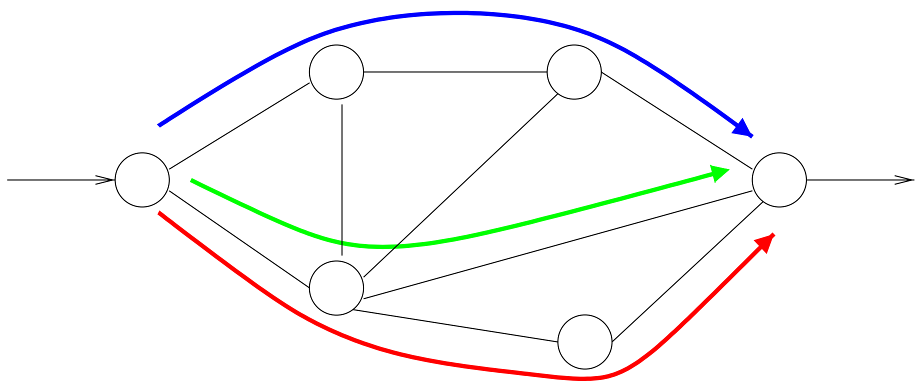
Routing

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



Features:

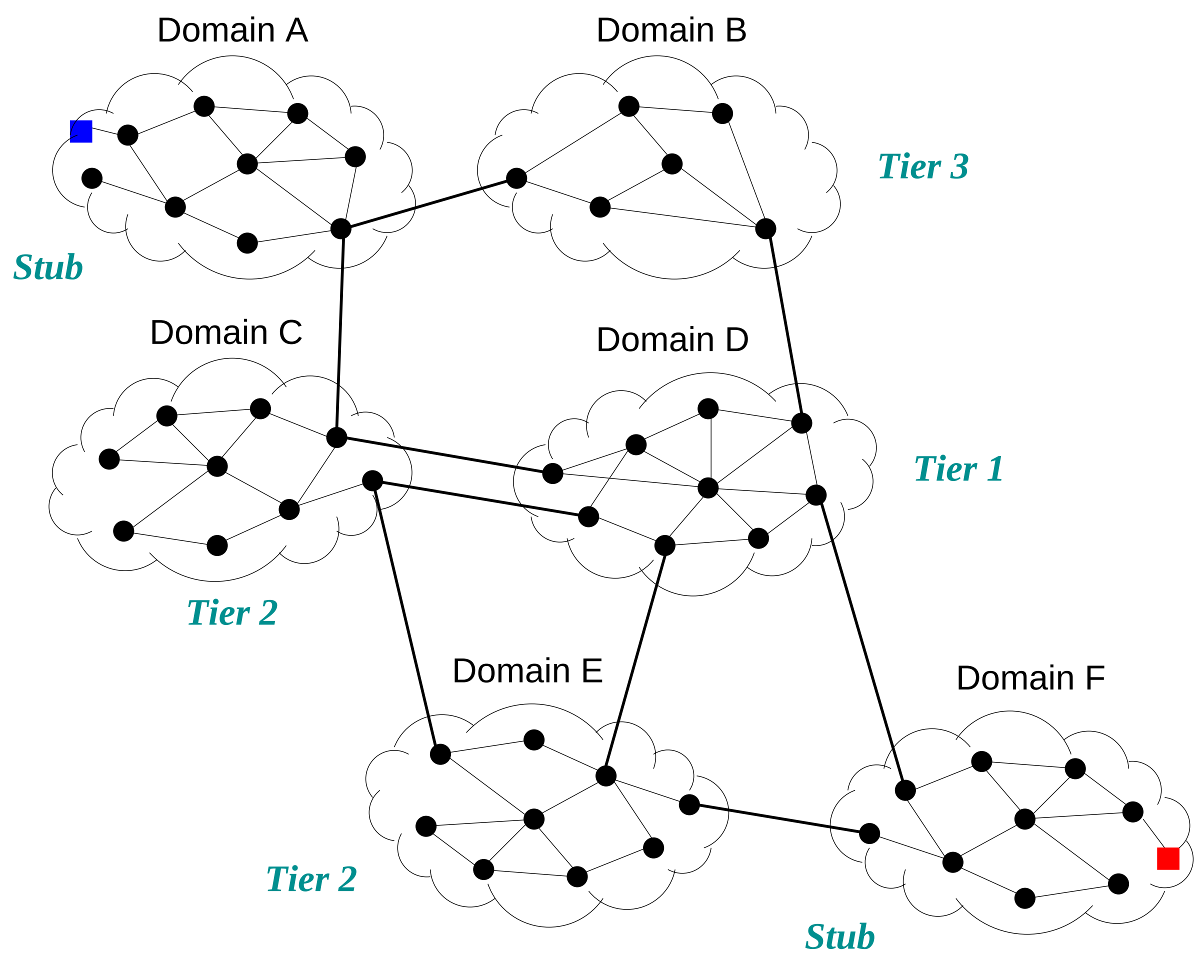
- Architecture
- Algorithms
- Implementation
- Performance

