

## Distance vector protocols

Distance Vector routing principles  
Routing loops and countermeasures to loops  
Bellman-Ford algorithm  
RIP, RIP-2

## Distance Vector Routing Principles

## Distance Vector Routing

- Distance vector (DV) protocols are based on the Bellman-Ford algorithm
- The routing table contains information about other known nodes
  - link (interface) identifier
  - distance (cost) in hops
- The nodes periodically send distance vectors based on the routing tables on all their links
- The nodes update their routing table with received distance vectors
- Routing Information Protocol (RIP) is a basic distance vector protocol

Routing table:

E to	Link	Distance
E	-	0
B	4	1
A	4	2
D	6	1
C	5	1

Distance vector:

E=0, B=1, A=2, D=1, C=1

## Let us study the principles of DV protocols

Example network with nodes A, B, C, D, E and links 1, 2, 3, 4, 5, 6.

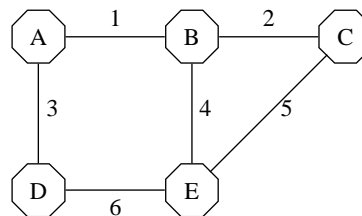
The cost of each link is 1.

Initial state: Nodes know their own addresses and interfaces, nothing more.

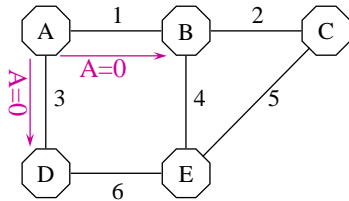
Node A creates its routing table:

From node A to ...	Link	Distance
A	- (local)	0

The corresponding distance vector (DV) is: A=0



## Generation of routing tables starts when all routers send their DVs on all interfaces



Let's look at reception in Node B. First the table of B is:

From node B to ...	Link	Distance
B	-	0

1. B receives the distance vector  $A=0$
2. B *increments the DV with +1*  $\Rightarrow A=1$
3. B looks for the result in its routing table, no match
4. B adds the result to its RT, the result is

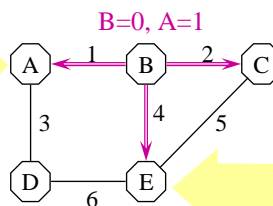
From node B to ...	Link	Distance
B	-	0
A	1	1

