Reti di Calcolatori

Algoritmi di routing (I parte)

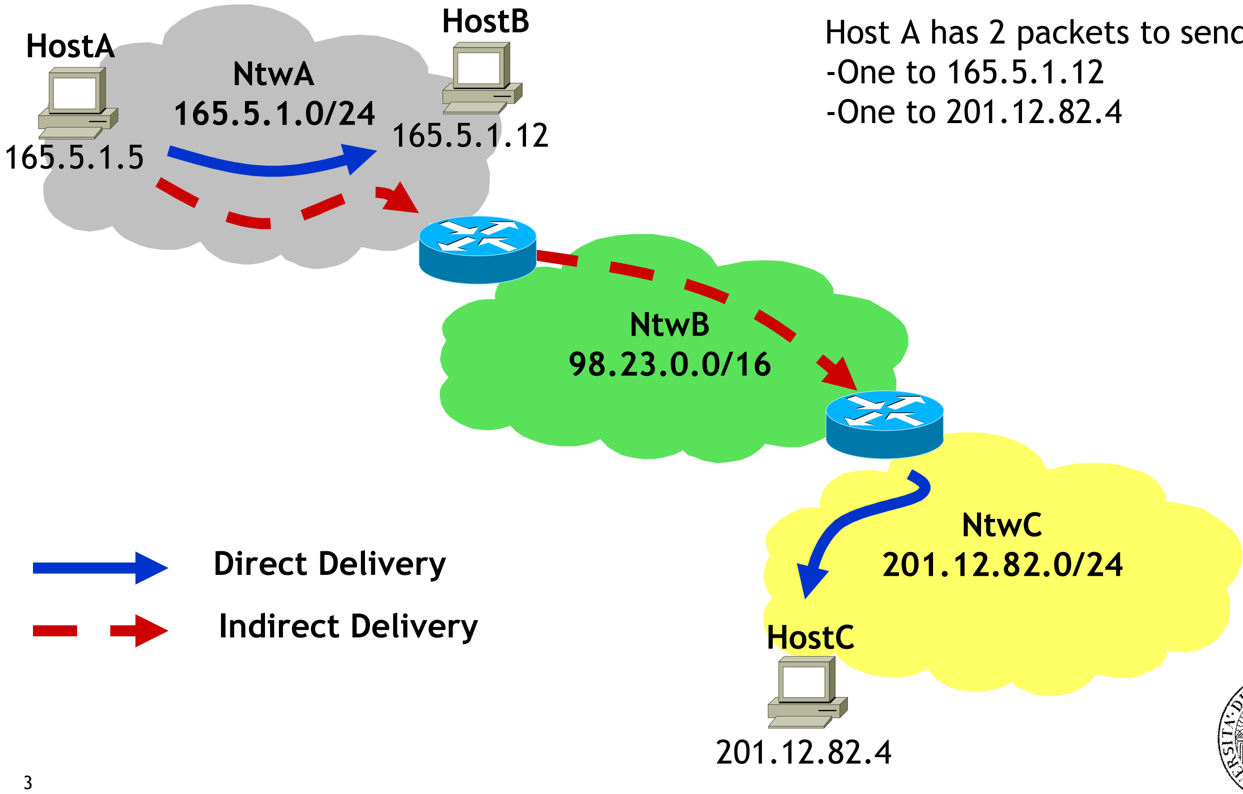
Universtità degli studi di Verona
Facoltà di Scienze MM.FF.NN.
A.A. 2009/2010
Laurea in Informatica
Docente: Damiano Carra

Direct / Indirect Delivery

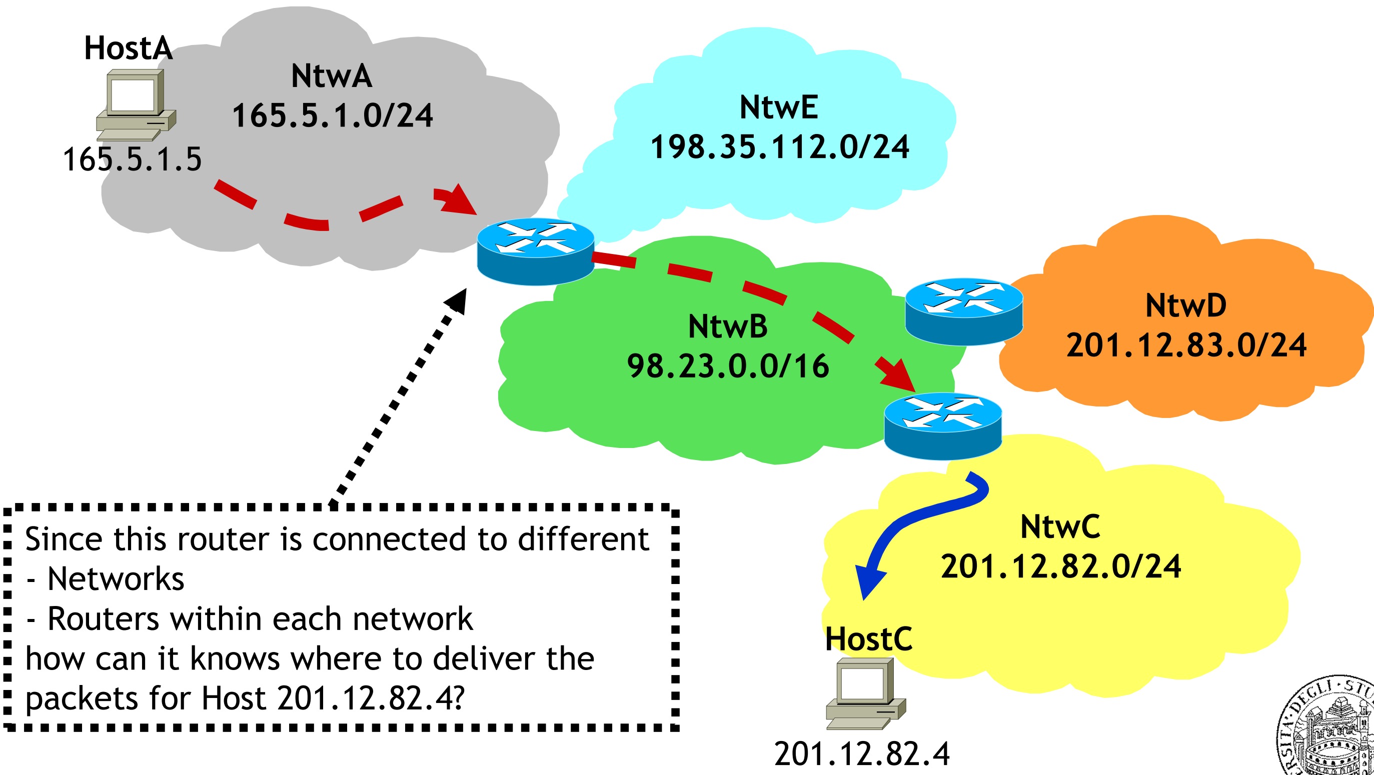
- When a host wants to send a message to a host that belongs to the same network
  - ➔ direct delivery
  - The IP address belongs to the same network
  - The physical address is obtain with ARP
- When a host wants to send a message to a host that belongs to another network
  - ➔ indirect delivery
  - Give the message to the router that will take care of the delivery
  - The intermediate step is chosen thanks to the routing algorithms

Direct / Indirect Delivery

Host A has 2 packets to send:
-One to 165.5.1.12
-One to 201.12.82.4



Direct / Indirect Delivery



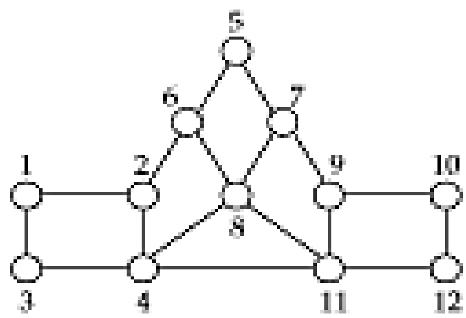
Since this router is connected to different
- Networks
- Routers within each network
how can it knows where to deliver the packets for Host 201.12.82.4?

Routing: What is it?

- ❑ Process of finding a path from a source to every destination in the network
- ❑ Suppose you want to connect to Antarctica from your desktop
  - what route should you take?
  - does a shorter route exist?
  - what if a link along the route goes down?
  - what if you're on a mobile wireless link?
- ❑ Routing deals with these types of issues

Basics

- ❑ A routing protocol sets up a routing table in routers
  - internal table that says, for each destination, which is the next output to take
- ❑ A node makes a local choice depending on global topology: this is the fundamental problem



ROUTING TABLE AT 1

Destination	Next hop	Destination	Next hop
1	—	7	2
2	2□	8□	2□
3	3□	9□	2□
4	3□	10□	2□
5	2□	11□	3□
6	2	12	3

Key problem

- How to make correct local decisions?
  - each router must know something about global state
- Global state
  - inherently large
  - dynamic
  - hard to collect
- A routing protocol must intelligently summarize relevant information

Requirements

- Minimize routing table space
  - fast to look up
  - less to exchange
- Minimize number and frequency of control messages
- Robustness: avoid
  - black holes
  - loops
  - oscillations
- Use optimal path

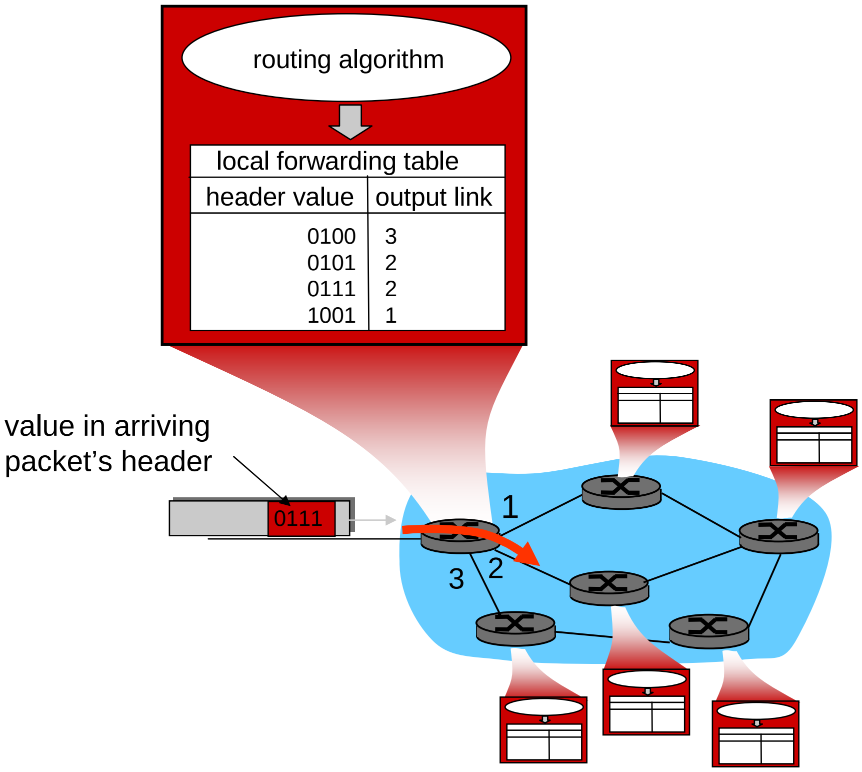
Different degrees of freedom

- ❑ Centralized vs. distributed routing
  - centralized is simpler, but prone to failure and congestion
- ❑ Global vs local information exchange
  - convey global information is expensive
- ❑ Static vs dynamic
  - static may work at the edge, not in the core
- ❑ Stochastic vs. deterministic
  - stochastic spreads load, avoiding oscillations, but misorders
- ❑ Single vs. multiple path
  - primary and alternative paths (compare with stochastic)
- ❑ State-dependent vs. state-independent
  - do routes depend on current network state (e.g. delay)

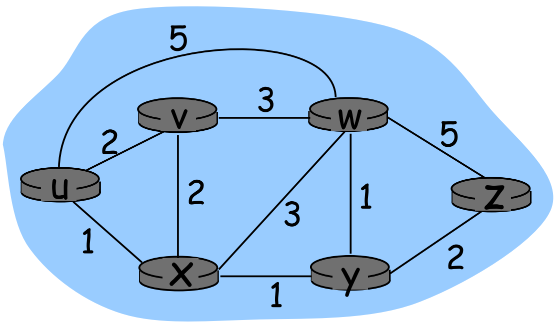
Dynamic Routing And Routers

- ❑ To ensure that all routers maintain information about how to reach each possible destination
  - each router uses a route propagation protocol
    - to exchange information with other routers
  - when it learns about changes in routes
    - updates the local routing table
- ❑ Because routers exchange information periodically
  - the local routing table is updated continuously

Interplay between routing, forwarding



Graph abstraction

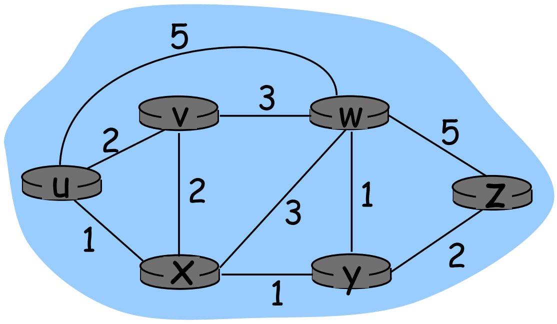


Graph: G = (N,E)

N = set of routers = { u, v, w, x, y, z }

E = set of links ={ (u,v), (u,x), (v,x), (v,w), (x,w), (x,y), (w,y), (w,z), (y,z) }

Graph abstraction: costs



- $c(x,x')$ = cost of link (x,x')
  - e.g., $c(w,z) = 5$
- cost could always be 1, or inversely related to bandwidth, or inversely related to congestion

Cost of path $(x_1, x_2, x_3,\ldots, x_p) = c(x_1,x_2) + c(x_2,x_3) + \ldots + c(x_{p-1},x_p)$

Question: What's the least-cost path between u and z ?

