La Sapienza
Università degli Studi di Roma
Dipartimento di Informatica e Sistemistica

Computer Networks II

Routing protocols - Overview

Luca Becchetti

Luca.Becchetti@dis.uniroma1.it

A.A. 2009/2010

Goals

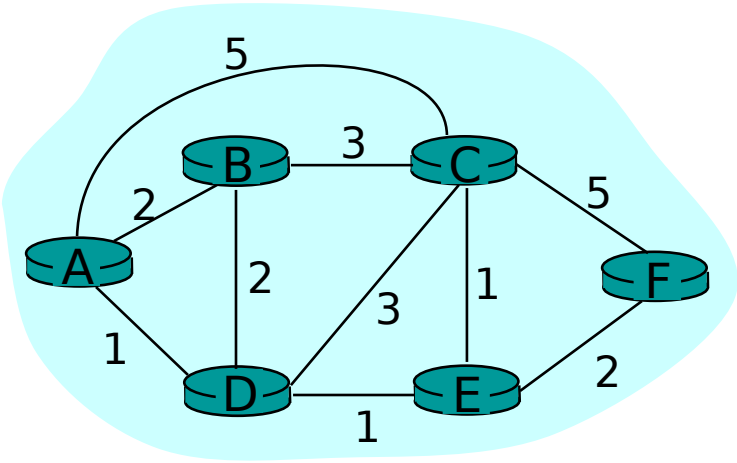
- Describe approaches and give overview of mechanisms adopted
- Describe key aspects of implementations
- Analyze two solutions adopted in practice
  - RIP
  - OSPF (more in detail)

- Some aspects to addressed in more detail by student [see references]

Routing

Routing protocol

Goal: find “good” source-destination path



Graph abstraction:

- Nodes are the routersr
- Links represent subnets
  - Weights on links: delay, cost in €, level of congestio
  - Special case: P2P link

- “Good” route:
  - Usually, shortest path
  - different definitions of shortest path possible

Classification of routing algorithms

Global vs distributed Info

Global:

- All nodes have full info about graph and link costs
- “Link state” algorithms

Distributed:

- Router knows neighbours and costs of links to them
- Iterative and distributed computation: info exchanged with neighbours
- “Distance vector” algorithms

Static vs dynamic

Static:

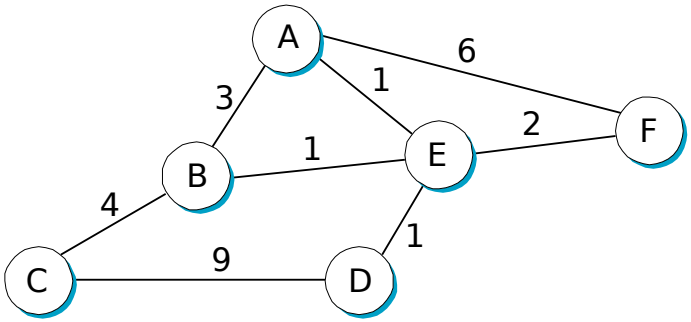
- Routes vary slowly over time

Dynamic:

- Routes change frequently
  - Periodic updates
  - Updates triggered by changes in links' costs

Forwarding (or routing) table

- Cost to reach a given destination
- Next hop (router) to destination
- This is only neccessary info
  - Real tables much more complex

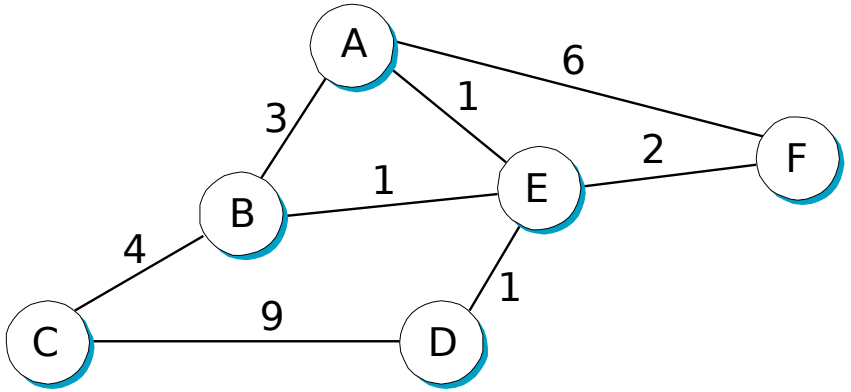


Destinazione	Outgoing link, cost
A	E,2
C	C,4
D	E,2
E	E,1
F	E,3

Routing table at B

Overview

- Forwarding vs Routing
  - forwarding: to select an output port based on destination address and routing table
  - routing: process by which routing table is built



