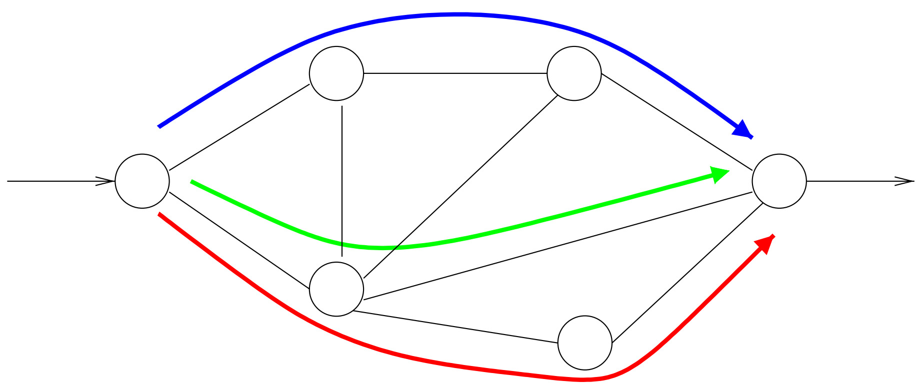
Routing

Problem: Given more than one path from source to destination, which one to take?



Features:

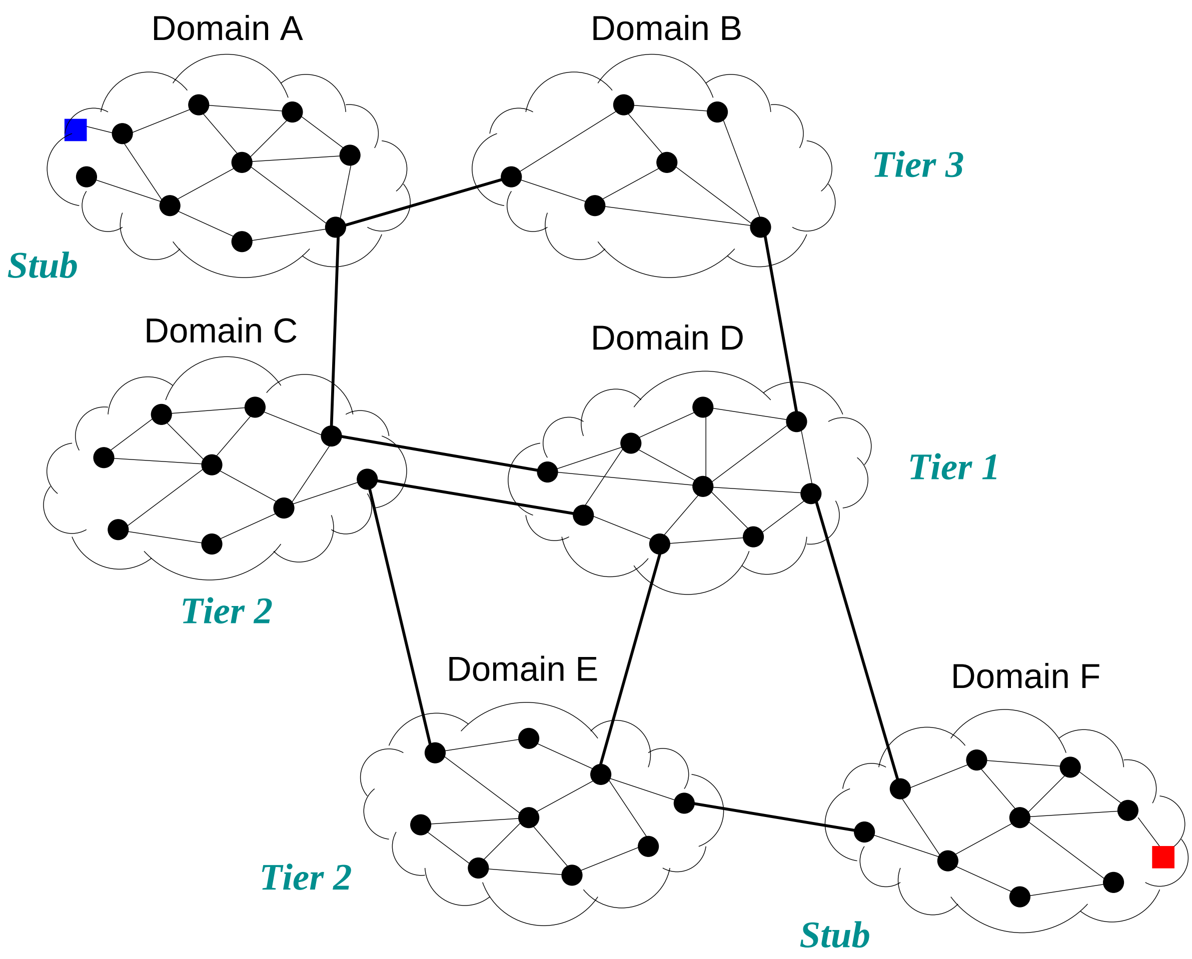
- Architecture
- Algorithms
- Implementation
- Performance