Routing algorithm: algorithm that finds least-cost path

Distance Vector Algorithms

Consistency criterion

Define

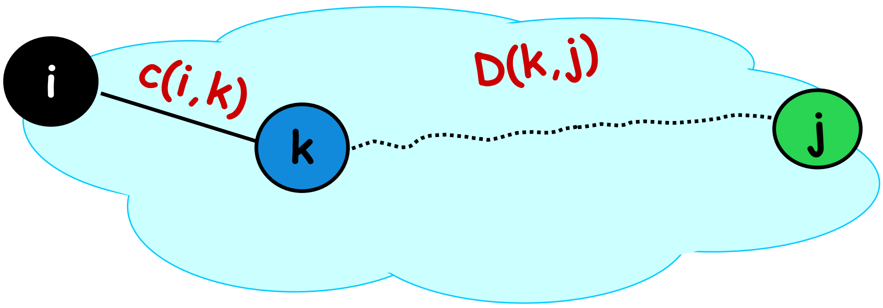
c(i,k) := cost from i to k (direct connection)

D(i,j) := cost of least-cost path from i to j

➔ The subset of a shortest path is also the shortest path between the two intermediate nodes

❑ Then, if the shortest path from node i to node j, with distance D(i,j), passes through neighbor k, with link cost c(i,k), we have:

- D(i,j) = c(i,k) + D(k,j)



Distance Vector (DV) algorithm

❑ Initial distance values (iteration 1):

- D(i,i) = 0 ;
- D(i,k) = c(i,k) if k is a neighbor (i.e. k is one-hop away); and
- D(i,j) = INFINITY for all other non-neighbors j.

❑ Note that the set of values D(i,*) is a distance vector at node i.

❑ The algorithm also maintains a next-hop value (forwarding table) for every destination j, initialized as:

- next-hop(i) = i;
- next-hop(k) = k if k is a neighbor, and
- next-hop(j) = UNKNOWN if j is a non-neighbor.

Distance Vector (DV) algorithm

- After every iteration each node i exchanges its distance vectors D(i,*) with its immediate neighbors.
- For any neighbor k, if c(i,k) + D(k,j) < D(i,j), then:
  - D(i,j) = c(i,k) + D(k,j)
  - next-hop(j) = k

In summary

Basic idea:

- From time-to-time, each node sends its own distance vector estimate to neighbors

Asynchronous

- When a node x receives new DV estimate from neighbor, it updates its own DV using B-F equation:

  $D(x,y) \leftarrow \min_v\{c(x,v) + D(v,y)\}$ for each node $y \in N$

- Under minor, natural conditions, the estimate D(x,y) converges to the actual least cost

In summary

- Iterative, asynchronous:

  each local iteration caused by:

  - local link cost change
  - DV update message from neighbor

- Distributed:

  each node notifies neighbors only when its DV changes

  - neighbors then notify their neighbors if necessary

Each node:

wait for (change in local link cost or msg from neighbor)

↓

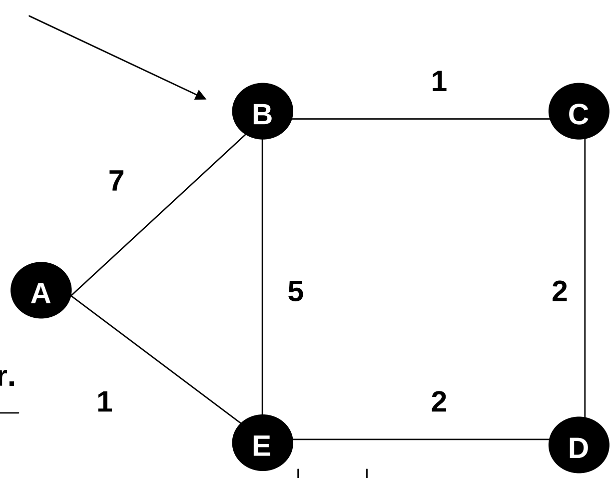
recompute estimates

↓

if DV to any dest has changed, *notify* neighbors

Distance Vector: example (starting point)

B	dist	thr.
→A	7	A
→B	0	-
→C	1	C
→D	-	-
→E	5	E



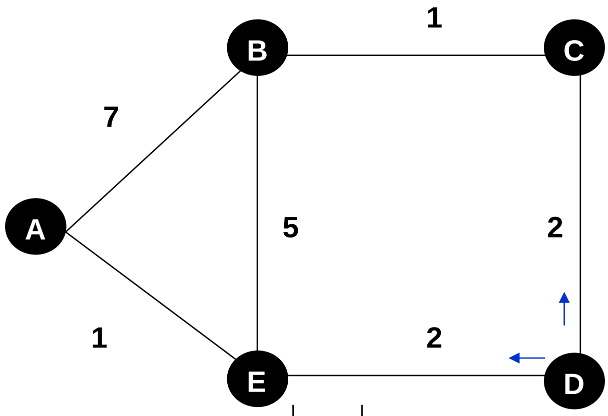
C	dist	thr.
→A	-	-
→B	1	B
→C	0	-
→D	2	D
→E	-	-

A	dist	thr.
→A	0	-
→B	7	B
→C	-	-
→D	-	-
→E	1	E

E	dist	thr.
→A	1	A
→B	5	B
→C	-	-
→D	2	D
→E	0	-

D	dist	thr.
→A	-	-
→B	-	-
→C	2	C
→D	0	-
→E	2	E

Distance Vector: example (running)

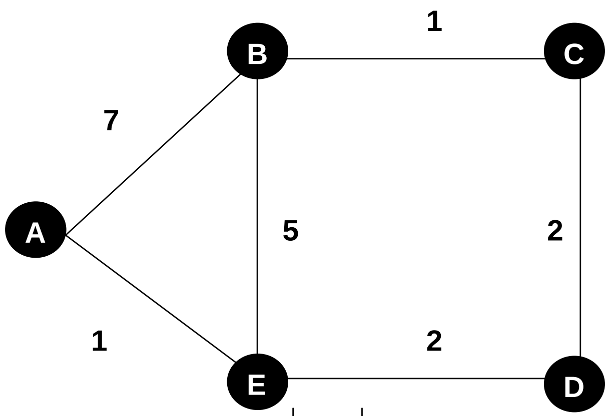


C	dist	thr.
→A	-	-
→B	1	B
→C	0	-
→D	2	D
→E	-	-

E	dist	thr.
→A	1	A
→B	5	B
→C	-	-
→D	2	D
→E	0	-

D	dist	thr.
→A	-	-
→B	-	-
→C	2	C
→D	0	-
→E	2	E

Distance Vector: example (running)



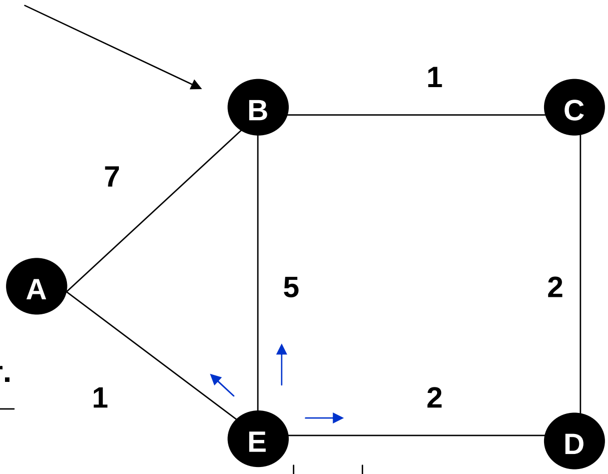
C	dist	thr.
→A	-	-
→B	1	B
→C	0	-
→D	2	D
→E	4	D

E	dist	thr.
→A	1	A
→B	5	B
→C	4	D
→D	2	D
→E	0	-

D	dist	thr.
→A	-	-
→B	-	-
→C	2	C
→D	0	-
→E	2	E

Distance Vector: example (running)

B	dist	thr.
→A	7	A
→B	0	-
→C	1	C
→D	-	-
→E	5	E



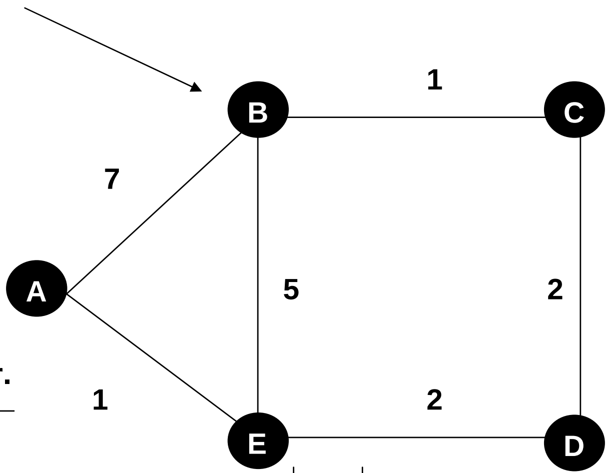
A	dist	thr.
→A	0	-
→B	7	B
→C	-	-
→D	-	-
→E	1	E

E	dist	thr.
→A	1	A
→B	5	B
→C	4	D
→D	2	D
→E	0	-

D	dist	thr.
→A	-	-
→B	-	-
→C	2	C
→D	0	-
→E	2	E

Distance Vector: example (running)

B	dist	thr.
→A	6	E
→B	0	-
→C	1	C
→D	7	E
→E	5	E



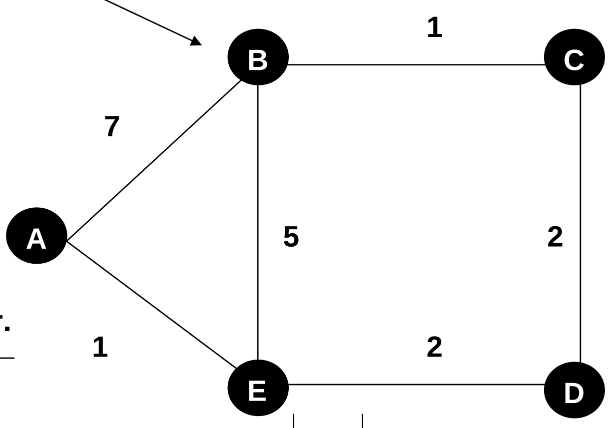
A	dist	thr.
→A	0	-
→B	6	E
→C	5	E
→D	3	E
→E	1	E

E	dist	thr.
→A	1	A
→B	5	B
→C	4	D
→D	2	D
→E	0	-

D	dist	thr.
→A	3	E
→B	7	E
→C	2	C
→D	0	-
→E	2	E

Distance Vector: example (final point)