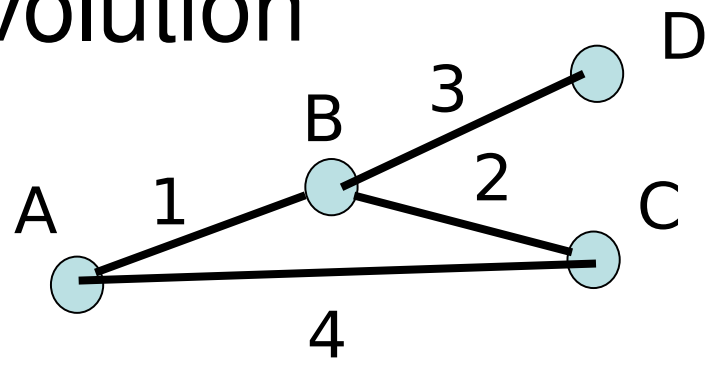
- Factors
  - static: topology
  - dynamic: load
- Summarizing:
  - Routing: find routes to possible destination, build Routing Table
  - Forwarding: forward incoming packet to neighbour router on the basis of packet's destination, using information stored in routing table

Distance Vector Algorithms – overview

- Base algorithm: Bellman –Ford (distributed version)
- Each node maintains a set of triples
  - **`(Destination, Cost, NextHop)`**
- Directly connected neighbors exchange updates
  - periodically (on the order of several seconds)
  - whenever table changes (called *triggered* update)
- Each update is a list of pairs:
  - (**`Destination, Cost)`**
  - If A receives (C, 10) from B (B has to be a neighbour) this means that cost to reach C over A is 10 + cost(A, B)
- Update local table if receive a “better” route
  - smaller cost
  - came from next-hop
- Refresh existing routes; delete them if they time out

Dynamic evolution

- Evolution at node A



Initial table at A

Dest	Cost	NextHop
B	1	B
C	4	C
D	inf	-

Update from B

Dest	Cost
C	2
D	3

entry for A missing

Update from C

Dest	Cost
B	2

entry for A missing

After update from B

Dest	Cost	NextHop
B	1	B
C	3	B
D	4	B

Distance Vector Routing Algorithm

Iterative:

- Iterates until no more info exchanged
- *self-terminating*: no "stop" message exists

Asynchronous:

- Nodes do not perform steps synchronously

Distributed:

- Every node communibates *only* with immediate neighbours

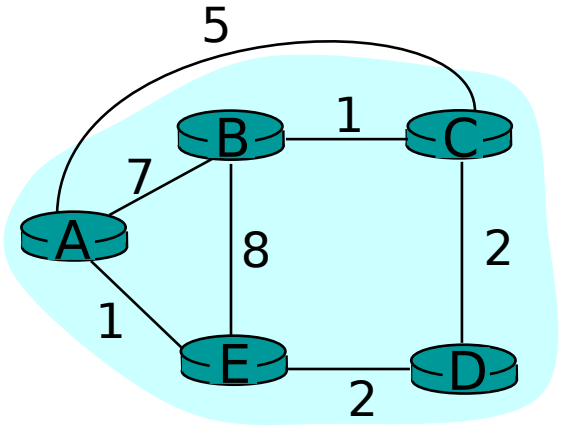
Distance Table (distance table present in every node)

- One entry for each destination
- One column for every node's neighbour
- Example: at node X, for dest. Y through Z:

$$D^{X}(Y,Z) = \text{Distance } \textit{from } X \textit{ to } Y \text{ through } Z$$

$$= c(X,Z) + \min_{w}\{D^{Z}(Y,w)\}$$

Distance table: example



$$D^E(C,D) = c(E,D) + \min_w\{D^D(C,w)\}$$
$$= 2+2 = 4$$

$$D^E(A,D) = c(E,D) + \min_w\{D^D(A,w)\}$$
$$= 2+3 = 5$$ Ciclo!

$$D^E(A,B) = c(E,B) + \min_w\{D^B(A,w)\}$$
$$= 8+6 = 14$$ Ciclo!