5. B generates its distance vector  $B=0, A=1$

## B creates its own DV and sends it to all neighbors

A to	Link	Distance
A	-	0
B	1	1

$A=2 > A=0$

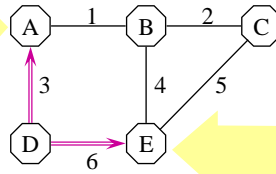


C to	Link	Distance
C	-	0
B	2	1
A	2	2

E to	Link	Distance
E	-	0
B	4	1
A	4	2

## D sends its distance vector to all neighbors

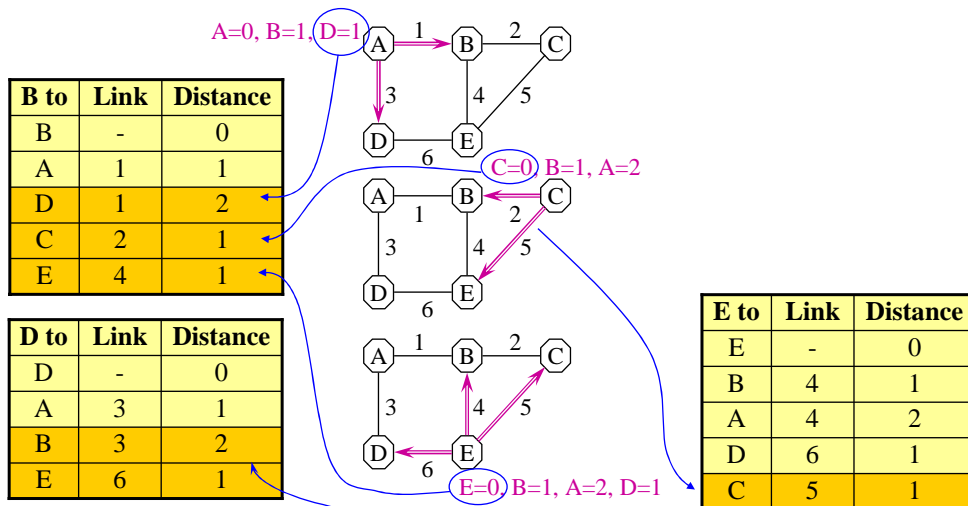
A to	Link	Distance
A	-	0
B	1	1
D	3	1



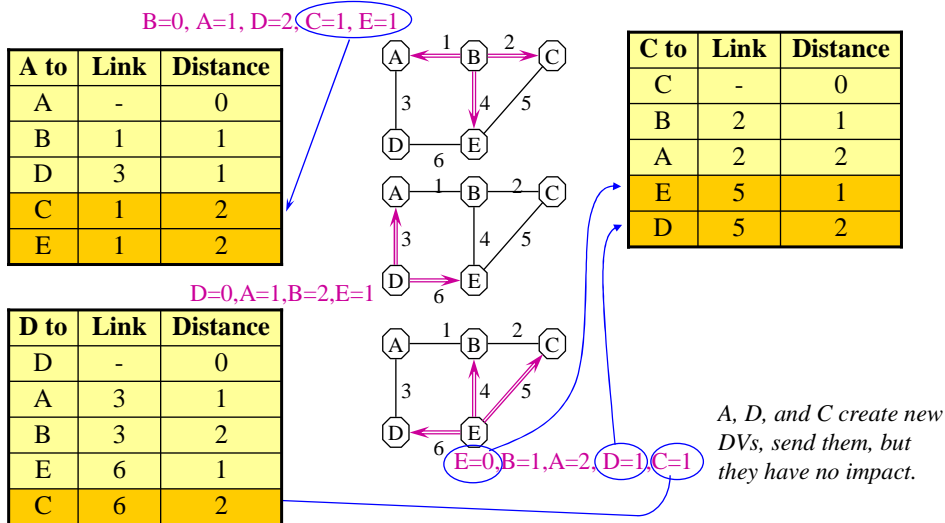
E to	Link	Distance
E	-	0
B	4	1
A	4	2
D	6	1

A=2 == A=2 ⇒ no change

## The nodes whose RT changed create DVs and send them to neighbors



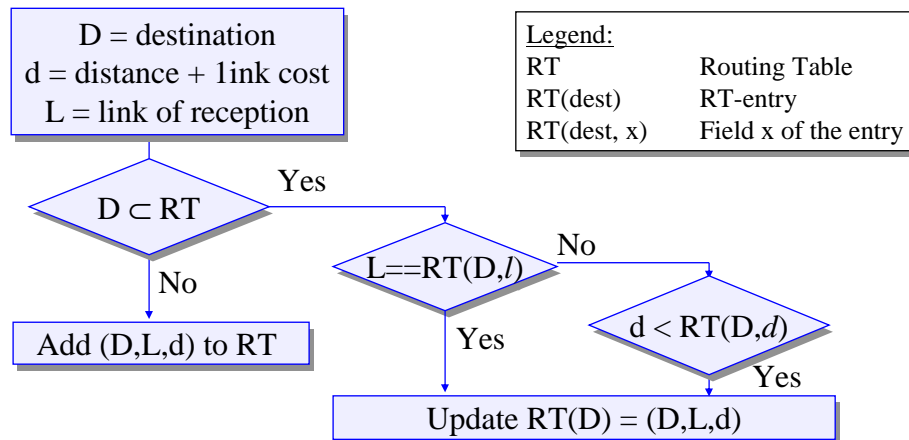
## Again the changes are sent ...



S-38.2121 / Fall-06 / RKa, NB

DV-9

## Processing of received distance vectors



*Note: this is simplified, shows only the principle!*

S-38.2121 / Fall-06 / RKa, NB

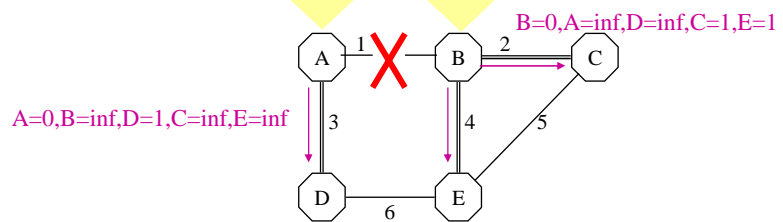
DV-10

## A link breaks...

## A round of updates starts on link failure

*A gives an infinite distance to the nodes reached through link 1*

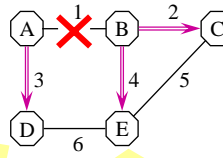
A to	Link	Distance	B to	Link	Distance
A	-	0	B	-	0
B	1	Inf.	A	1	Inf.
D	3	1	D	1	Inf.
C	1	Inf.	C	2	1
E	1	Inf.	E	4	1



