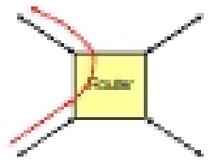
15-441 Computer Networking

Lecture 12: Intra-Domain Routing

RIP (Routing Information Protocol) &
OSPF (Open Shortest Path First)

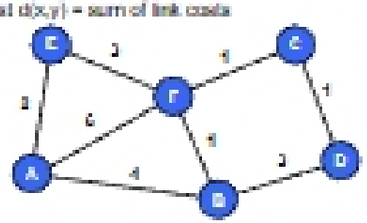
Router Operation

- When Packet Arrives at Router
  - Examine header to determine intended destination
  - Look up in table to determine next hop in path
  - Send packet out appropriate port
- Today's lecture
  - How to generate the routing table



Graph Model

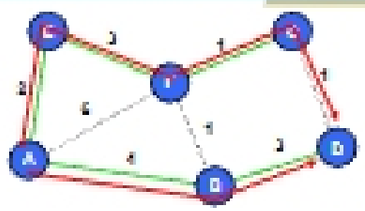
- Represent each router as node
- Direct link between routers represented by edge
  - Symmetric links → undirected graph
- Edge "cost" c(x,y) denotes measure of difficulty of using link
  - delay, $ cost, or congestion level
- Task
  - Determine least cost path from every node to every other node
    - Path cost d(x,y) = sum of link costs



Routes from Node A

Table for A

Dest	Cost	Next Hop
A	0	A
B	4	B
C	6	E
D	7	B
E	3	E
F	5	E



- Properties
  - Some set of shortest paths forms tree
    - Shortest path spanning tree
  - Solution not unique
    - E.g., A-E-F-C-D also has cost 7

Ways to Compute Shortest Paths

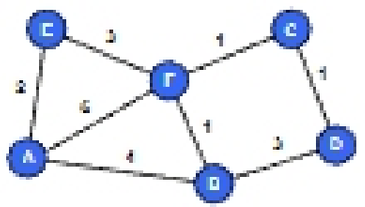
- Centralized
  - Collect graph structure in one place
  - Use standard graph algorithm
  - Disseminate routing tables
- Partially Distributed
  - Every node collects complete graph structure
  - Each computes shortest paths from it
  - Each generates own routing table
  - "Link-state" algorithm
- Fully Distributed
  - No one has copy of graph
  - Nodes construct their own tables iteratively
  - Each sends information about its table to neighbors
  - "Distance-Vector" algorithm

Outline

- Distance Vector
- Link State

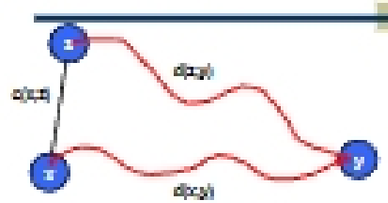
Distance-Vector Method

Initial Table for A		
Dest	Cost	Next Hop
A	0	A
B	4	B
C	∞	–
D	∞	–
E	2	E
F	6	F



- Idea
  - At any time, have cost/next hop of best known path to destination
  - Use cost ∞ when no path known
- Initially
  - Only have entries for directly connected nodes

Distance-Vector Update



- Update(x,y,z)

```
d ← c(x,z) + d(z,y)      # Cost of path from x to y with first hop z
if d < d(x,y)
    # Found better path
    return d,z           # Updated cost / next hop
else
    return d(x,y), nexthop(x,y)   # Existing cost / next hop
```

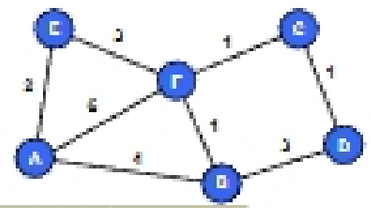
Algorithm

- Bellman-Ford algorithm
- Repeat
  For every node x
    For every neighbor z
      For every destination y
        d(x,y) ← Update(x,y,z)
- Until Converge

Start

Optimum 1-hop paths

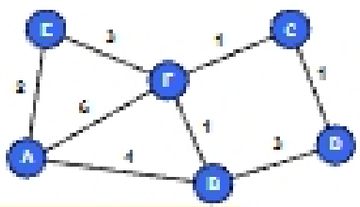
Table for A			Table for B			Table for C			Table for D			Table for E			Table for F		
Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop
A	0	A	A	4	A	A	∞	–	A	∞	–	A	2	A	A	6	A
B	4	B	B	0	B	B	∞	–	B	3	B	B	∞	–	B	1	B
C	∞	–	C	∞	–	C	0	C	C	1	C	C	∞	–	C	1	C
D	∞	–	D	3	D	D	1	D	D	0	D	D	∞	–	D	∞	–
E	2	E	E	∞	–	E	∞	–	E	∞	–	E	0	E	E	3	E
F	6	F	F	1	F	F	1	F	F	3	F	F	3	F	F	0	F



Iteration #1

Optimum 2-hop paths

Table for A			Table for B			Table for C			Table for D			Table for E			Table for F		
Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop
A	0	A	A	4	A	A	7	F	A	7	B	A	2	A	A	5	B
B	4	B	B	0	B	B	2	F	B	3	B	B	4	F	B	1	B
C	7	F	C	2	F	C	0	C	C	1	C	C	4	F	C	1	C
D	7	B	D	3	D	D	1	D	D	0	D	D	∞	–	D	2	C
E	2	E	E	4	F	E	4	F	E	∞	–	E	0	E	E	3	E
F	5	E	F	1	F	F	1	F	F	3	C	F	3	F	F	0	F



Iteration #2

Optimum 3-hop paths

Table for A			Table for B			Table for C			Table for D			Table for E			Table for F		
Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop
A	0	A	A	4	A	A	6	F	A	7	B	A	2	A	A	5	B
B	4	B	B	0	B	B	2	F	B	3	B	B	4	F	B	1	B
C	6	E	C	2	F	C	0	C	C	1	C	C	4	F	C	1	C
D	7	B	D	3	D	D	1	D	D	0	D	D	5	F	D	2	C
E	2	E	E	4	F	E	4	F	E	5	C	E	0	E	E	3	E
F	5	E	F	1	F	F	1	F	F	3	C	F	3	F	F	0	F