Architecture

Hierarchical routing:

⟶ Internet: intra-domain vs. inter-domain routing

⟶ separate decision making



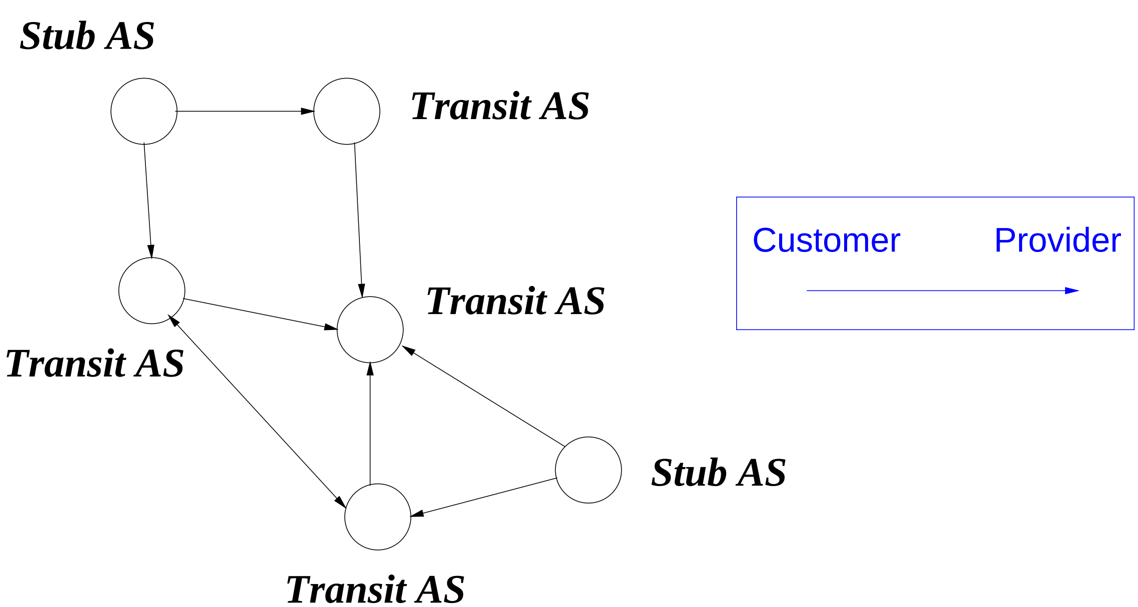
Granularity

- Router
- Domain: autonomous system (AS)