B	dist	thr.
→A	6	E
→B	0	-
→C	1	C
→D	3	C
→E	5	E



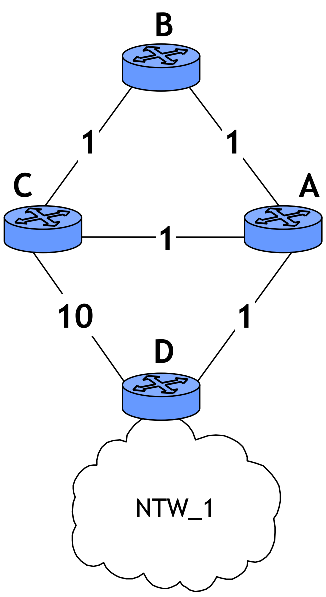
C	dist	thr.
→A	5	D
→B	1	B
→C	0	-
→D	2	D
→E	4	D

A	dist	thr.
→A	0	-
→B	6	E
→C	5	E
→D	3	E
→E	1	E

E	dist	thr.
→A	1	A
→B	5	B
→C	4	D
→D	2	D
→E	0	-

D	dist	thr.
→A	3	E
→B	3	C
→C	2	C
→D	0	-
→E	2	E

Problem: “counting to infinity”



Router A

Dest	Next	Metric
NTW_1	D	2

Router B

Dest	Next	Metric
NTW_1	A	3

Router C

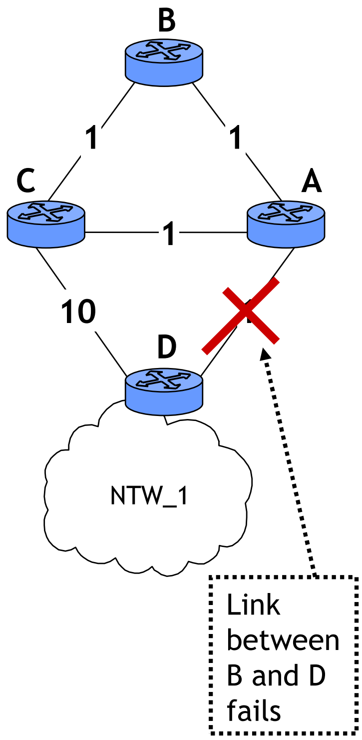
Dest	Next	Metric
NTW_1	A	3

Router D

Dest	Next	Metric
NTW_1	dir	1

- ❑ Consider the entries in each routing table for network NTW_1
- ❑ Router D is directly connected to NTW_1

Problem: “counting to infinity”



time →

Router A

Dest	Next	Metric
NTW_1	Unr.	-

Router B

Dest	Next	Metric
NTW_1	A	3

Router C

Dest	Next	Metric
NTW_1	A	3

Router D

Dest	Next	Metric
NTW_1	dir	1

Router A

Dest	Next	Metric
NTW_1	C	4

Router B

Dest	Next	Metric
NTW_1	C	4

Router C

Dest	Next	Metric
NTW_1	B	4

Router D

Dest	Next	Metric
NTW_1	dir	1

Router A

Dest	Next	Metric
NTW_1	C	5

Router B

Dest	Next	Metric
NTW_1	C	5

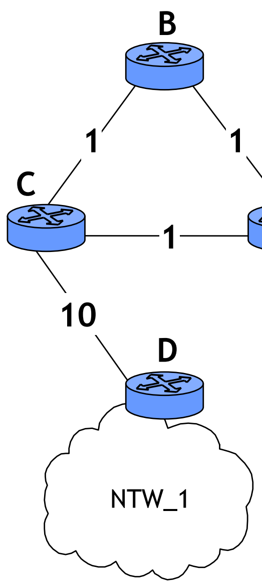
Router C

Dest	Next	Metric
NTW_1	B	5

Router D

Dest	Next	Metric
NTW_1	dir	1

Problem: “counting to infinity”



time →

…

Router A

Dest	Next	Metric
NTW_1	C	11

Router B

Dest	Next	Metric
NTW_1	C	11

Router C

Dest	Next	Metric
NTW_1	B	11

Router D

Dest	Next	Metric
NTW_1	dir	1

Router A

Dest	Next	Metric
NTW_1	C	12

Router B

Dest	Next	Metric
NTW_1	C	12

Router C

Dest	Next	Metric
NTW_1	D	11

Router D

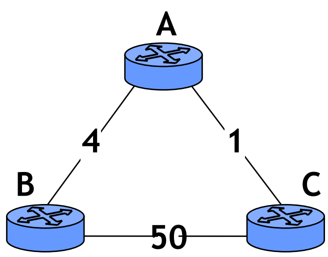
Dest	Next	Metric
NTW_1	dir	1

Solution to "counting to infinity"

- Maximum number of hops bounded to 15
  - this limits the convergence time
- Split Horizon
  - simple
    - each node *omits* routes learned from one neighbor in update sent to that neighbor
  - with poisoned reverse
    - each node *include* routes learned from one neighbor in update sent to that neighbor, setting their metrics to infinity
      - drawback: routing message size greater than simple Split Horizon

Distance Vector: link cost changes

- If link cost changes:
  - good news travels fast
    - good = cost decreases
  - bad news travels slow
    - bad = cost increases



- Exercise
  - try to apply the algorithm in the simple scenario depicted above when
    - the cost of the link A → B changes from 4 to 1
    - the cost of the link A → B changes from 4 to 60

Routing Information Protocol (RIP)

RIP - History

- Late 1960s: Distance Vector protocols were used in the ARPANET
- Mid-1970s: XNS (Xerox Network system) routing protocol is the precursor of RIP in IP (and Novell's IPX RIP and Apple's routing protocol)
- 1982 Release of routed for BSD Unix
- 1988 RIPv1 (RFC 1058)
  - classful routing
- 1993 RIPv2 (RFC 1388)
  - adds subnet masks with each route entry
  - allows classless routing
- 1998 Current version of RIPv2 (RFC 2453 and STD 56)

RIP at a glance

- A simple intradomain protocol
- Straightforward implementation of Distance Vector Routing…
  - Distributed version of Bellman-Ford (DBF)

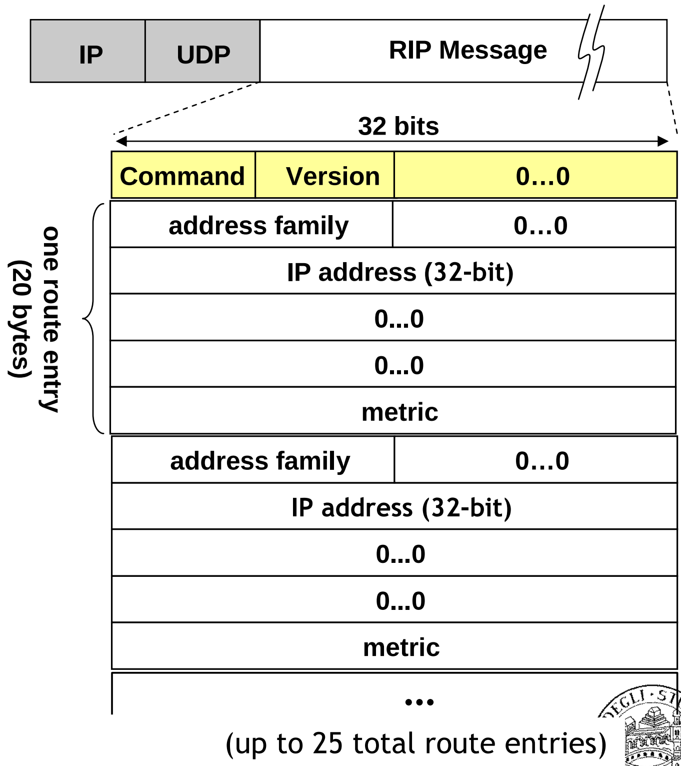
  …with well known issues
  - slow convergence
  - works with limited network size
- Strengths
  - simple to implement
  - simple management
  - widespread use

RIP at a glance

- Metric based on hop count
  - maximum hop count is 15, with “16” equal to “∞”
    - imposed to limit the convergence time
  - the network administrator can also assign values higher than 1 to a single hop
- Each router advertises its distance vector every 30 seconds (or whenever its routing table changes) to all of its neighbors
  - RIP uses UDP, port 520, for sending messages
- Changes are propagated across network
- Routes are timeout (set to 16) after 3 minutes if they are not updated

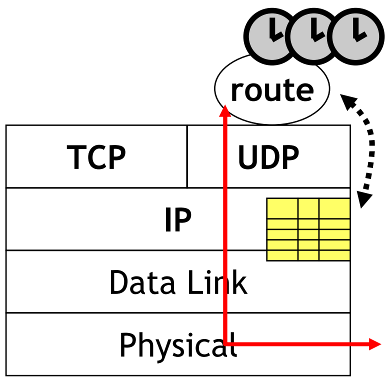
RIP: Message Format

- ❑ Command: 1=request 2=response
  - Updates are replies whether asked for or not
  - Initializing node broadcasts request
  - Requests are replied to immediately
- ❑ Version: 1
- ❑ Address family: 2 for IP
- ❑ IP address: non-zero network portion, zero host portion
  - Identifies particular network
- ❑ Metric
  - Path distance from this router to network
  - Typically 1, so metric is hop count



RIP procedures: introduction

- ❑ RIP routing tables are managed by application-level process
  - e.g., *routed* on UNIX machines
- ❑ Advertisements are sent in UDP packets (port 520)
- ❑ RIP maintains 3 different timers to support its operations
  - Periodic update timer (25-30 sec)
    - used to sent out update messages
  - Invalid timer (180 sec)
    - If update for a particular entry is not received for 180 sec, route is invalidated
  - Garbage collection timer (120 sec)
    - An invalid route in marked, not immediately deleted
    - For next 120 s. the router advertises this route with distance infinity

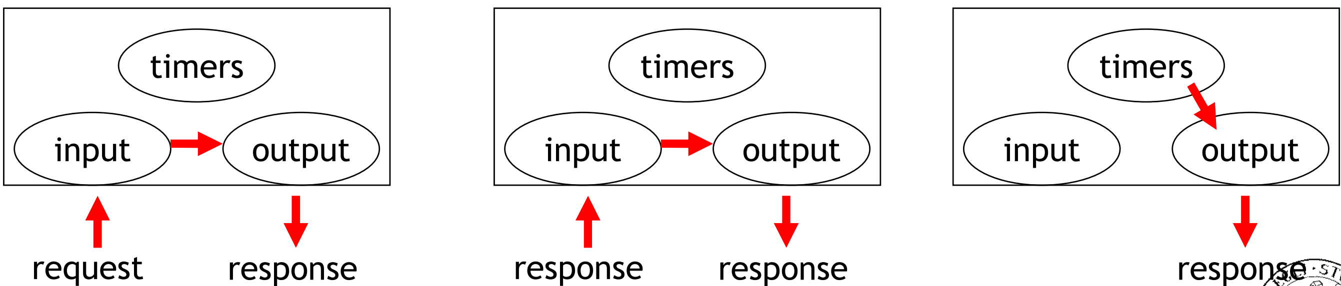


RIP procedures: input processing

- ❑ Request Messages
  - they may arrive from routers which have just come up
  - action: the router responds directly to the requestor's address and port
    - request is processed entry by entry
- ❑ Response Messages
  - they may arrive from routers that perform regular updates, triggered updates or respond to a specific query
  - action: the router updates its routing table
    - in case of new route or changed routes, the router starts a triggered update procedure