Costo per la destinazione via

Destinazione	$D^E()$ A	B	D
A	1	14	5
B	7	8	5
C	6	9	4
D	4	11	2

Distance table gives RT

Cost to destination via

$D^E()$

Destinazione	A	B	D
A	1	14	5
B	7	8	5
C	6	9	4
D	4	11	2

Destinazione	Outgoing link, costo
A	A,1
B	D,5
C	D,4
D	D,2

Distance table → Routing Table

Distance Vector Routing: overview

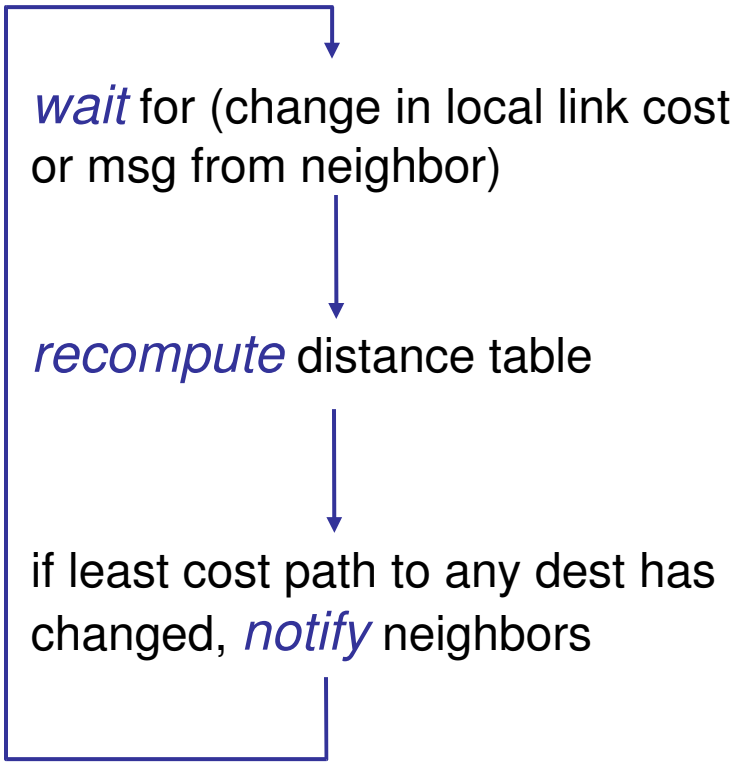
Iterative, asynchronous:
each local iteration caused by:

- local link cost change
- message from neighbor: its least cost path change from neighbor

Distributed:

- each node notifies neighbors *only* when its least cost path to any destination changes
  - neighbors then notify their neighbors if necessary

Each node:



Distance Vector Algorithm

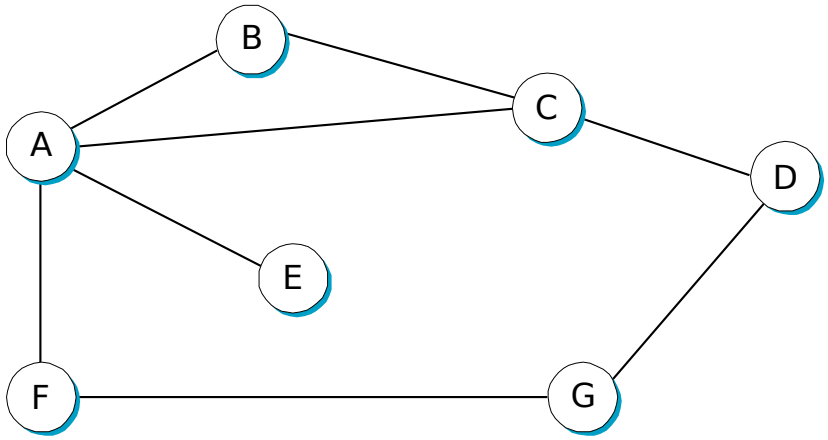
At every node X:

1 Initialization:
2 for every adjacent node v:
3 $D^X(*,v)$ = infinity /* '*' means "for every entry" */
4 $D^X(v,v)$ = c(X,v)
5 for every destination y
6 send $\min_w D^X(y,w)$ to every neigh. /* w varies over all neighbours of X */