Architecture

Hierarchical routing:

$\longrightarrow$ Internet: intra-domain vs. inter-domain routing

$\longrightarrow$ separate decision making



Granularity of routing network:

- Router
- Domain: autonomous system (AS)

  → 16 bit identifier: ASN (e.g., Purdue 17)

  → assigned by IANA along with IP prefix block (CIDR)

Network topology (i.e., map/connectivity):

- Router graph

  → node: router

  → edge: physical link between two routers

- AS graph

  → node: AS

  → edge: physical link between 2 or more border routers

  → sometimes at exchange point or network

Router type:

- border router

  $\longrightarrow$ includes access router (to stub customer)

- backbone router

AS type:

- stub AS

  $\longrightarrow$ no forwarding

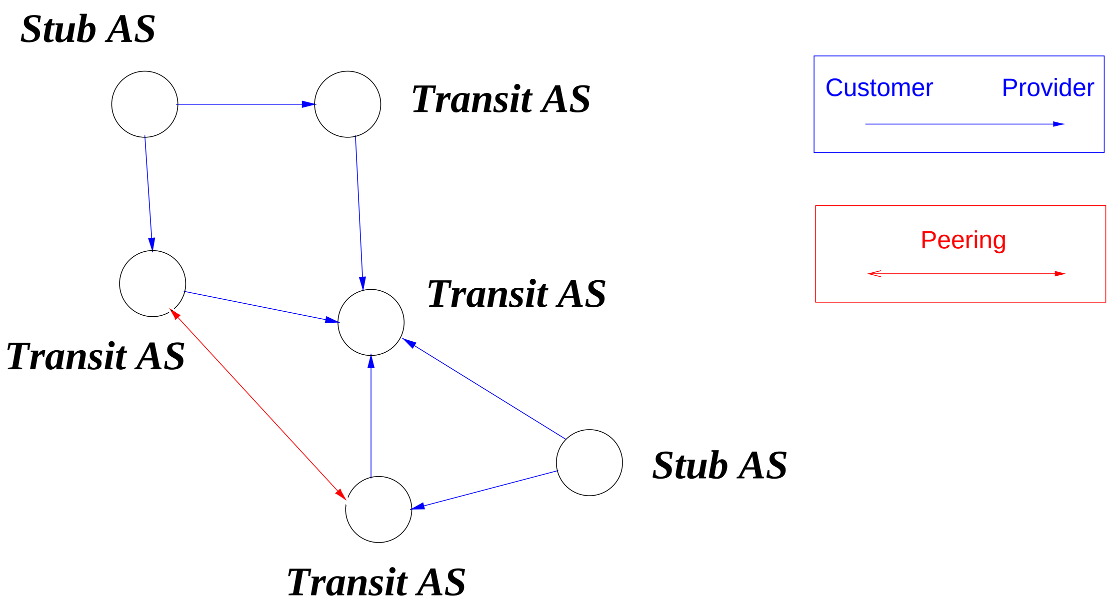
  $\longrightarrow$ may be multi-homed (more than one provider)