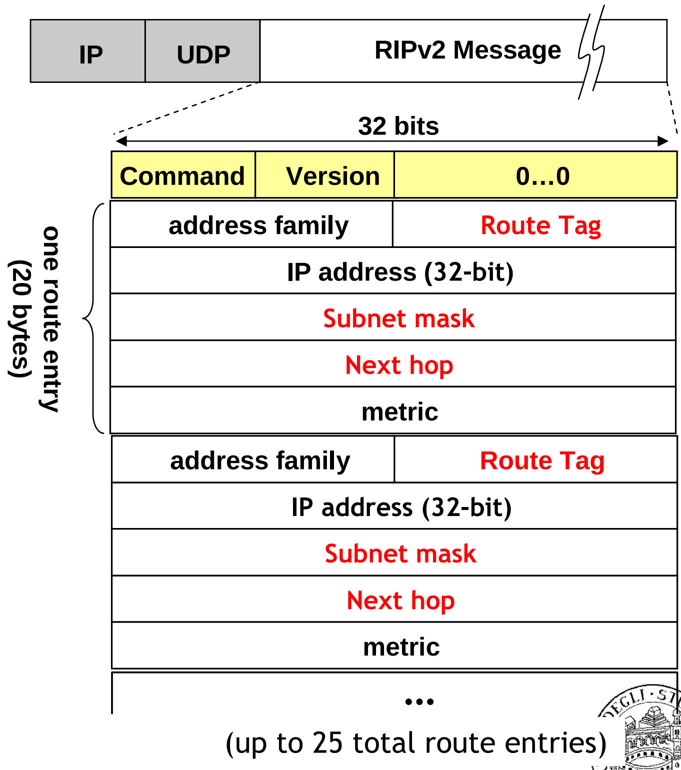
RIP procedures: output processing

- ❑ Output are generated
  - when the router comes up in the network
  - if required by the input processing procedures
  - by regular routing update
- ❑ Action: the router generates the messages according to the commands received
  - the messages contain entries from the routing table



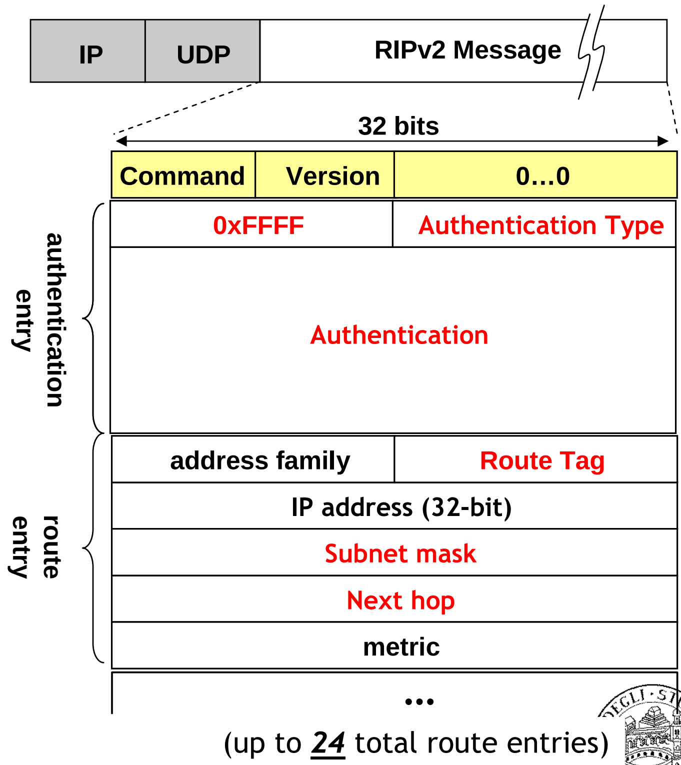
RIPv2: Message Format

- ❑ Version: 2
- ❑ Route Tag: used to carry information from other routing protocols
  - e.g., autonomous system number
- ❑ Subnet mask for IP address
- ❑ Next hop
  - identifies a better next-hop address on the same subnet than the advertising router, if one exists (otherwise 0….0)



RIPv2: authentication

- ❑ Any host sending packets on UDP port 520 would be considered a router
- ❑ Malicious users can inject fake routing entries
- ❑ With authentication, only authorized router can send Rip packets
  - Authentication type
    - password
    - MD5
  - Authentication
    - plain text password
    - MD5 hash



RIPv2: other aspects

- Explicit use of subnets
- Interoperability
  - RIPv1 and RIPv2 can be present in the same network since RIPv1 simply ignores fields not known
    - RIPv2 responds to RIPv1 Request with a RIPv1 Response
- Multicast
  - instead of broadcasting RIP messages, RIPv2 uses multicast address 224.0.0.9

RIP limitations: the cost of simplicity

- Destinations with metric more than 15 are unreachable
  - If larger metric allowed, convergence becomes lengthy
- Simple metric leads to sub-optimal routing tables
  - Packets sent over slower links
- Accept RIP updates from any device (if no security is implemented)
  - Misconfigured device can disrupt entire configuration

Reti di Calcolatori

Algoritmi di routing (II parte)

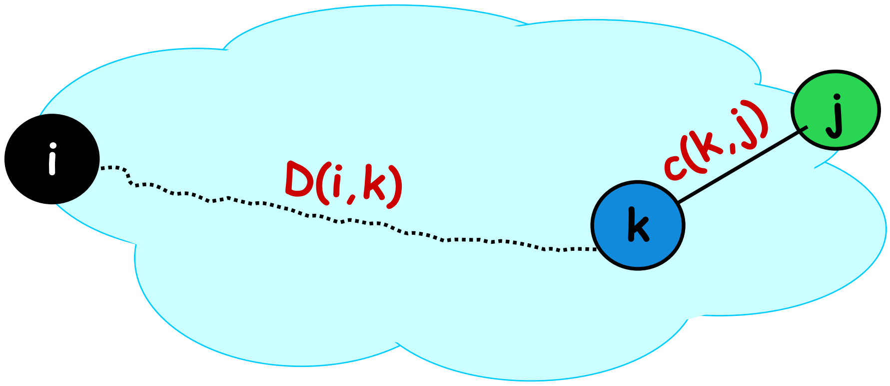
Recap on Distance Vector algorithms

- Each node has a local visibility
  - direct neighbors
- Information about the routing are inferred from the information obtained from the neighbors
  - but the "structure" is not specified

- Link state algorithms, instead, try to have a global view

Link State (LS) Approach

- ❑ The link state (Dijkstra) approach is iterative, but it pivots around destinations j, and their predecessors k = p(j)
  - Observe that an alternative version of the consistency condition holds for this case: D(i,j) = D(i,k) + c(k,j)



- ❑ Each node i collects all link states c(*,*) first and runs the complete Dijkstra algorithm locally.

Link State (LS) Approach…

- ❑ After each iteration, the algorithm finds a new destination node j and a shortest path to it.
- ❑ After m iterations the algorithm has explored paths, which are m hops or smaller from node i.
  - It has an m-hop view of the network just like the distance-vector approach
- ❑ The Dijkstra algorithm at node i maintains two sets:
  - set N that contains nodes to which the shortest paths have been found so far, and
  - set M that contains all other nodes.
  - For all nodes k, two values are maintained:
    - D(i,k): current value of distance from i to k.
    - p(k): the predecessor node to k on the shortest known path from i

Dijkstra: Initialization

- Initialization:
  - $D(i,i) = 0$ and $p(i) = i$;
  - $D(i,k) = c(i,k)$ and $p(k) = i$ if k is a neighbor of I
  - $D(i,k)$ = INFINITY and $p(k)$ = UNKNOWN if k is not a neighbor of I
  - Set N = { i }, and next-hop (i) = I
  - Set M = { j | j is not i}

- Initially set N has only the node i and set M has the rest of the nodes.
- At the end of the algorithm, the set N contains all the nodes, and set M is empty

Dijkstra: Iteration

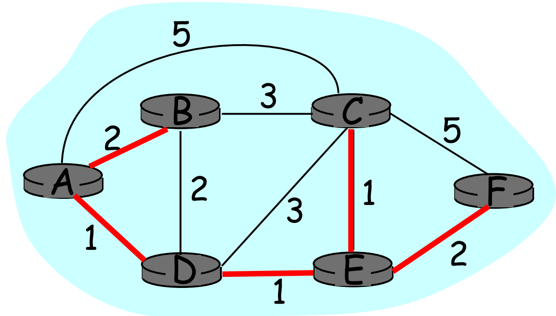
- In each iteration, a new node j is moved from set M into the set N.
  - Node j has the minimum distance among all current nodes in M, i.e. $D(i,j) = \min_{\{l \in M\}} D(i,l)$.
  - If multiple nodes have the same minimum distance, any one of them is chosen as j.
  - Next-hop(j) = the neighbor of i on the shortest path
    - Next-hop(j) = next-hop(p(j)) if p(j) is not i
    - Next-hop(j) = j if p(j) = i
  - Now, in addition, the distance values of any neighbor k of j in set M is reset as:
    - If $D(i,k) < D(i,j) + c(j,k)$, then

$$D(i,k) = D(i,j) + c(j,k), \text{ and } p(k) = j.$$

- This operation is called "relaxing" the edges of node j.

Dijkstra's algorithm: *example*

Step	set 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					



The shortest-paths spanning tree rooted at A is called an SPF-tree

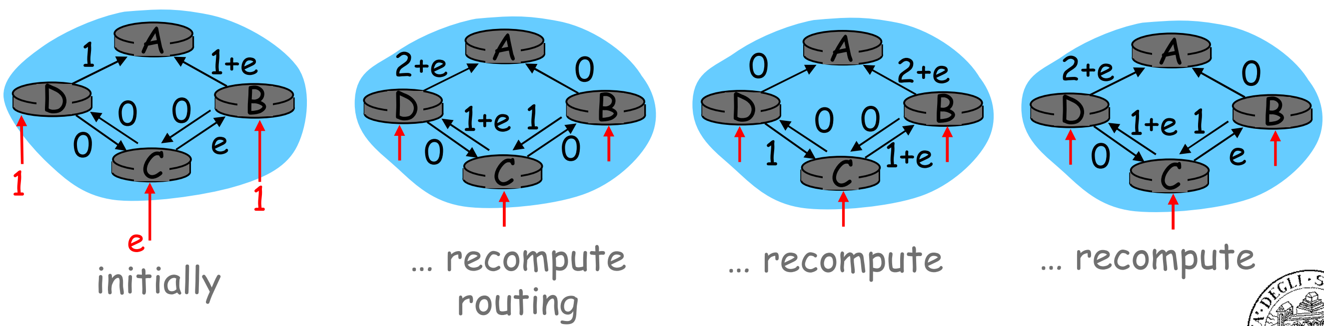
Dijkstra's algorithm, discussion

Algorithm complexity: n nodes

- each iteration: need to check all nodes, w, not in N
- n(n+1)/2 comparisons: $O(n^2)$
- more efficient implementations possible: O(nlogn)

Oscillations possible:

- e.g., link cost = amount of carried traffic



Misc: How to assign the Cost Metric?

- Choice of link cost defines traffic load
  - Low cost = high probability link belongs to SPT and will attract traffic
- Tradeoff: convergence vs load distribution
  - Avoid oscillations
  - Achieve good network utilization
- Static metrics (weighted hop count)
  - Does not take traffic load (demand) into account.
- Dynamic metrics (cost based upon queue or delay etc)
  - Highly oscillatory, very hard to dampen (DARPAnet experience)
- Quasi-static metric:
  - Reassign static metrics based upon overall network load (demand matrix), assumed to be quasi-stationary

Summary: Distributed Routing Techniques

Link State

- Topology information is flooded within the routing domain
- Best end-to-end paths are computed locally at each router.
- Best end-to-end paths determine next-hops.
- Based on minimizing some notion of distance
- Works only if policy is shared and uniform
- Examples: OSPF

Vectoring

- Each router knows little about network topology
- Only best next-hops are chosen by each router for each destination network.
- Best end-to-end paths result from composition of all next-hop choices
- Does not require any notion of distance
- Does not require uniform policies at all routers
- Examples: RIP

Comparison of LS and DV algorithms

Message complexity

❑LS: with n nodes, E links, O(nE) msgs sent

❑DV: exchange between neighbors only

- convergence time varies

Speed of Convergence

❑LS: O(n2) algorithm requires O(nE) msgs

- may have oscillations

❑DV: convergence time varies

- may be routing loops
- count-to-infinity problem

Robustness: what happens if router malfunctions?

