Routing Algorithms

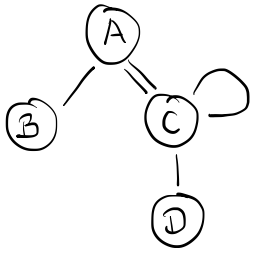
1. Graph theory fundamentals

Graph theory provides the necessary abstractions for talking about networks and routing, so first a brief introduction.

- A graph is a tuple $G = (V, E)$ where
  - V is a set, the vertices (or nodes in the network context)
  - (u, v) for $u, v \in V$ denotes an edge (or link in the network context)
  - E is a multiset of edges (multiple distinct edges are allowed)
- A graph may be directed (then edge $(u,v) \neq (v,u)$ if $u \neq v$)
  or undirected (then $(u,v) = (v,u)$ denote the same edge)

Examples (in the usual planar representation):

(a)

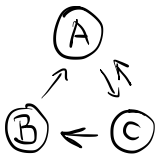


undirected

$V = \{A, B, C, D\}$

$E = \{(A,B), (A,C), (A,C), (C,C), (C,D)\}$

(b)



directed

$V = \{A, B, C\}$

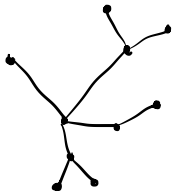
$E = \{(B,A), (C,B), (A,C), (C,A)\}$

- A path from $u \in V$ to $v \in V$ is a tuple

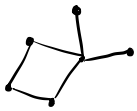
$$(u, w_1), (w_1, w_2), (w_2, w_3), \ldots, (w_{n-1}, v) \qquad w_1, \ldots, w_{n-1} \in V$$

must match

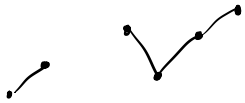
- A path is simple if $w_1, \dots, w_{n-1}$ are distinct ("No self-intersections")
- A cycle is a path with $v = v$.
- G is connected if there is a path from v to v for all $v, v \in V$.
- A tree is a connected graph without cycles.



tree



not a tree



not a tree

- A rooted tree is a tree where one vertex is marked as the "root".

Note: File systems (without links) are modelled as rooted trees (often without saying "rooted"), networks

2. Link-state routing

Step ①: Each node ("router") constructs a complete view of the network (graph)

Step ②: Routing table is computed independently on each node

For ①: Each node ("router") periodically does the following:

(a) Discover its neighbors (e.g. probing each of its wire connections)

(b) Determine link costs (delays, bandwidth, and/or other criteria)

(c) Flood the network with this information by sending a link-state packet (LSP) containing

- ID of self
- cost of link to each nearest neighbor

- sequence number
- "time-to-live" (TTL)

to each of its neighbors who will compare the sequence number with the last sequence number stored and, if newer,

- store the new LSP
- forward the LSP to each neighbor but the sender
- decrement TTL for each stored LSP, delete LSP if TTL = 0

→ "reliable flooding"

Goal: each node can, from this information, construct the network as a graph. (Include a link only if both neighbors agree.)

For ②: Use shortest-path algorithm (independently on each node)

E.g. Dijkstra's algorithm, here written as running on node u:

- $c(x,y)$ denotes the cost of link (x,y)
- $D(v)$ denotes current estimate of distance (= total cost) from u to v
- N set of all nodes ("routers")
- M set of all nodes for which the final cost is not known

Initialization: $M := N$

$$D(v) := \begin{cases} 0 & \text{if } v = u \\ \infty & \text{if } v \neq u \end{cases}$$

Step: while M has more than one element:

Let $w \in M$ be a node for which $D(w)$ is minimal

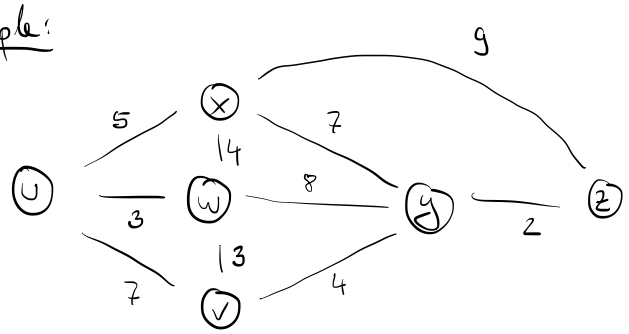
Remove w from M

for $v \in \text{neighbors}(w) \cap M$:

$$D(v) = \min \{D(v),\ D(w) + c(w,v)\}$$

"check if it's cheaper to reach v via w."