Distance Vector Algorithm (cont.):

```
8  loop
9    wait (until link cost to some neighbour V changes
10         or update from neighbour V received)
11
12   if (c(X,V)changes by an amount d)
13     /* update link costs for all destination over X by  d
14     /* note: d may be positive or negative */
15    for every destination y:  D^X(y,V) =  D^X(y,V) + d
16
17   else if (update from V for destination Y)
18     /* shortest path from V to some Y changed  */
19     /* V sends new valu for min_w D^v(Y,w)   */
20     /* let "newval"  be new value  */
21     for   every  y:     D^X(Y,V) = c(X,V) + newval
22
23   if new   value for min_w D^X(Y,w) towards destination Y
24      send new value for min_w D^X(Y,w) to every neigh.
25
26  forever
```

Example - Table at B



Initial

Destination	Cost	NextHop
A	1	A
C	1	C
D	∞	-
E	∞	-
F	∞	-
G	∞	-

Final

Destination	Cost	NextHop
A	1	A
C	1	C
D	2	C
E	2	A
F	2	A
G	3	A

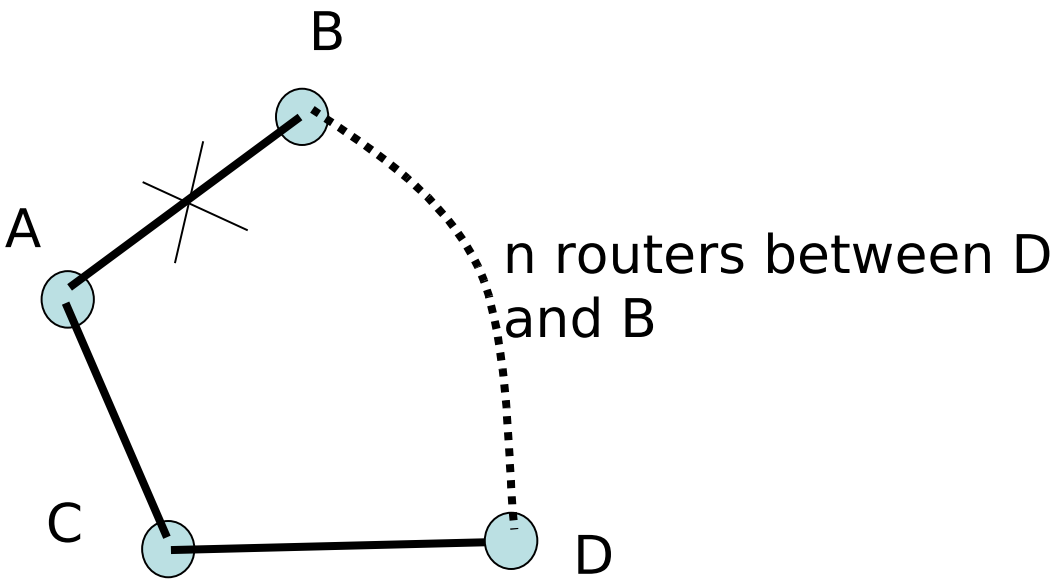
Full example on Peterson and Davie's book

Routing Loops

- Unit costs. Link (A, B) faults, cost from A to B -> inf.

Routing Table at A

Dest	Cost	NextHop
B	1	B
C	1	C
D	2	C
.....		



- A receives periodic update from C before sending own update
- A updates entry for B (entry becomes (B 3 C)) and sends update to C
- Process iterates until cost from C to B through D becomes lower than cost through A

Loop-Breaking Heuristics

- Algorithm converges but this can be slow
- Fixes
  - Set infinity to 16
  - Split horizon
  - Split horizon with poison reverse
    - Do not send update to a neighbour if update caused by that neighbour's previous updates
  - Work often but not always

Link state algorithms

- Base algorithm: Dijkstra
  - Every router needs to know network topology (connectivity and link costs)
- Implementation aspects
  - Information gathering
    - Flooding
  - Define cost metrics
- A link state protocol: OSPF (Open Shortest Path First)
  - Dijkstra's algorithm
  - Protocol for the exchange of link state information
  - Advanced aspects
    - Hierarchical routing
    - Scalability' -> route summarization