❑LS:

- node can advertise incorrect link cost
- each node computes only its own table

❑DV:

- DV node can advertise incorrect path cost
- each node's table used by others
  - error propagate thru network

OSPF
Open Shortest Path First

Open Shortest Path First

- ❑ Nel 1988 IETF ha avviato la standardizzazione di un nuovo protocollo di routing
- ❑ IETF ha elencato in fase di avvio della standardizzazione un insieme di requisiti che il nuovo protocollo avrebbe dovuto rispettare:
  - soluzione NON proprietaria – aperta
  - parametri di distanza multipli
  - algoritmo dinamico
  - routing basato su *Type of Service*
  - *load balancing*
  - supporto di sistemi gerarchici
  - funzionalità di sicurezza
- ❑ Open Shortest Path First (1990, RFC 1247)

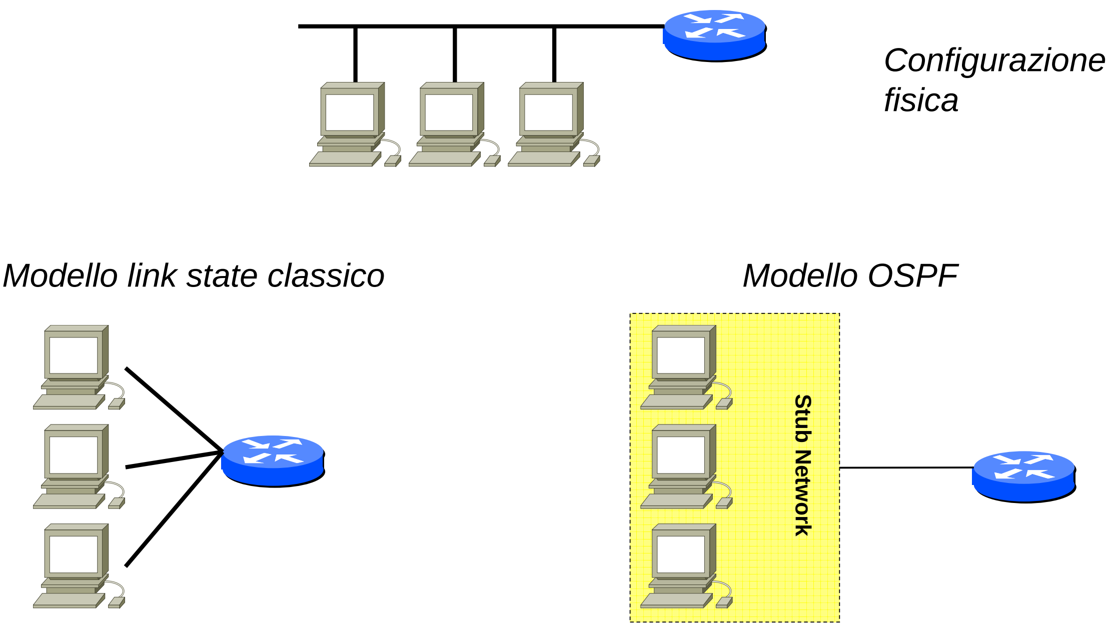
Criteri di progettazione

- ❑ I tre principali criteri di progettazione del protocollo OSPF sono:
  - distinzione tra host e router
  - reti broadcast
  - suddivisione delle reti di grandi dimensioni

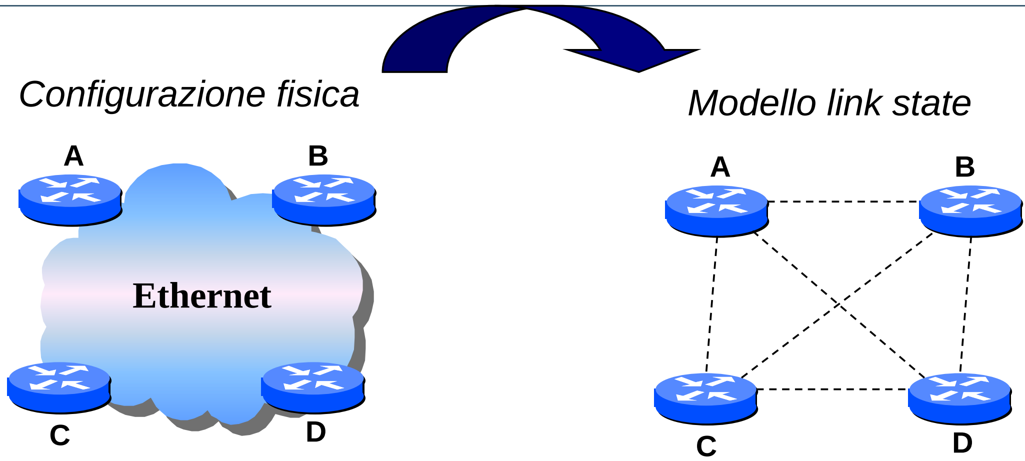
Distinzione host/router (1)

- Nelle reti IP generalmente gli host sono collocati nelle aree periferiche della rete a sottoreti locali connesse alla Big Internet attraverso router
- Il modello link state prevede che il database *link state* includa una entry per ogni link tra host e router
- OSPF introduce il concetto di link ad una *stub network*
  - il link viene identificato dall'indirizzo della sottorete

Distinzione host/router (2)



Reti broadcast (1)

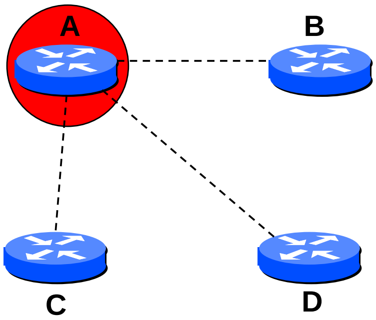


$$\frac{N \cdot (N-1)}{2}$$

adiacenze

Reti broadcast (2)

- Elezione di un nodo della rete broadcast a *designated router*
- L'aggiornamento delle adiacenze viene fatto da tutti gli altri router solamente verso il designated router

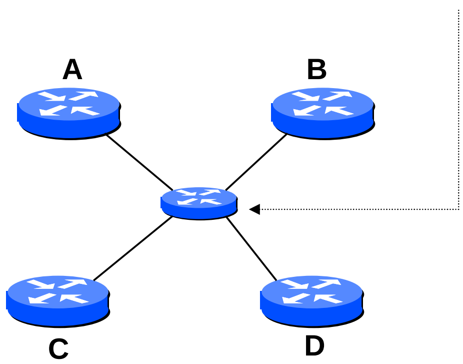


Reti broadcast (3)

- Ogni aggiornamento di link viene notificato solamente al designated router
  - indirizzo multicast “all designated router”: 224.0.0.6
- Se l’aggiornamento modifica il database link state del designated router, quest’ultimo lo propaga in flooding a tutti gli altri nodi
  - indirizzo multcast “all OSPF router”: 224.0.0.5
- Affidabilità ottenuta attraverso un *backup designated router* operante in modo “silenzioso”

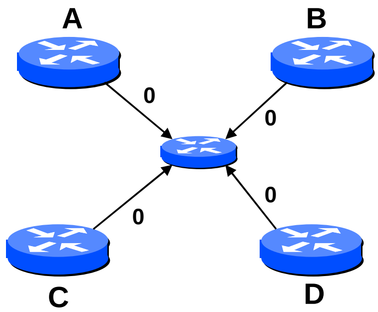
Reti broadcast (4)

- La rete viene rappresentata come un **nodo virtuale**
- Due link per ogni router
  - dal nodo vitruale al router
  - dal router al nodo virtuale (**network link**)



Reti broadcast (5)

- La distanza tra due nodi “raddoppia”
- Soluzione: i network link hanno peso nullo

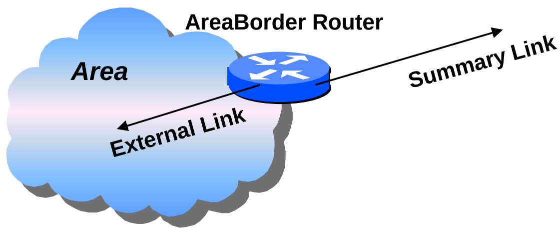


Aree multiple (1)

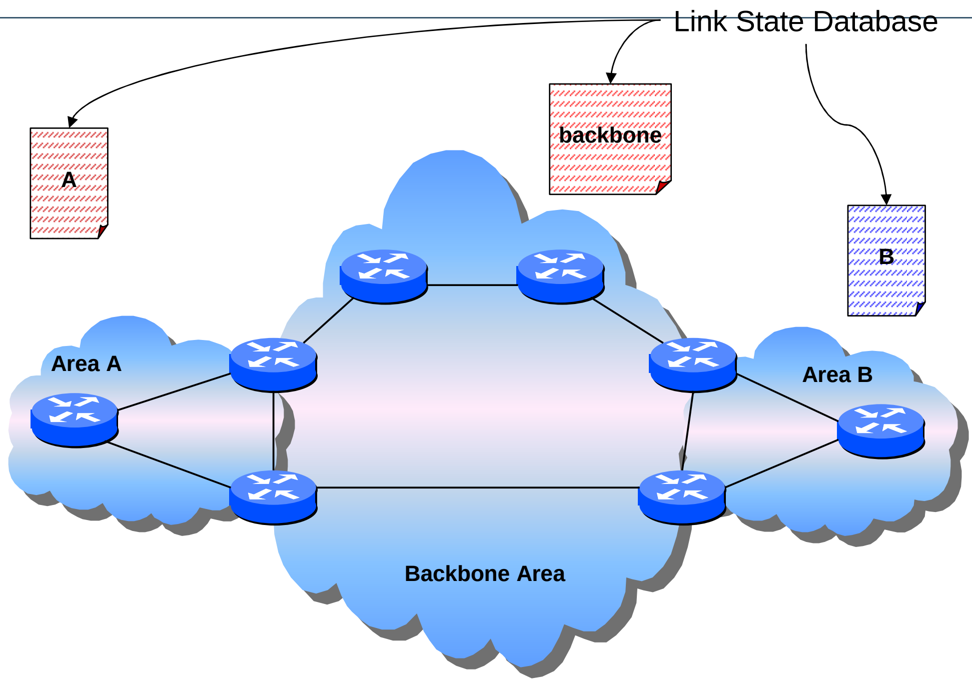
- Il numero di nodi della rete influenza direttamente:
  - dimensione del database di link state di ogni nodo
  - tempo di calcolo dei percorsi ottimi nella rete
  - quantità dei messaggi di routing distribuiti
- OSPF prevede di “spezzare” l’intera rete in un insieme di sezioni independenti chiamate **aree**
- Sono locali ad ongi area
  - i record del database di link state
  - il flooding dei messaggi di routing
  - il calcolo dei percorsi ottimi di instradamento
- **Backbone Area**: area di livello gerarchico superiore

Aree multiple (2)

- **Area-Border Router**
  - router sono configurati come appartenenti a più aree in modo da garantire l'instradamento inter-area
- Gli Area-Border Router distribuiscono
  - external link state record
    - informano i nodi di un area relativamente ai percorsi uscenti
  - summary link state record
    - informano i nodi della backbone area dei percorsi entranti

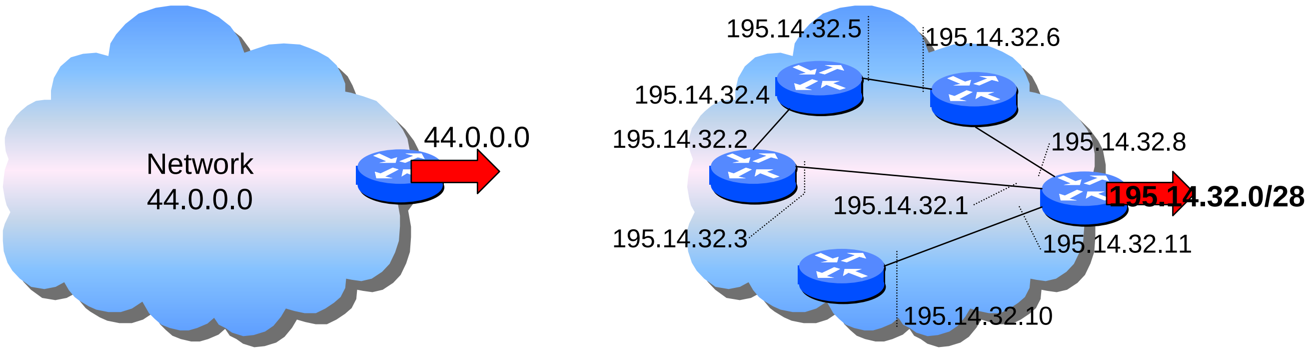


Aree multiple (3)



