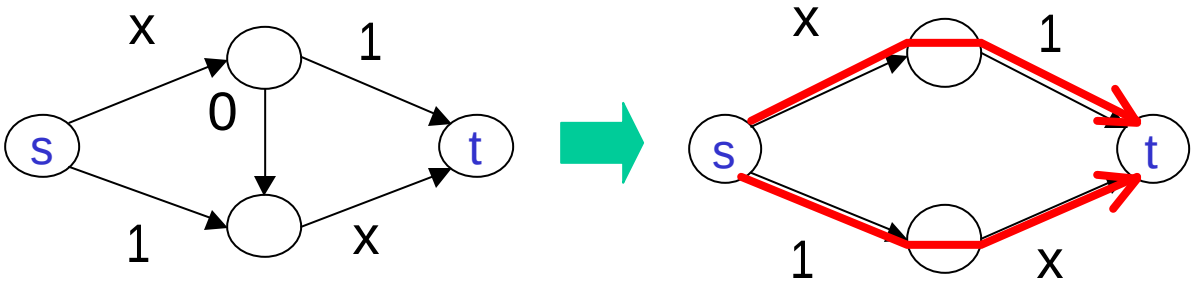
All traffic experiences more latency!

- example from [Braess 68]

Designing Networks for Selfish Users

The Problem:

- given network $G = (V,E,l)$
- find subnetwork minimizing latency experienced by all selfish users in a Nash flow



Input

Optimal Solution

Previous Work

- [Braess 68], [Murchland 70]
  - network design problem defined
- [Steinberg/Zangwill 83], etc.
  - When is the trivial algorithm optimal?

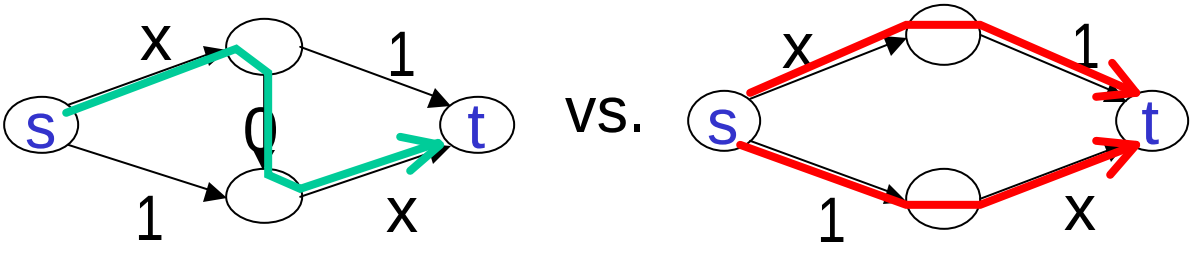
Def: The trivial algorithm is to build the entire network.

Guarantees for the Trivial Algorithm

Fact: The trivial algorithm is a $|V|/2$-approximation algorithm.

Def: a linear latency function is of the form $l_e(x)=a_e x+b_e$

Fact: For linear latency fns, the trivial algorithm is a $4/3$-approximation algorithm.



Designing Networks for Selfish Users is Hard

Thm 1: For $\varepsilon > 0$, no $(|V|/2 - \varepsilon)$-approximation algorithm exists (unless P=NP).

Thm 2: For linear latency functions, no $(4/3 - \varepsilon)$-approx algorithm exists (unless P=NP).

Cor: in general, Braess's Paradox cannot be detected efficiently.

Linear Latency

Thm: [Roughgarden/Tardos 2000]
In a network with linear latency fns:

average latency of Nash flow = 4/3 × average latency of any other flow

Corollary: the trivial algorithm has approximation ratio 4/3.

Hardness: reduction from finding disjoint paths in a digraph

General Latency - An Easy Upper Bound?

Proof approach from linear case:

We hope: In a network with general latency fns:

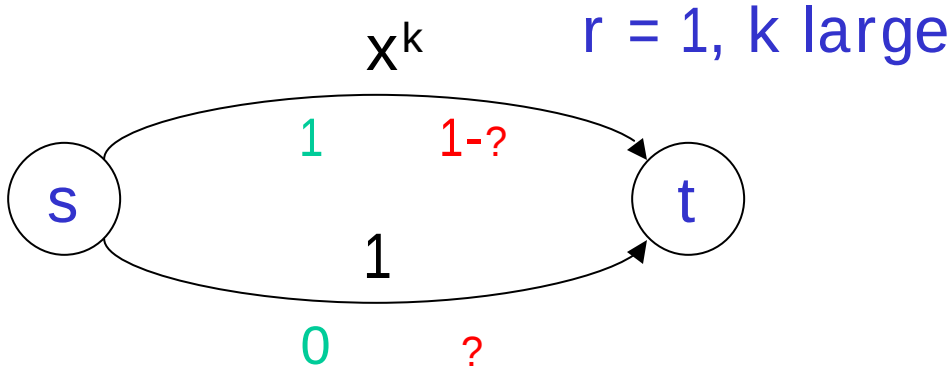
average latency of Nash flow $= \beta \times$ average latency of any other flow

[perhaps with $\beta = \beta(|V|,|E|)$]

Then: the trivial algorithm has approximation ratio β.