Dijkstra's algorithm

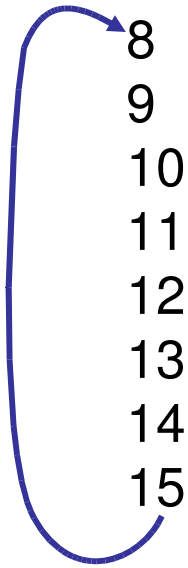
- Every nodes knows topology and link costs
  - Can be achieved using “link state broadcast”
  - All nodes posses same info
- Every node computes shortest paths to all other nodes i
  - Provides routing table for node
- Iterative: after k iterations, shortest paths to k destinations found

Notation

- c(i,j): link cost between i and j.
- D(v): last estimated value for shortest path cost to destination V
- p(v): v's predecessor on shortest path to v
- N: set of nodes for which shortest path cost has been exactly computed

Dijkstra's algorithm/cont.

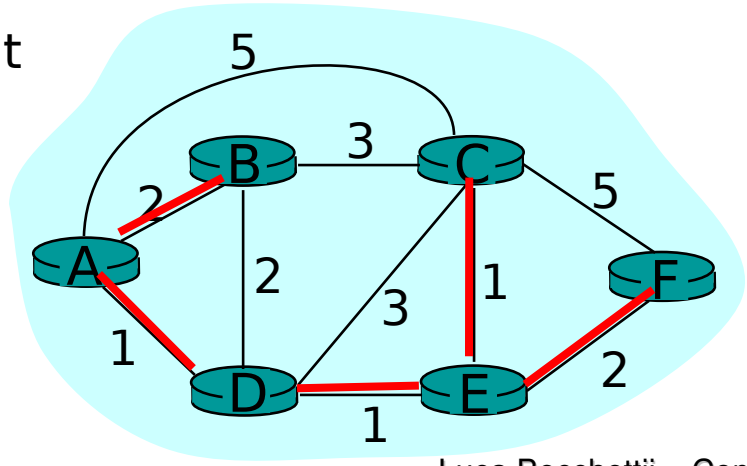
```
1  Initialization:
2    N = {A}
3    for all nodes v
4      if v adjacent to A
5        then D(v) = c(A,v)
6        else D(v) = infty
7
8   loop
9     find w not in N such that D(w) is a minimum
10    add w to N
11    update D(v) for all v adjacent to w and not in N:
12       D(v) = min( D(v), D(w) + c(w,v) )
13    /* new cost to v is either old cost to v or known
14     shortest path cost to w plus cost from w to v */
15  until all nodes in N
```



Dijkstra's algorithm: esempio

Step	start N	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	infinity	infinity
1	AD	2,A	4,D		2,D	infinity
2	ADE	2,A	3,E			4,E
3	ADEB		3,E			4,E
4	ADEBC					4,E
5	ADEBCF					

Routing Table at the end



Note: every sub-path of a shortest path is a shortest path between its end points

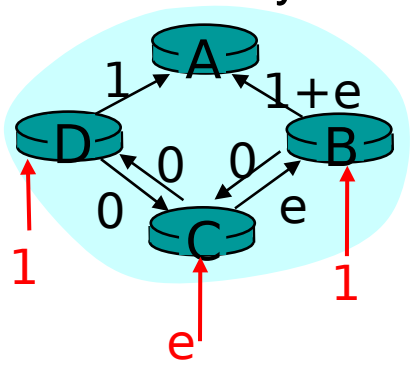
Properties

Computational complexity: n nodi

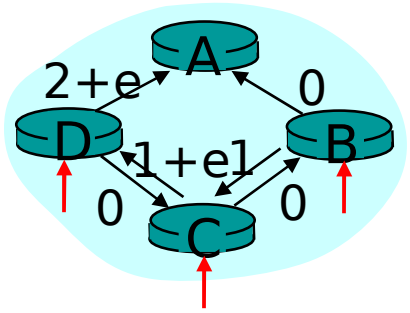
- Every iteration: check all nodes w not in N
- n*(n+1)/2 comparison worst case: O(n^2)
- More efficient heap-based implementation: O(nlogn)

Route oscillations possiblei:

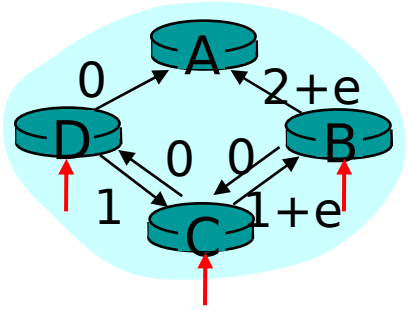
- e.g., link cost = congestion
  - Why this metric?