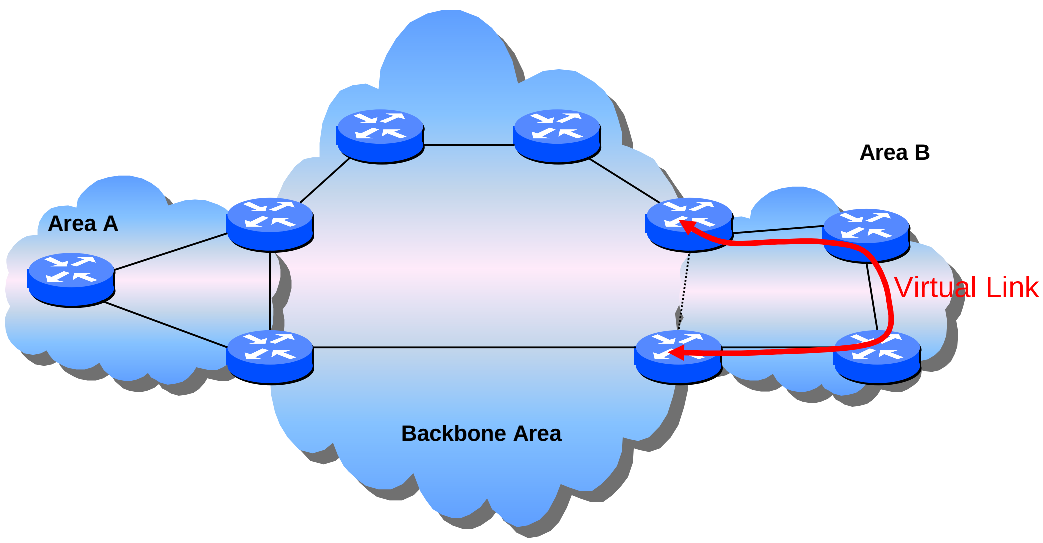
Aree multiple (4)

❑ Ogni border router “sommarizza” le informazioni di instradamento relative alla propria area



Aree multiple (5)

❑ Guasti in una Backbone Area possono essere gestiti utilizzando i *virtual link*



Type of Service

- Per ogni link nel database link state possono essere memorizzate più metriche
  - Type of Service Metrics
- Al momento di aggiornamento delle tabelle di routing vengono distribuite tutte le metriche presenti per ogni link
- Il calcolo del percorso ottimo viene fatto
  - sempre per quanto riguarda la metrica di default (ToS 0)
  - opzionalmente per le altre metriche
- I pacchetti IP vengono quindi instradati sulla base del valore contenuto nel campo ToS del loro header

Il protocollo OSPF

- Il protocollo OSPF utilizza a sua volta 3 protocolli per svolgere le proprie funzionalità
  - Hello Protocol
  - Exchange Protocol
  - Flooding Protocol

Messaggi OSPF (1)

- I messaggi OSPF sono trasportati direttamente all'interno dei pacchetti IP
  - non viene utilizzato il livello di trasporto
- Tutti i messaggi OSPF condividono lo stesso header

<table>
<tr><td>Version #</td><td>Type</td><td colspan="2">Packet length</td></tr>
<tr><td colspan="4">Router ID</td></tr>
<tr><td colspan="4">Area ID</td></tr>
<tr><td colspan="2">Checksum</td><td colspan="2">Auth Type</td></tr>
<tr><td colspan="4">Authentication</td></tr>
<tr><td colspan="4">Authentication</td></tr>
</table>

Messaggi OSPF (2)

- Version # = 2
- Type: indica il tipo di messaggio
- Packet Length: numero di byte del messaggio
- Router ID: indirizzo IP del router di riferimento

<table>
<tr><td>Version #</td><td>Type</td><td colspan="2">Packet length</td></tr>
<tr><td colspan="4">Router ID</td></tr>
<tr><td colspan="4">Area ID</td></tr>
<tr><td colspan="2">Checksum</td><td colspan="2">Auth Type</td></tr>
<tr><td colspan="4">Authentication</td></tr>
<tr><td colspan="4">Authentication</td></tr>
</table>

Messaggi OSPF (3)

- Area ID: identificativo dell'area
  - 0 per la Bacvbone area
- Auth Type: tipo di autenticazione
  - 0 no autenticazione, 1 autenticazione con passwd
- Authentication: password

Version #	*Type*	*Packet length*
Router ID		
Area ID		
Checksum		*Auth Type*
Authentication		
Authentication		

Il protocollo Hello

- Funzioni:
  - verificare l'operatività dei link
  - elezione del *designated router* (e relativo elemento di backup)
- Messaggi:
  - Hello

Common header (type = 1, hello)		
Network mask		
Hello interval	*Options*	*Priority*
Dead interval		
Designated router		
Backup Designated router		
Neighbor		

Hello Protocol: formato pacchetto (1)

- Network mask: maschera della sottorete cui appartiene l'interfaccia
- Hello interval: intervallo temporale di separazione tra due messaggi di Hello

Common header (type = 1, hello)		
Network mask		
Hello interval	*Options*	*Priority*
Dead interval		
Designated router		
Backup Designated router		
Neighbor		

Hello Protocol: formato pacchetto (2)

- Designated router: indirizzo IP del designated router
  - 0 se non è stato ancora eletto
- Backup desigated router: indirizzo IP del backup designated router

Common header (type = 1, hello)		
Network mask		
Hello interval	*Options*	*Priority*
Dead interval		
Designated router		
Backup Designated router		
Neighbor		

Hello Protocol: formato pacchetto (3)

- Neighbor: lista di nodi adiacenti da cui ha ricevuto un messaggio di Hello negli ultimi **dead interval** secondi

Common header (type = 1, hello)		
Network mask		
Hello interval	*Options*	*Priority*
Dead interval		
Designated router		
Backup Designated router		
Neighbor		

Hello protocol: procedure

- Regole di elezione del designated router
  - viene utilizzato il campo **priority** del pacchetto Hello
  - ogni router viene configurato staticamente con un valore di priority
    - [0,255]
  - viene selezionato il router con il più alto valore
  - i router con priorità 0 non possono essere eletti

Il protocollo Exchange

- Funzioni:
  - sincronizzazione dei database link state (bring up adjacencies) tra due router che hanno appena verificato l'operatività bidirezionale del link che li connette
  - protocollo client-server
  - messaggi:
    - Database Description Packets
    - Link State Request
    - Link State Update
  - N.B. il messaggio Link State Update viene distribuito secondo le politiche del protocollo di Flooding

Exchange Protocol: messaggi (1)

- Database Description

Common header (type = 2, db description)			
0	*0*	*Options*	*0*
DD sequence number			
Link State Type			
Link State ID			
Advertising router			
Link State Sequence Number			
Link State Checksum		*Link State Age*	

Exchange Protocol: messaggi (2)

- Link State Request

Common header (type = 3, link state request)
Link State Type
Link State ID
Advertising router

- Link state Update

Common header (type = 4, link state update)
Number of link state advertisement
Link state advertisement #1
Link state advertisement #2

Il protocollo di Flooding

- Funzioni:
  - aggiornare il database link state dell'autonomous system a seguito del cambiamento di stato di un link
- Messaggi:
  - Link State Update

Common header (type = 4, link state update)
Number of link state advertisement
Link state advertisement #1
Link state advertisement #2

Reti di Calcolatori

Esercizi su routing

Convenzioni utilizzate

- Ogni nodo invia gli update periodicamente ogni T secondi
- Tutti i nodi sono sincronizzati e iniziano a scambiarsi i distance vector (DV) a partire dal tempo t=0;
  - i successivi update vengono inviati dai diversi nodi esattamente nello stesso istante;
- Se il costo di un link cambia (ad es. se un link si guasta), il nodo aspetta il successivo invio degli update
  - non notifica immediatamente il cambiamento ai vicini
  - esempio: se il link si guasta al tempo t = 3T + T/2, l'update viene inviato al tempo t = 4T;
    - semplificazione rispetto al caso generale, dove invece si invia subito un update;
- Se un update ricevuto dai vicini fa cambiare la tabella di routing di un nodo, il nodo aspetta il successivo invio degli update per notificare tale cambiamento
  - non notifica immediatamente il cambiamento ai vicini)
  - semplificazione rispetto al caso generale, dove invece si invia subito un update;

Convenzioni utilizzate

- I nodi utilizzano gli update dei vicini per aggiornare la tabella di routing, e poi scartano l'update ricevuto
  - non tengono memoria del precedente DV ricevuto;
- Se un link si guasta, tutte le destinazioni che hanno come next-hop il nodo coinvolto vengono poste come irraggiungibili
- In definitiva:
  1. ogni nodo invia il proprio DV all'istante T, 2T, 3T, …
  2. ogni nodo riceve il DV dei vicini una frazione di tempo successiva all'istante T, 2T, 3T, …
  3. con i DV ricevuti ogni nodo aggiorna la propria tabella di routing e torna al punto 1;

Algoritmo di aggiornamento delle tabelle di routing

- Convenzione
  - c(i,j) e' il costo del link diretto tra il nodo "i" e il suo vicino "j"
  - D(i,k) e' il costo del cammino tra il nodo "i" e il nodo "k"
- Al generico nodo "i"
  - Inizializzazione
    - D(i,i) = 0 e Next-hop(i) = "i";
    - D(i,j) = c(i,j) e Next-hop(i) = "j" se "j" e' un vicino
    - D(i,k) = inf. e Next-hop(i) = - per tutti gli altri

Aggiornamento delle tabelle di routing

- Per ogni distance vector (DV) ricevuto dal nodo "j"
  - per ogni destinazione "k" contenuta nel DV
    - il nodo calcola c(i,j)+D(j,k) e lo confronta con D(i,k) della propria tabella di routing;
    - se c(i,j)+D(j,k) < D(i,k)
      - D(i,k) = c(i,j)+D(j,k) e next hop = j
    - altrimenti, se next hop == j
      - D(i,k) = c(i,j)+D(j,k)

Il nodo "i" aggiorna la propria tabella

Se arriva un update negativo, lo dobbiamo registrare

- Se il link verso il nodo "q" si guasta
  - per ogni destinazione "k" contenuta nella tabella di routing
    - altrimenti, se next hop == q
      - D(i,q) = inf.

Notazione utilizzata

Tabella di routing
(ad es. del nodo A)

da A	dist	next
A	0	A
B	5	B
C	-	-
D	7	B

nodo stesso

vicino

sconosciuto

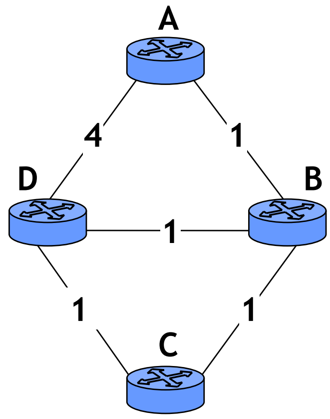
raggiungibile da altri nodi

Distance Vector
(ad es. inviato da A)

da A	dist
A	1
B	5
C	-
D	2

Esercizio 1

- Con riferimento alla rete in figura, ove e' utilizzato l'algoritmo Distributed Bellman-Ford (DBF) classico senza alcun meccanismo aggiuntivo
  - Si indichi quale sarà la tabella di routing dei diversi nodi a regime
  - Si mostrino i messaggi scambiati nel caso in cui il link tra A e D si guasti
  - Si mostrino i messaggi scambiati nel caso in cui il link tra A e B si guasti
  - Nel caso in cui l'algoritmo implementi split-horizon con poison-reverse, si mostrino i messaggi scambiati nel caso in cui il link tra A e B si guasti



Esercizio 1 - Soluzione

Tabelle a regime

da A	dist	next
A	0	A
B	1	B
C	2	B
D	2	B

da B	dist	next
A	1	A
B	0	B
C	1	C
D	1	D

da C	dist	next
A	2	B
B	1	B
C	0	C
D	1	D

da D	dist	next
A	2	B
B	1	B
C	1	C
D	0	D

dopo il guasto del link A-D ➔ nessun cambiamento

Tabelle subito dopo il guasto del link A-B

da A	dist	next
A	0	A
B	inf	-
C	inf	-
D	inf	-

da B	dist	next
A	inf	-
B	0	B
C	1	C
D	1	D

Tabelle dopo il guasto | Distance Vector ricevuti dai vicini | Tabelle dopo l'iterazione