Example:



Step	M	D(u)	D(v)	D(w)	D(x)	D(y)	D(z)
0	u,v,w,x,y,z	0	∞	∞	∞	∞	∞
1	v,w,x,y,z	0	7 via v	3 via w	5 via x	∞	∞
2	v,x,y,z	0	6 via w	3	5	11 via w	∞
3	v,y,z	0	6	3	5	11	14 via x
4	y,z	0	6	3	5	10 via w	14
5	z	0	6	3	5	10	12 via w

Routing table for u: u | w | w | x | w | w

Note: u only needs to decide about the first link in the path, the subsequent routing decisions are done by the intermediate routers. That's why u's routing table does not store full paths.

Potential problems with link-state routing:

- Potentially routing loops if view of network is different for different routers
- Oscillations if cost depends on available bandwidth
- Cost of "flooding" if network is large and changes frequently.

3. Distance-vector routing

Goal: avoid construction of full network graph on each node

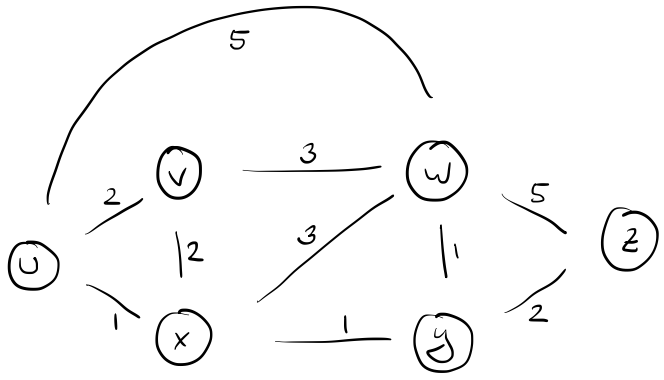
- Write $D_x(y)$ to denote estimate of node $x \in N$ to each $y \in N$
- Each node keeps its own distance vector D_x
- Each node knows its neighbors and the cost $c(x,y)$ to each of its neighbors (by probing, as for link-state routing)
- Each node periodically broadcasts its distance vector to each neighbor
- Upon receiving a distance vector from a neighbor, update own distance via the Bellman-Ford equation

$$D_x(y) = \min_{v \in \text{neighbors}(x)} c(x,v) + D_v(y)$$

Advantage:
- Simple
- No need to be careful about getting coinciding cost estimates of both neighbors

Disadvantage:
- Distance vectors can be huge for big networks (bandwidth)
- "Good news travels fast, bad news travels slow"

Example:



FROM \ TO	u	v	w	x	y	z
u	0	2	5 4 3 (x) (x)	1	∞ 2 (x)	∞ 9 4 (x) (x)
v	2	0	3	2	∞ 3 (x)	∞ 8 5 (w) (x)
w	5 4 3 (x) (y)	3	0	3 2 (y)	1	5 3 (y)
x	1	2	3 2 (y)	0	1	∞ 3 (y)
y	∞ 2 (x)	∞ 3 (x)	1	1	0	2
z	∞ 9 4 (w) (y)	∞ 8 5 (w) (y)	5 3 (y)	∞ 3 (y)	2	0

Resulting routing table (note: nobody has the global view, each router only "sees" its own row):

FROM \ TO	u	v	w	x	y	z
u	u	v	x	x	x	x
v	u	v	w	x	x	x
w	y	v	w	y	y	y
x	u	v	y	x	y	y
y	x	x	w	x	y	z
z	y	y	y	y	y	z