  $\longrightarrow$ 16 bit identifier

Network representation

- Router graph
- AS graph



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

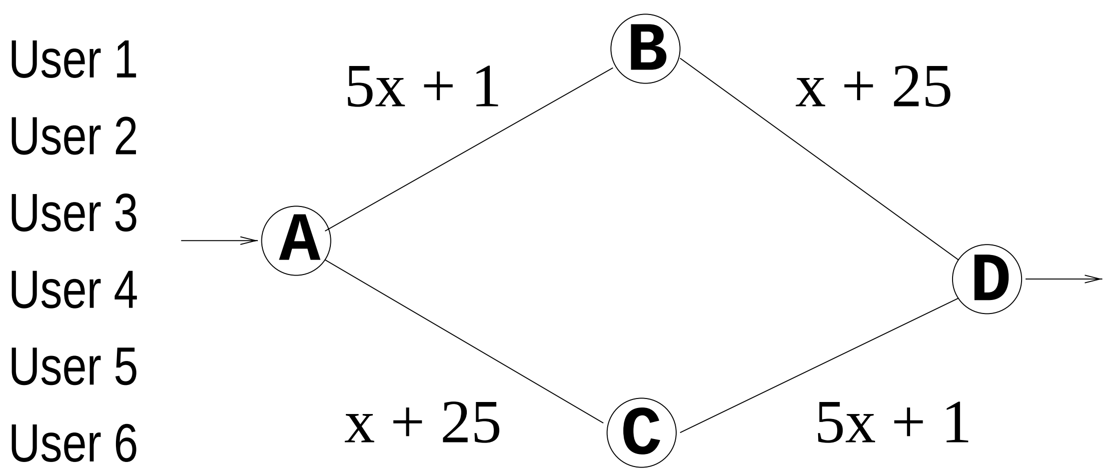
Some pitfalls of user optimal routing:

$\longrightarrow$ stemming from selfishness

- Fluttering or ping pong effect
- Braess paradox

Braess paradox example:

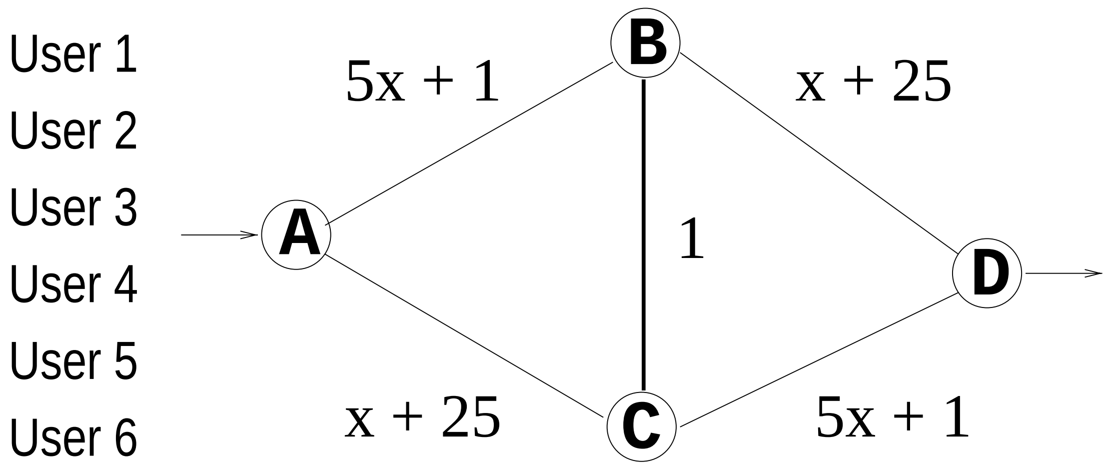
- 6 users sending 1 Mbps traffic
- Delay on shared link increases with traffic volume x
- Users make routing decisions one after the other



- 3 users will take $A \to B \to D$
- 3 users will take $A \to C \to D$
- 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

$\longrightarrow$ paradox possible due to selfishness

$\longrightarrow$ D. Braess (1969)