Tabelle dopo il guasto

A

da A	dist	next
A	0	A
B	inf	-
C	inf	-
D	inf	-

B

da B	dist	next
A	inf	-
B	0	B
C	1	C
D	1	D

C

da C	dist	next
A	2	B
B	1	B
C	0	C
D	1	D

D

da D	dist	next
A	2	B
B	1	B
C	1	C
D	0	D

Distance Vector ricevuti dai vicini

A

da D	dist
A	2
B	1
C	1
D	0

B

da C	dist
A	2
B	1
C	0
D	1

da D	dist
A	2
B	1
C	1
D	0

C

da B	dist
A	inf
B	0
C	1
D	1

da D	dist
A	2
B	1
C	1
D	0

D

da A	dist
A	0
B	inf
C	inf
D	inf

da B	dist
A	inf
B	0
C	1
D	1

da C	dist
A	2
B	1
C	0
D	1

Tabelle dopo l'iterazione

A

da A	dist	next
A	0	A
B	5	D
C	5	D
D	4	D

B

da B	dist	next
A	3	C
B	0	B
C	1	C
D	1	D

poteva scegliere anche D

C

da C	dist	next
A	3	D
B	1	B
C	0	C
D	1	D

D

da D	dist	next
A	3	C
B	1	B
C	1	C
D	0	D

Distance Vector ricevuti dai vicini alla successiva iterazione | Tabelle dopo l'iterazione

Tabelle

A

da A	dist	next
A	0	A
B	5	D
C	5	D
D	4	D

B

da B	dist	next
A	3	C
B	0	B
C	1	C
D	1	D

C

da C	dist	next
A	3	D
B	1	B
C	0	C
D	1	D

D

da D	dist	next
A	3	C
B	1	B
C	1	C
D	0	D

Distance Vector ricevuti dai vicini alla successiva iterazione

A

da D	dist
A	3
B	1
C	1
D	0

B

da C	dist
A	3
B	1
C	0
D	1

da D	dist
A	3
B	1
C	1
D	0

C

da B	dist
A	3
B	0
C	1
D	1

da D	dist
A	3
B	1
C	1
D	0

D

da A	dist
A	0
B	5
C	5
D	4

da B	dist
A	3
B	0
C	1
D	1

da C	dist
A	3
B	1
C	0
D	1

Tabelle dopo l'iterazione

A

da A	dist	next
A	0	A
B	5	D
C	5	D
D	4	D

B

da B	dist	next
A	4	C
B	0	B
C	1	C
D	1	D

poteva scegliere anche D

C

da C	dist	next
A	4	D
B	1	B
C	0	C
D	1	D

poteva scegliere anche B

D

da D	dist	next
A	4	A
B	1	B
C	1	C
D	0	D

predilige il colleg. diretto

Distance Vector ricevuti dai vicini alla successiva iterazione

Tabelle dopo l'iterazione

A

da A	dist	next
A	0	A
B	5	D
C	5	D
D	4	D

da D	dist
A	4
B	1
C	1
D	0

da A	dist	next
A	0	A
B	5	D
C	5	D
D	4	D

B

da B	dist	next
A	4	C
B	0	B
C	1	C
D	1	D

da C	dist
A	4
B	1
C	0
D	1

da D	dist
A	4
B	1
C	1
D	0

da B	dist	next
A	5	C
B	0	B
C	1	C
D	1	D

poteva scegliere anche D

C

da C	dist	next
A	4	D
B	1	B
C	0	C
D	1	D

da B	dist
A	4
B	0
C	1
D	1

da D	dist
A	4
B	1
C	1
D	0

da C	dist	next
A	5	D
B	1	B
C	0	C
D	1	D

poteva scegliere anche B

D

da D	dist	next
A	4	A
B	1	B
C	1	C
D	0	D

da A	dist
A	0
B	5
C	5
D	4

da B	dist
A	4
B	0
C	1
D	1

da C	dist
A	4
B	1
C	0
D	1

da D	dist	next
A	4	A
B	1	B
C	1	C
D	0	D

Distance Vector ricevuti dai vicini alla successiva iterazione

Tabelle dopo l'iterazione

A

da A	dist	next
A	0	A
B	5	D
C	5	D
D	4	D

da D	dist
A	4
B	1
C	1
D	0

da A	dist	next
A	0	A
B	5	D
C	5	D
D	4	D

B

da B	dist	next
A	5	C
B	0	B
C	1	C
D	1	D

da C	dist
A	5
B	1
C	0
D	1

da D	dist
A	4
B	1
C	1
D	0

da B	dist	next
A	5	D
B	0	B
C	1	C
D	1	D

deve scegliere D

C

da C	dist	next
A	5	D
B	1	B
C	0	C
D	1	D

da B	dist
A	5
B	0
C	1
D	1

da D	dist
A	4
B	1
C	1
D	0

da C	dist	next
A	5	D
B	1	B
C	0	C
D	1	D

poteva scegliere solo D

D

da D	dist	next
A	4	A
B	1	B
C	1	C
D	0	D

da A	dist
A	0
B	5
C	5
D	4

da B	dist
A	5
B	0
C	1
D	1

da C	dist
A	5
B	1
C	0
D	1

da D	dist	next
A	4	A
B	1	B
C	1	C
D	0	D

Split horizon con poison reverse

- In questo caso, i Distance Vector inviati da “i” a “j” contengono esplicitamente un valore pari ad infinito nelle righe in cui “i” ha come next hop “j”
- Esempio

Tabella del nodo C

da C	dist	next
A	2	B
B	1	B
C	0	C
D	1	D

DV inviato da C a B

da C	dist
A	inf
B	inf
C	0
D	1

Tabelle dopo il guasto	Distance Vector ricevuti dai vicini	Tabelle dopo l’iterazione

A

Tabella dopo il guasto:

da A	dist	next
A	0	A
B	inf	-
C	inf	-
D	inf	-

Distance Vector ricevuti:

da D	dist
A	2
B	1
C	1
D	0

Tabella dopo l’iterazione:

da A	dist	next
A	0	A
B	5	D
C	5	D
D	4	D

B

Tabella dopo il guasto:

da B	dist	next
A	inf	-
B	0	B
C	1	C
D	1	D

Distance Vector ricevuti:

da C	dist
A	inf
B	inf
C	0
D	1

da D	dist
A	inf
B	inf
C	1
D	0

Tabella dopo l’iterazione:

da B	dist	next
A	inf	-
B	0	B
C	1	C
D	1	D

C

Tabella dopo il guasto:

da C	dist	next
A	2	B
B	1	B
C	0	C
D	1	D

Distance Vector ricevuti:

da B	dist
A	inf
B	0
C	inf
D	1

da D	dist
A	2
B	1
C	inf
D	0

Tabella dopo l’iterazione:

da C	dist	next
A	3	D
B	1	B
C	0	C
D	1	D

D

Tabella dopo il guasto:

da D	dist	next
A	2	B
B	1	B
C	1	C
D	0	D

Distance Vector ricevuti:

da A	dist
A	0
B	inf
C	inf
D	inf

da B	dist
A	inf
B	0
C	1
D	inf

da C	dist
A	2
B	1
C	0
D	inf

Tabella dopo l’iterazione:

da D	dist	next
A	3	C
B	1	B
C	1	C
D	0	D

Distance Vector ricevuti dai vicini alla successiva iterazione

Tabelle dopo l'iterazione

A

da A	dist	next
A	0	A
B	5	-
C	5	-
D	4	-

da D	dist
A	2
B	1
C	1
D	0

da A	dist	next
A	0	A
B	5	D
C	5	D
D	4	D

B

da B	dist	next
A	inf	-
B	0	B
C	1	C
D	1	D

da C	dist
A	3
B	inf
C	0
D	1

da D	dist
A	3
B	inf
C	1
D	0

da B	dist	next
A	4	C
B	0	B
C	1	C
D	1	D

C

da C	dist	next
A	3	D
B	1	B
C	0	C
D	1	D

da B	dist
A	inf
B	0
C	inf
D	1

da D	dist
A	inf
B	1
C	inf
D	0

da C	dist	next
A	inf	-
B	1	B
C	0	C
D	1	D

D

da D	dist	next
A	3	C
B	1	B
C	1	C
D	0	D

da A	dist
A	0
B	inf
C	inf
D	inf

da B	dist
A	inf
B	0
C	1
D	inf

da C	dist
A	inf
B	1
C	0
D	inf

da D	dist	next
A	4	A
B	1	B
C	1	C
D	0	D

Distance Vector ricevuti dai vicini alla successiva iterazione

Tabelle dopo l'iterazione

A

da A	dist	next
A	0	A
B	5	D
C	5	D
D	4	D

da D	dist
A	2
B	1
C	1
D	0

da A	dist	next
A	0	A
B	5	D
C	5	D
D	4	D

B

da B	dist	next
A	4	C
B	0	B
C	1	C
D	1	D

da C	dist
A	inf
B	inf
C	0
D	1

da D	dist
A	4
B	inf
C	1
D	0

da B	dist	next
A	5	D
B	0	B
C	1	C
D	1	D

C

da C	dist	next
A	inf	-
B	1	B
C	0	C
D	1	D

da B	dist
A	inf
B	0
C	inf
D	1

da D	dist
A	4
B	1
C	inf
D	0

da C	dist	next
A	5	D
B	1	B
C	0	C
D	1	D

D

da D	dist	next
A	4	A
B	1	B
C	1	C
D	0	D

da A	dist
A	0
B	inf
C	inf
D	inf

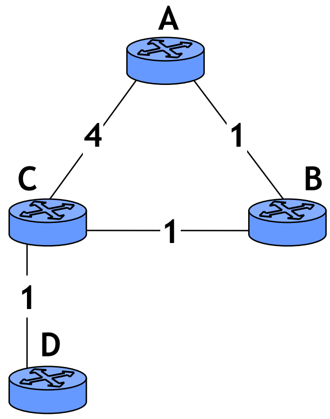
da B	dist
A	4
B	0
C	1
D	inf

da C	dist
A	inf
B	1
C	0
D	inf

da D	dist	next
A	4	A
B	1	B
C	1	C
D	0	D

Esercizio 2

- Con riferimento alla rete in figura, ove e' utilizzato l'algoritmo Distributed Bellman-Ford (DBF) classico senza alcun meccanismo aggiuntivo
  - Si indichi quale sarà la tabella di routing dei diversi nodi a regime
  - Si mostrino i messaggi scambiati nel caso in cui il link tra A e C cambi costo, da 4 a 1
  - Si mostrino i messaggi scambiati nel caso in cui il link tra C e D si guasti (evento successivo al cambio del costo del link A-C da 4 a 1)
  - Nel caso in cui l'algoritmo implementi split-horizon con poison-reverse, si mostrino i messaggi scambiati nel caso in cui il link tra C e D si guasti (evento successivo al cambio del costo del link A-C da 4 a 1)



Esercizio 2 - Soluzione

Tabelle a regime

da A	dist	next
A	0	A
B	1	B
C	2	B
D	3	B

da B	dist	next
A	1	A
B	0	B
C	1	C
D	2	C

da C	dist	next
A	2	B
B	1	B
C	0	C
D	1	D

da D	dist	next
A	3	C
B	2	C
C	1	C
D	0	D

Tabelle subito dopo il cambio di costo del link A-C

da A	dist	next
A	0	A
B	1	B
C	1	C
D	3	B

da C	dist	next
A	1	A
B	1	B
C	0	C
D	1	D

Tabelle dopo il guasto — Distance Vector ricevuti dai vicini — Tabelle dopo l'iterazione

A

Tabelle dopo il guasto:

da A	dist	next
A	0	A
B	1	B
C	1	C
D	3	B

Distance Vector ricevuti dai vicini:

da B	dist
A	1
B	0
C	1
D	2

da C	dist
A	1
B	1
C	0
D	1

Tabelle dopo l'iterazione:

da A	dist	next
A	0	A
B	1	B
C	1	C
D	2	C

B

Tabelle dopo il guasto:

da B	dist	next
A	1	A
B	0	B
C	1	C
D	2	C

Distance Vector ricevuti dai vicini:

da A	dist
A	0
B	1
C	1
D	3

da C	dist
A	1
B	1
C	0
D	1

Tabelle dopo l'iterazione:

da B	dist	next
A	1	A
B	0	B
C	1	C
D	2	C

C

Tabelle dopo il guasto:

da C	dist	next
A	1	A
B	1	B
C	0	C
D	1	D

Distance Vector ricevuti dai vicini:

da A	dist
A	0
B	1
C	1
D	3

da B	dist
A	1
B	0
C	1
D	2

Tabelle dopo l'iterazione:

da C	dist	next
A	1	A
B	1	B
C	0	C
D	1	D

D

Tabelle dopo il guasto:

da D	dist	next
A	3	C
B	2	C
C	1	C
D	0	D

Distance Vector ricevuti dai vicini:

da C	dist
A	1
B	1
C	0
D	1

Tabelle dopo l'iterazione:

da D	dist	next
A	2	C
B	2	C
C	1	C
D	0	D

Dopo un'iterazione siamo gia' a regime!
(le tabelle sono stabili)