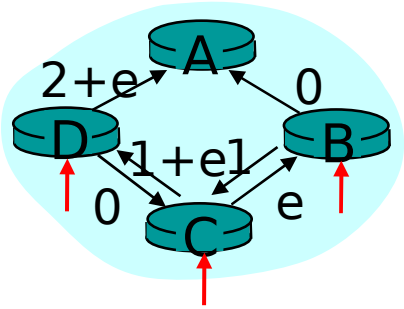
Start



recompute routing



… recompute



… recompute

Implementation

Strategies for link state information collection

Link State

- Dijkstra requires full knowledge of network at every router
- Technique: Reliable flooding
  - Node sends own link-state info to all neighbours
  - A node receiving link-state info forwards it to neighbours
  - Periodic updates
  - Use TTL to eliminate obsolete information

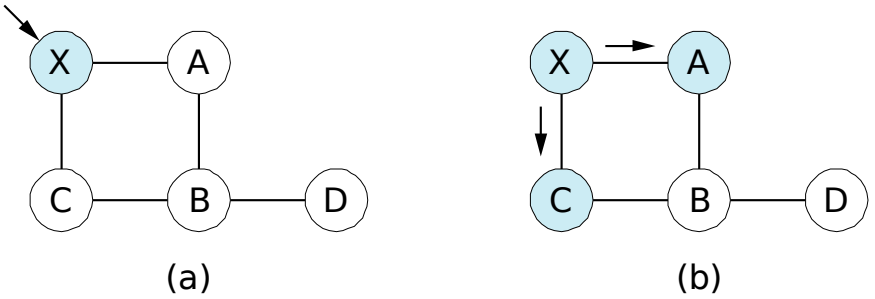
Link **S**tate **P**ackets

- ID of node that generated packet
- List (neighbour, cost of link to neighbour)
- SEQNO
- TTL

- LSP generation
  - Timer expires
  - Topology/link costs change

Link **S**tate **P**ackets

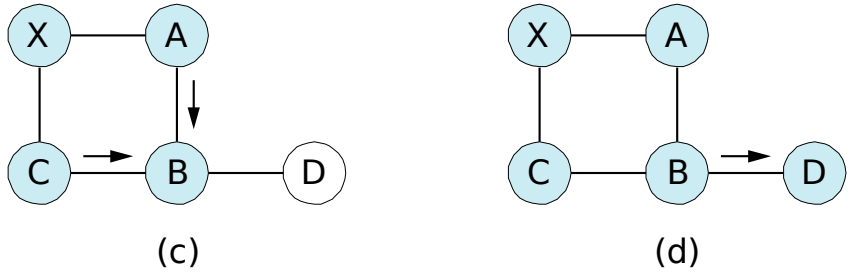
- **Reliable flooding**
  - store most recent LSP from each node
  - forward LSP to all nodes except one that sent it
  - generate new LSP periodically
    - increment SEQNO
  - start SEQNO at 0 when reboot
  - decrement TTL of each stored LSP
    - discard when TTL=0



Example: LSP originating from X and received by B

(X, ((C, 10), (A, 2)), 15, 5)

ID Neigh. Cost SEQNO TTL



Link State - recap.

- For every router at start up or by any change : send LSA (Link State Advertisement)
  - LSA contains LSP with link state info for all links connected to router
- Router exchange link state info using flooding
- As soon as router's DB complete: compute Shortest Path Tree towards all other routers
- Generate routing table:
  - Generic entry: (dest, cost, next hop)
- Update upon changes

References

- TCP/IP guide: http://www.tcpipguide.com/free/t_TCPIPRoutingProtocolsGatewayProtocols.htm
- Also: J. F. Kurose and K. W. Ross. Computer Networking: A Top-Down Approach, 4/E, Chapter 4 (4.5 and 4.6)