$\longrightarrow$ cannot arise in system optimal routing

$\longrightarrow$ i.e., cooperative routing

Adam Smith: let the "invisible hand" do its work

$\longrightarrow$ doesn't always lead to best outcome

$\longrightarrow$ capitalism vs. communism

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

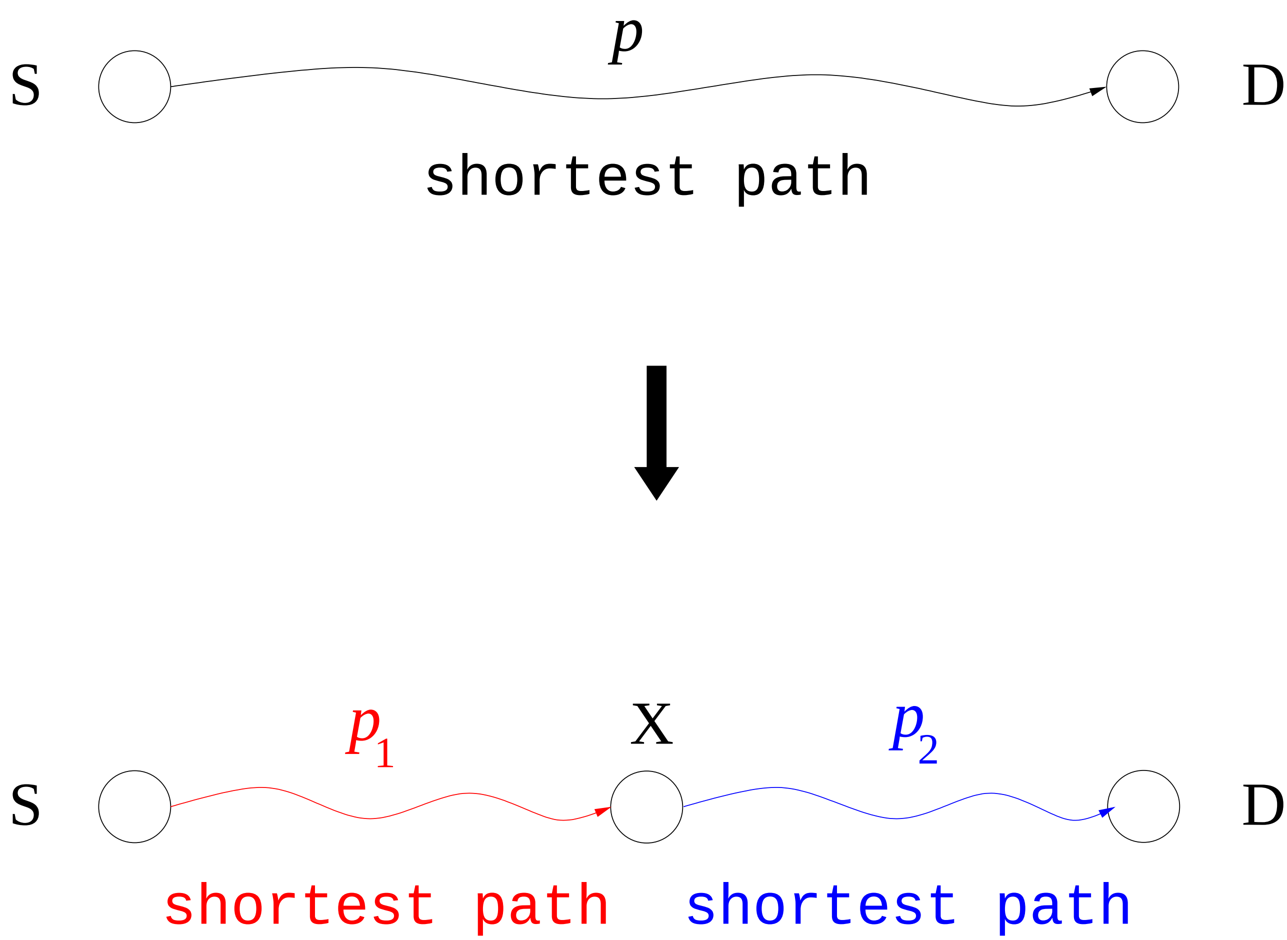
  $\rightarrow S \overset{p}{\leadsto} D$

- 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 with shortest path from $\mathcal{T}$

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

   $\rightarrow$ uses shortest path decomposition property

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

- Can also be run "backwards"

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

  $\longrightarrow$ single-destination/all-source shortest path

- 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

  → OSPF (Open Shortest Path First)

  → link state algorithm

- Minimum spanning tree routed at S:

  → multicasting: multicast tree

  → 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
- A 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

  $\longrightarrow$ local interaction

  $\longrightarrow$ no need to send update if no change

  $\longrightarrow$ if change, entire distance vector must be sent
- Knows shortest distance, but not path

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

  $\longrightarrow$ e.g., stability
- Internet protocol implementation

  $\longrightarrow$ RIP (Routing Information Protocol)