Tabelle subito dopo il cambio il guasto del link C-D

da C	dist	next
A	1	A
B	1	B
C	0	C
D	inf	-

da D	dist	next
A	inf	-
B	inf	-
C	inf	-
D	0	D

Tabelle dopo il guasto — Distance Vector ricevuti dai vicini — Tabelle dopo l'iterazione — Distance Vector ricevuti dai vicini — Tabelle dopo l'iterazione

A

Tabelle dopo il guasto:

da A	dist	next
A	0	A
B	1	B
C	1	C
D	2	C

Distance Vector ricevuti dai vicini:

da B	dist
A	1
B	0
C	1
D	2

da C	dist
A	1
B	1
C	0
D	inf

Tabelle dopo l'iterazione:

da A	dist	next
A	0	A
B	1	B
C	1	C
D	3	B

Distance Vector ricevuti dai vicini:

da B	dist
A	1
B	0
C	1
D	3

da C	dist
A	1
B	1
C	0
D	3

Tabelle dopo l'iterazione:

da A	dist	next
A	0	A
B	1	B
C	1	C
D	4	B

B

Tabelle dopo il guasto:

da B	dist	next
A	1	A
B	0	B
C	1	C
D	2	C

Distance Vector ricevuti dai vicini:

da A	dist
A	0
B	1
C	1
D	2

da C	dist
A	1
B	1
C	0
D	inf

Tabelle dopo l'iterazione:

da B	dist	next
A	1	A
B	0	B
C	1	C
D	3	A

Distance Vector ricevuti dai vicini:

da A	dist
A	0
B	1
C	1
D	3

da C	dist
A	1
B	1
C	0
D	3

Tabelle dopo l'iterazione:

da B	dist	next
A	1	A
B	0	B
C	1	C
D	4	A

C

Tabelle dopo il guasto:

da C	dist	next
A	1	A
B	1	B
C	0	C
D	inf	-

Distance Vector ricevuti dai vicini:

da A	dist
A	0
B	1
C	1
D	2

da B	dist
A	1
B	0
C	1
D	2

Tabelle dopo l'iterazione:

da C	dist	next
A	1	A
B	1	B
C	0	C
D	3	A

poteva scegliere anche B

Distance Vector ricevuti dai vicini:

da A	dist
A	0
B	1
C	1
D	3

da B	dist
A	1
B	0
C	1
D	3

Tabelle dopo l'iterazione:

da C	dist	next
A	1	A
B	1	B
C	0	C
D	4	A

Distance Vector ricevuti dai vicini | Tabelle dopo l'iterazione | Distance Vector ricevuti dai vicini | Tabelle dopo l'iterazione

A

da A	dist	next
A	0	A
B	1	B
C	1	C
D	4	B

Distance Vector ricevuti dai vicini:

da B	dist
A	1
B	0
C	1
D	4

da C	dist
A	1
B	1
C	0
D	4

Tabelle dopo l'iterazione:

da A	dist	next
A	0	A
B	1	B
C	1	C
D	5	B

Distance Vector ricevuti dai vicini:

da B	dist
A	1
B	0
C	1
D	5

da C	dist
A	1
B	1
C	0
D	4

Tabelle dopo l'iterazione:

da A	dist	next
A	0	A
B	1	B
C	1	C
D	6	B

B

da B	dist	next
A	1	A
B	0	B
C	1	C
D	4	A

Distance Vector ricevuti dai vicini:

da A	dist
A	0
B	1
C	1
D	4

da C	dist
A	1
B	1
C	0
D	4

Tabelle dopo l'iterazione:

da B	dist	next
A	1	A
B	0	B
C	1	C
D	5	A

Distance Vector ricevuti dai vicini:

da A	dist
A	0
B	1
C	1
D	5

da C	dist
A	1
B	1
C	0
D	4

Tabelle dopo l'iterazione:

da B	dist	next
A	1	A
B	0	B
C	1	C
D	6	A

C

da C	dist	next
A	1	A
B	1	B
C	0	C
D	4	A

Distance Vector ricevuti dai vicini:

da A	dist
A	0
B	1
C	1
D	4

da B	dist
A	1
B	0
C	1
D	4

Tabelle dopo l'iterazione:

da C	dist	next
A	1	A
B	1	B
C	0	C
D	5	A

Distance Vector ricevuti dai vicini:

da A	dist
A	0
B	1
C	1
D	5

da B	dist
A	1
B	0
C	1
D	5

Tabelle dopo l'iterazione:

da C	dist	next
A	1	A
B	1	B
C	0	C
D	6	A

…all'infinito

Soluzione con Split Horizon (+ poison reverse)

Tabelle dopo il guasto | Distance Vector ricevuti dai vicini | Tabelle dopo l'iterazione | Distance Vector ricevuti dai vicini | Tabelle dopo l'iterazione

A

Tabelle dopo il guasto:

da A	dist	next
A	0	A
B	1	B
C	1	C
D	2	C

Distance Vector ricevuti dai vicini:

da B	dist
A	inf
B	0
C	1
D	2

da C	dist
A	inf
B	1
C	0
D	inf

Tabelle dopo l'iterazione:

da A	dist	next
A	0	A
B	1	B
C	1	C
D	3	B

Distance Vector ricevuti dai vicini:

da B	dist
A	inf
B	0
C	1
D	inf

da C	dist
A	1
B	1
C	0
D	inf

Tabelle dopo l'iterazione:

da A	dist	next
A	0	A
B	1	B
C	1	C
D	inf	-

B

Tabelle dopo il guasto:

da B	dist	next
A	1	A
B	0	B
C	1	C
D	2	C

Distance Vector ricevuti dai vicini:

da A	dist
A	0
B	inf
C	1
D	2

da C	dist
A	1
B	inf
C	0
D	inf

Tabelle dopo l'iterazione:

da B	dist	next
A	1	A
B	0	B
C	1	C
D	3	A

Distance Vector ricevuti dai vicini:

da A	dist
A	0
B	inf
C	1
D	inf

da C	dist
A	1
B	1
C	0
D	inf

Tabelle dopo l'iterazione:

da B	dist	next
A	1	A
B	0	B
C	1	C
D	inf	-

C

Tabelle dopo il guasto:

da C	dist	next
A	1	A
B	1	B
C	0	C
D	inf	-

Distance Vector ricevuti dai vicini:

da A	dist
A	0
B	1
C	inf
D	inf

da B	dist
A	1
B	0
C	inf
D	inf

Tabelle dopo l'iterazione:

da C	dist	next
A	1	A
B	1	B
C	0	C
D	inf	-

Distance Vector ricevuti dai vicini:

da A	dist
A	0
B	1
C	1
D	3

da B	dist
A	1
B	0
C	1
D	3

Tabelle dopo l'iterazione:

da C	dist	next
A	1	A
B	1	B
C	0	C
D	4	A

Soluzione con Split Horizon (+ poison reverse)

Distance Vector ricevuti dai vicini — Tabelle dopo l'iterazione

A

da A	dist	next	da B	dist	da C	dist	da A	dist	next
A	0	A	A	inf	A	inf	A	0	A
B	1	B	B	0	B	1	B	1	B
C	1	C	C	1	C	0	C	1	C
D	inf	-	D	inf	D	4	D	5	C

B

da B	dist	next	da A	dist	da C	dist	da B	dist	next
A	1	A	A	0	A	1	A	1	A
B	0	B	B	inf	B	inf	B	0	B
C	1	C	C	1	C	0	C	1	C
D	inf	-	D	inf	D	4	D	5	C

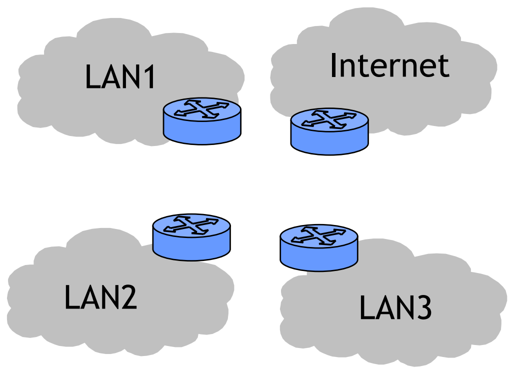
C

da C	dist	next	da A	dist	da B	dist	da C	dist	next
A	1	A	A	0	A	1	A	1	A
B	1	B	B	1	B	0	B	1	B
C	0	C	C	inf	C	inf	C	0	C
D	4	A	D	inf	D	inf	D	inf	-

Siamo tornati al punto di partenza, con la distanza verso D uguale a 5 invece che uguale a 2! Anche qui l'iterazione procede all'infinito

Esercizio 3

- Si consideri la rete rappresentata in figura a lato,
  - i quattro router (RA, RB, RC e RD) sono connessi tra loro da canali punto-punto;
  - sulla rete è in funzione un protocollo di routing di tipo Distance Vector che implementa split-horizon con poison-reverse,
  - La metrica utilizzata e' il numero di hop;
  - i distance-vector inviati dai router RA/B/C/D su ciascuna delle loro interfacce sono



	Router A		Router B			Router C		Router D		
	Interf-1	Interf-2	Interf-1	Interf-2	Interf-3	Interf-1	Interf-2	Interf-1	Interf-2	Interf-3
→ LAN 1	inf	1	2	inf	2	4	inf	3	3	inf
→ LAN 2	2	inf	inf	1	1	3	inf	2	2	inf
→ LAN 3	4	inf	3	3	inf	inf	1	2	inf	2
→Internet	3	inf	2	2	inf	2	inf	inf	1	1

Esercizio 3 (cnt'd)

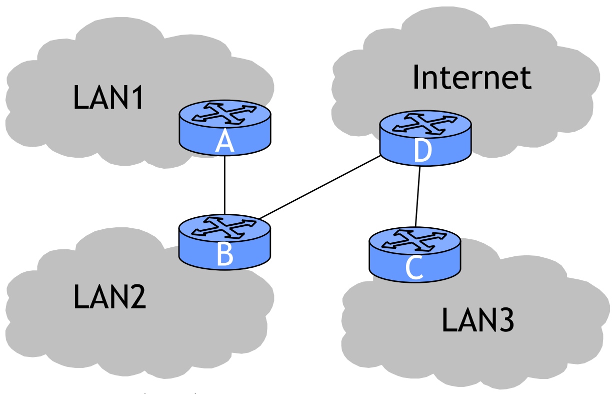
- Domande:
  - Si disegni la topologia del backbone.
  - Si scrivano le tabelle di routing dei router RA/B/C/D.
  - Si dica se in caso di guasti ad uno qualsiasi dei canali punto-punto si possano verificare dei routing loop. Se si, se ne specifichi la natura (permanenti, transitori, ...). Si motivi la risposta.

Esercizio 3: soluzione

- Router A
  - Interf.1: da A a LAN1
  - Interf.2: da A a B
- Router B
  - Interf.1: da B a LAN2
  - Interf.2: da B a A
  - Interf.3: da B a D
- Router C
  - Interf.1: da C a LAN3
  - Interf.2: da C a D
- Router D
  - Interf.1: da D a Internet
  - Interf.2: da D a B
  - Interf.3: da D a C
- In caso di guasti ai link, la rete viene partizionata, per cui ci saranno sicuramente dei routing loop permanenti (vedi esercizio precedente)

da A	dist	next
LAN1	1	dir
LAN2	2	B
LAN3	4	B
Internet	3	B

da D	dist	next
LAN1	3	B
LAN2	2	B
LAN3	2	C
Internet	1	dir



da B	dist	next
LAN1	2	A
LAN2	1	dir
LAN3	3	D
Internet	2	D

da C	dist	next
LAN1	4	D
LAN2	3	D
LAN3	1	dir
Internet	2	D