## D, E and C update their routing tables

A=0,B=inf,D=1,C=inf,E=inf  
+1 =  
A=1,B=inf,D=2,C=inf,E=inf

B=0,A=inf,D=inf,C=1,E=1



C to	Link	Distance
C	-	0
B	2	1
A	2	Inf.
E	5	1
D	5	2

D to	Link	Distance
D	-	0
A	3	1
B	3	Inf.
E	6	1
C	6	2

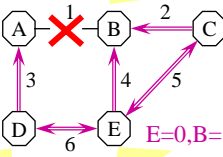
E to	Link	Distance
E	-	0
B	4	1
A	4	Inf.
D	6	1
C	5	1

if  $L == RT(D, l)$  the routing table is updated even if the new distance is longer! (see "processing of received distance vectors")

## D, C, E generate their distance vectors...

A to	Link	Distance
A	-	0
B	1	Inf.
D	3	1
C	3	3
E	3	2

B to	Link	Distance
B	-	0
A	1	Inf.
D	4	2
C	2	1
E	4	1



C=0,B=1,A=inf,E=1,D=2

D=0,A=1,B=inf,E=1,C=2

E=0,B=1,A=inf,D=1,C=1

D to	Link	Distance
D	-	0
A	3	1
B	6	2
E	6	1
C	6	2

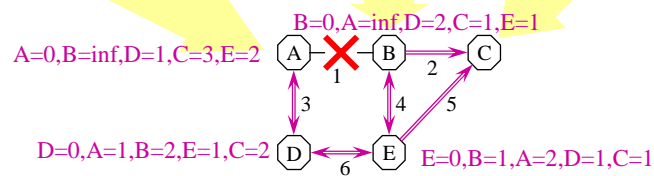
E to	Link	Distance
E	-	0
B	4	1
A	6	2
D	6	1
C	5	1

## A, B, D, E generate their distance vectors

A to	Link	Distance
A	-	0
B	3	3
D	3	1
C	3	3
E	3	2

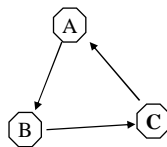
B to	Link	Distance
B	-	0
A	4	3
D	4	2
C	2	1
E	4	1

C to	Link	Distance
C	-	0
B	2	1
A	5	3
E	5	1
D	5	2



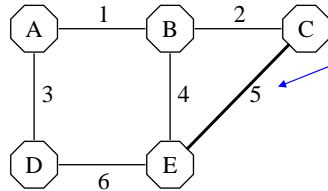
The result is that all nodes are able to communicate with all other nodes again.

## Routing loops





## The DV-protocol may create a transient routing loop



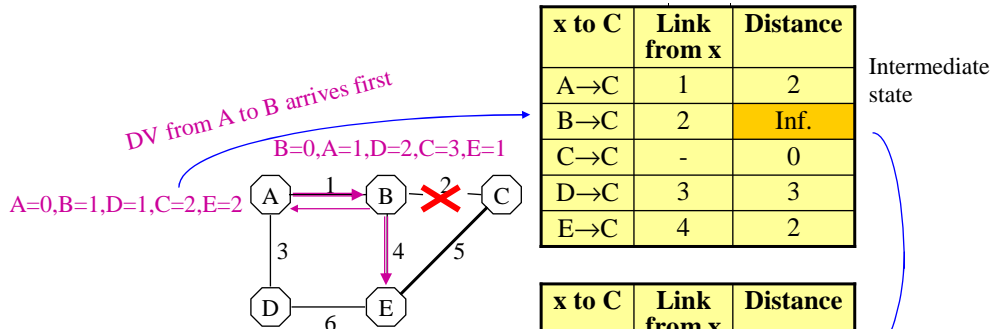
Let's assume that cost of link 5 is 8.

A stable initial state for routes to C would be:

x to C	Link from x	Distance
A→C	1	2
B→C	2	1
C→C	-	0
D→C	3	3
E→C	4	2

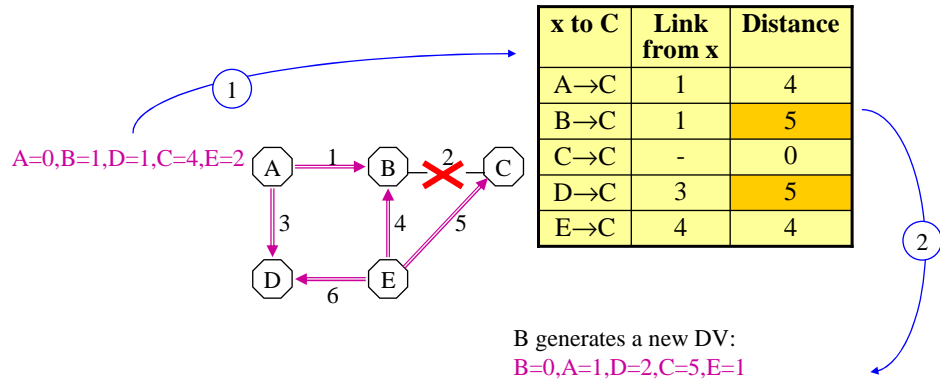
Let's just look at the first link of each route.