Difficulties

Problem: a Nash flow can incur arbitrarily more latency than other flows, even if |V|=|E|=2:

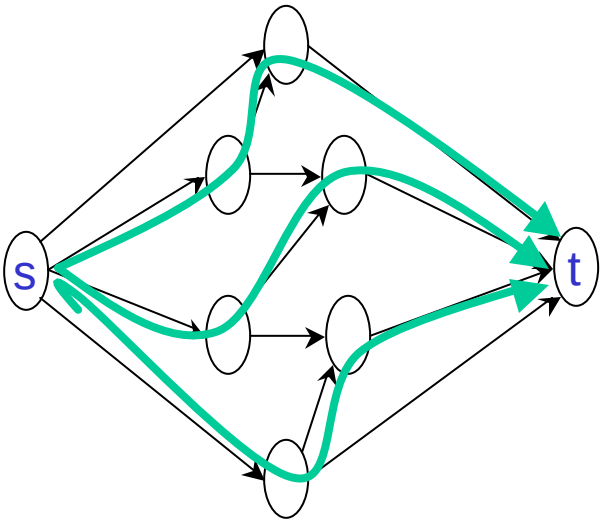


Nash has average latency 1, but average latency ≈ 0 is possible

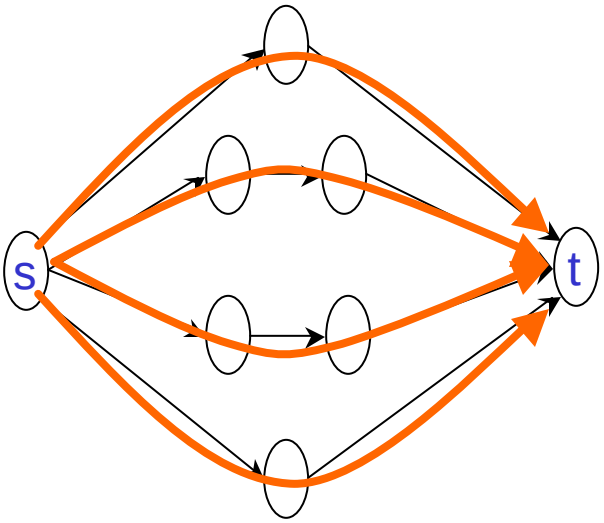
Nevertheless: The trivial algorithm is a |V|/2-approximation algorithm.

Lower Bound for General Latency

Bad Networks:



Nash in whole graph
common latency = 4



Nash in opt subgraph
common latency = 1

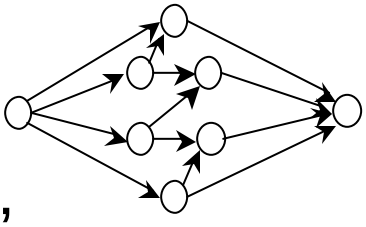
$\Rightarrow$ provides lower bound of $|V|/2$ for performance of trivial algorithm

Toward a Hardness Result

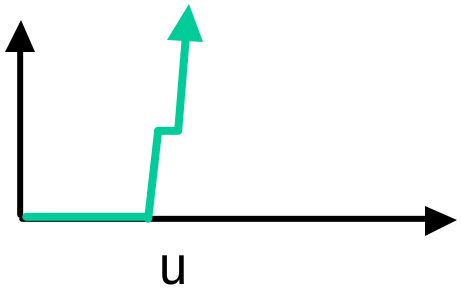
Thm: guarantee of |V|/2 is best possible, unless P=NP.

Notes on Proof:

- reduction from Partition
- start with networks like:



- replace each "cross-edge" with parallel edges representing Partition instance
- use latency functions to encode "capacities":



Extensions

Remark: hardness of network design not particular to general, linear latency fns

E.g.: polynomials with degree = k:

- trivial algorithm achieves performance guarantee $O(k/\log k)$
- hardness: $O(k/\log k)$