- transit AS

  $\longrightarrow$ tier-1: global reachability & no provider above

  $\longrightarrow$ tier-2 or tier-3: providers above

AS graph:



Inter-AS relationship: bilateral

- customer-provider: customer subscribes BW from provider

  $\longrightarrow$ most common

  $\longrightarrow$ customer can reach provider's reachable IP space

- peering:

  $\longrightarrow$ only the peer's IP address and below

  $\longrightarrow$ the peer's provider's address space: invisible

Common peering:

- among tier-1 providers

  $\longrightarrow$ ensures global reachability

- among tier-2 providers

  $\longrightarrow$ economic factors

- among stubs

  $\longrightarrow$ economic factors

  $\longrightarrow$ e.g., content provider & access ("eyeball") provider

Route or path: criteria of goodness

- Hop count
- Delay

  $\rightarrow$ composed of three parts
- Bandwidth

  $\rightarrow$ available bandwidth
- Loss rate

Composition of goodness metric:

$\longrightarrow$ quality of end-to-end path

- Additive: hop count, delay
- Min: bandwidth
- Multiplicative: loss rate

Goodness of routing:

$\longrightarrow$ assume N users or sessions

$\longrightarrow$ suppose path metric is delay

- System optimal routing

  $\rightarrow$ choose paths to minimize $\sum_{i=1}^{N} D_i$

- User optimal routing

  $\rightarrow$ each user i chooses path to minimize D_i

  $\rightarrow$ selfish actions

Pros/cons:

- System optimal routing:
  - Good: minimizes delay for the system as a whole
  - Bad: complex and difficult to scale
- User optimal routing:
  - Good: simple
  - Bad: may not make efficient use of resources
    $\rightarrow$ utilization

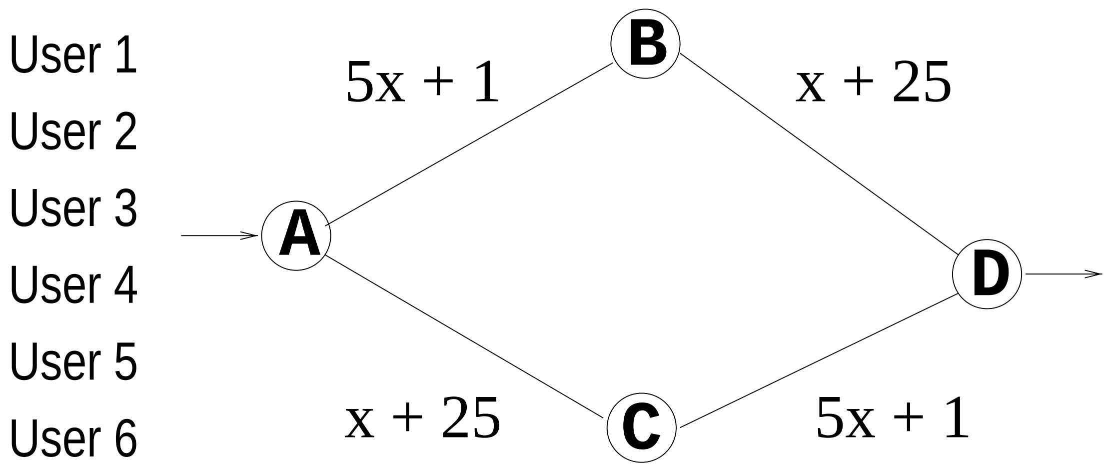
Some pitfalls of user optimal routing:

$\longrightarrow$ stemming from selfishness

- Fluttering or ping pong effect
- Braess paradox

Braess paradox example:

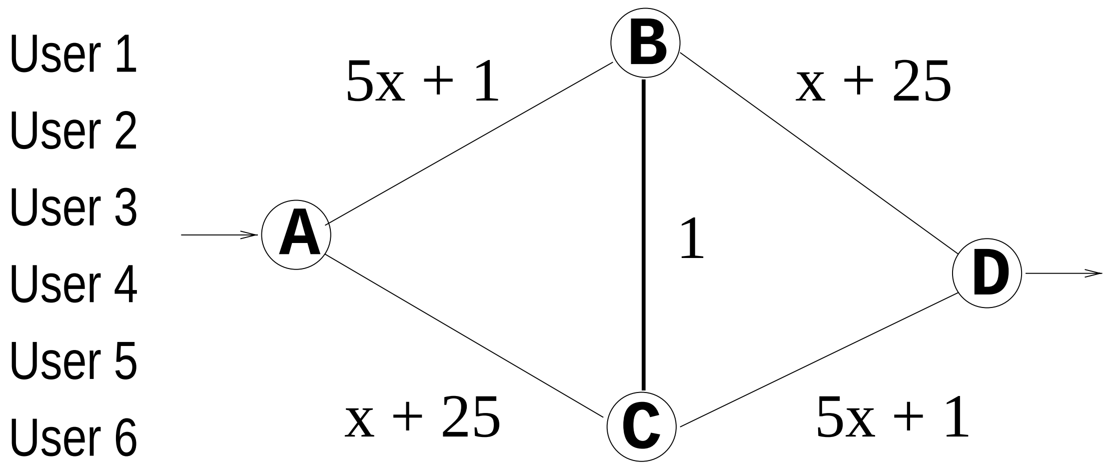
- 6 users sending 1 Mbps traffic
- Delay on shared link increases with traffic volume x
- Users make routing decisions one after the other



- 3 users will take $A \rightarrow B \rightarrow D$
- 3 users will take $A \rightarrow C \rightarrow D$
- total delay per user: $(5 \cdot 3 + 1) + (3 + 25) = 44$

Resource provisioning:

$\longrightarrow$ high bandwidth link is added between B and C



- User 1: $A \rightarrow B \rightarrow C \rightarrow D$ (13)
- User 2: $A \rightarrow B \rightarrow C \rightarrow D$ (23)
- User 3: $A \rightarrow B \rightarrow C \rightarrow D$ (33)
- User 4: $A \rightarrow B \rightarrow C \rightarrow D$ (43)
- User 5: $A \rightarrow B \rightarrow D$ (52)
- User 6: $A \rightarrow C \rightarrow D$ (52)

Adding extra link should improve things, but has the opposite effect

$\longrightarrow$ high-speed link induces load imbalance

$\longrightarrow$ paradox possible due to selfishness

$\longrightarrow$ D. Braess (1969)

$\longrightarrow$ cannot arise in system optimal routing

$\longrightarrow$ i.e., cooperative routing

Modus operandi of the Internet: user optimal routing

$\longrightarrow$ simplicity wins the day

Algorithms

Find short, in particular, shortest paths from source to destination.

Key observation on shortest paths:

- Assume p is a shortest path from S to D

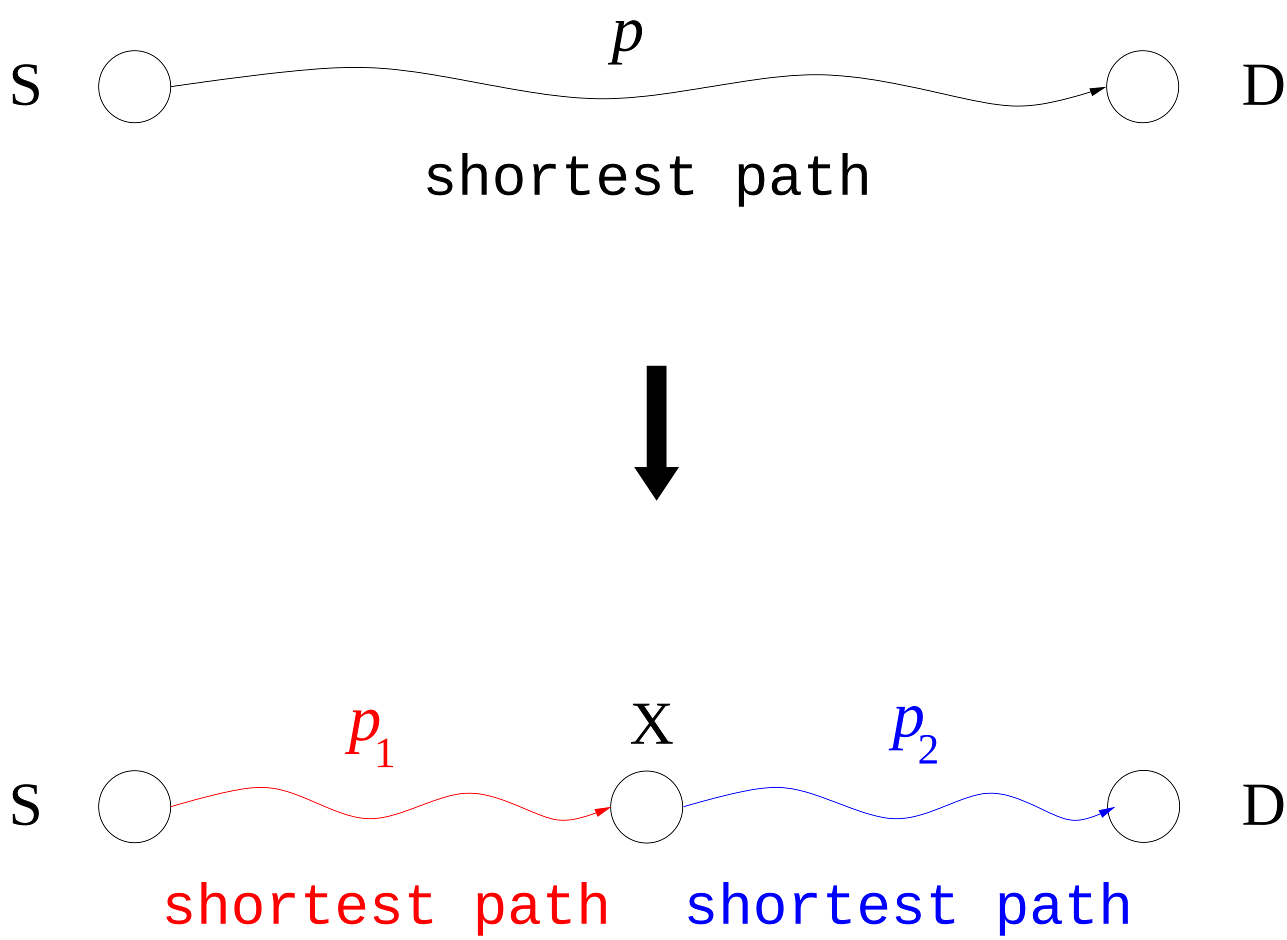
  $\rightarrow S \overset{p}{\leadsto} D$

- Pick any intermediate node X on the path
- Consider the two segments p_1 and p_2

  $\rightarrow S \overset{p_1}{\leadsto} X \overset{p_2}{\leadsto} D$

- The path p_1 from S to X is a shortest path, and so is the path p_2 from X to D

Illustration:



$\longrightarrow$ reverse implication need not hold

$\longrightarrow$ suggests algorithm for finding shortest path

Procedure: Grow a routing tree $\mathcal{T}$ rooted at source S

$\longrightarrow$ initially $\mathcal{T}$ only contains S

1. Find a node X with shortest path from S

   $\rightarrow$ there may be more than one such node

   $\rightarrow$ add X (and path $S \overset{p}{\rightsquigarrow} X$) to routing tree $\mathcal{T}$

2. Find node $Y \notin \mathcal{T}$ with shortest path from S

   $\rightarrow$ update existing paths if going through Y is shorter

   $\rightarrow$ i.e., $\min\{d(S, Z), d(S, Y) + \ell(Y, Z)\}$

   $\rightarrow$ need only check for $Z \notin \mathcal{T}$

3. Repeat step two until no more nodes left to add

Observations:

$\longrightarrow$ once node is added, it's final (no backtracking)

$\longrightarrow$ builds minimum spanning tree routed at S

$\longrightarrow$ Dijkstra's algorithm

Remarks:

- Running time: $O(n^2)$ time complexity

  $\longrightarrow$ n: number of nodes

- If heap is used: $O(|E| \log |V|)$

  $\longrightarrow$ good for sparse graphs: $|E| \ll n^2$

  $\longrightarrow$ e.g., if linear: $O(n \log n)$

- Can also be run "backwards"

  $\longrightarrow$ start from destination D and go to all sources

  $\longrightarrow$ a variant used in inter-domain routing

  $\longrightarrow$ forward version: used in intra-domain routing

- Source S requires global link distance knowledge

  $\longrightarrow$ centralized algorithm (center: source S)

  $\longrightarrow$ every router runs Dijkstra with itself as source

- Internet protocol implementation

  $\longrightarrow$ OSPF (Open Shortest Path First)

  $\longrightarrow$ link state algorithm

  $\longrightarrow$ broadcast protocol

- Minimum spanning tree routed at S:

  $\longrightarrow$ multicasting: multicast tree

  $\longrightarrow$ standardized but not implemented on Internet

Distributed/decentralized shortest path algorithm:

$\longrightarrow$ Bellman-Ford algorithm

$\longrightarrow$ based on shortest path decomposition property

Key procedure:

- Each node X maintains current shortest distance to all other nodes

  $\rightarrow$ a distance vector

- Each node advertises to neighbors its current best distance estimates

  $\rightarrow$ i.e., neighbors exchange distance vectors

- Node X, upon receiving an update from neighbor Y, performs update: for all Z

$$d(X, Z) \leftarrow \min\{\, d(X, Z),\ d(Y, Z) + \ell(X, Y)\,\}$$

. . . same criterion as Dijkstra's algorithm

Remarks:

- Running time: $O(n^3)$
- Each source or router only talks to neighbors

  → local interaction

  → no need to send update if no change

  → if change, entire distance vector must be sent
- Knows shortest distance, but not path

  → just the next hop is known
- Elegant but additional issues compared to Dijkstra's algorithm

  → e.g., stability
- Internet protocol implementation

  → RIP (Routing Information Protocol)

QoS routing:

Given two or more performance metrics—e.g., delay and bandwidth—find path with delay less than target delay D (e.g., 100 ms) and bandwidth greater than target bandwidth B (e.g., 1.5 Mbps)

$\longrightarrow$ from shortest path to best QoS path

$\longrightarrow$ multi-dimensional QoS metric

$\longrightarrow$ other: jitter, hop count, etc.

How to find best QoS path that satisfies all requirements?

Brute-force

- Enumerate all possible paths
- Rank them

How many paths are there:

- If there are n nodes, there can be up to

$$\frac{n(n-1)}{2}$$

  undirected links

- Hence, from source S there can be up to

$$(n-1)\,(n-2)\cdots 3\,2\,1 \;=\; (n-1)!$$

  paths

- By Stirling's formula

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

  $\longrightarrow$ superexponential

  $\longrightarrow$ too many for brute-force

Is there a more clever or better algorithm?

$\longrightarrow$ as of Nov. 15, 2004: unknown

$\longrightarrow$ specifically: QoS routing is NP-complete

$\longrightarrow$ strong evidence there may not exist good algorithm

In networking: several problems turn out to be NP-complete

$\longrightarrow$ e.g., scheduling, control, ...

$\longrightarrow$ "P = NP" problem

$\longrightarrow$ one of the hardest problems in science ever

Doesn't matter too much for QoS routing

$\longrightarrow$ little demand for very good algorithm

$\longrightarrow$ roughly ok is fine

$\longrightarrow$ intra-domain: short paths

$\longrightarrow$ inter-domain: other factors ("policy")

Policy routing:

$\longrightarrow$ policy is not precisely defined

$\longrightarrow$ almost anything goes

Routing criteria include

- Performance

  $\rightarrow$ e.g., short paths

- Trust

  $\rightarrow$ what in the world is "trust"?

- Economics

  $\rightarrow$ pricing

  $\rightarrow$ flexibility through multiple providers

- Politics, social issues, etc.

$\longrightarrow$ no good understanding of "policy" to date

$\longrightarrow$ anecdotal