## Link 2 fails



All packets to C are sent to B.  
B sends them to A. A sends them back to B... until TTL=0.  
(Bouncing effect)

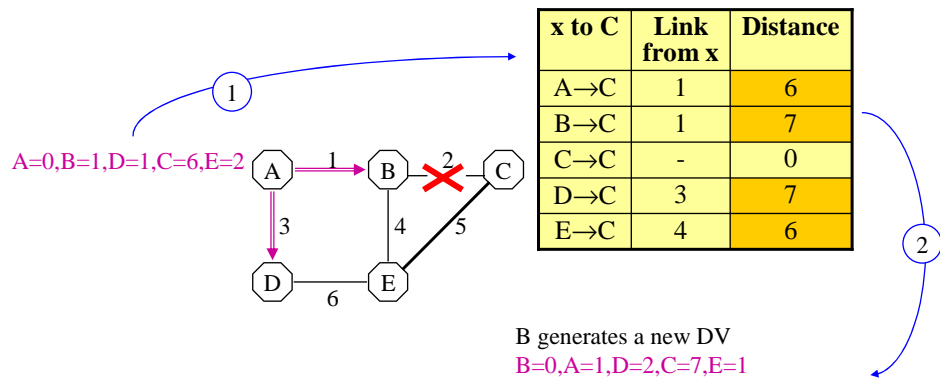
## A and E send their distance vectors



⇒ Distance seen by A to C grows to 6

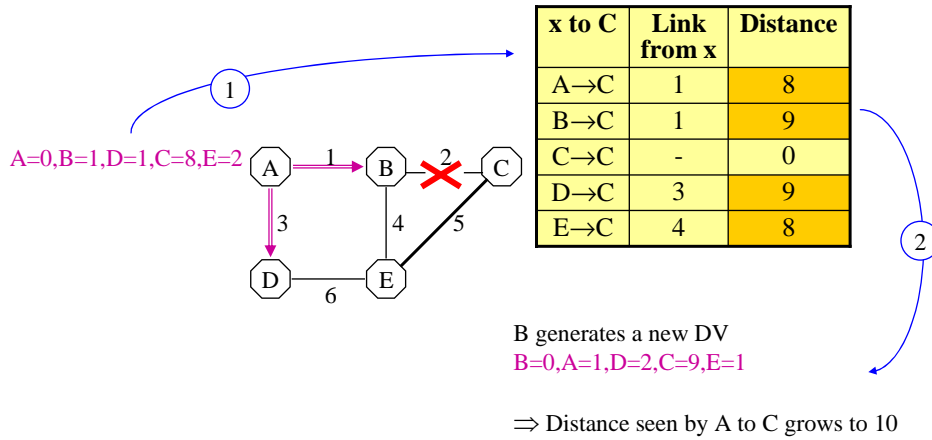
Distance vectors sent by C do not change anything because of high link cost  
 DV-19

## A sends a new distance vector

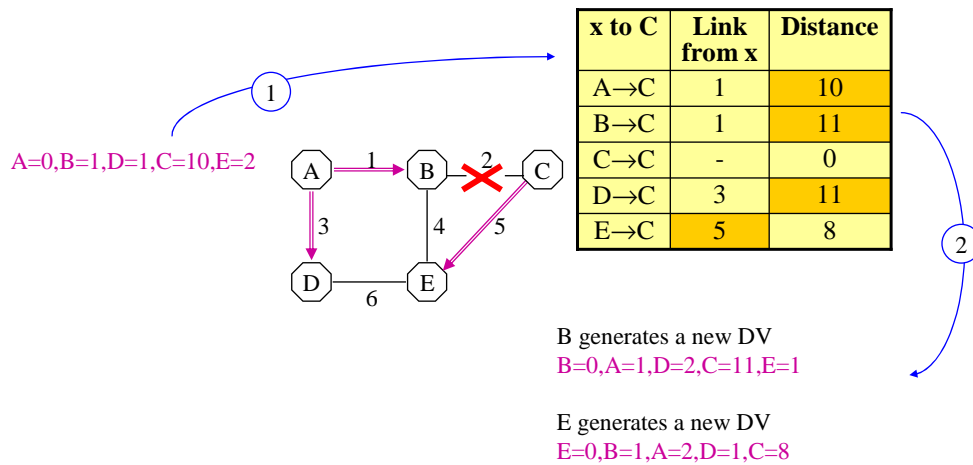


⇒ Distance seen by A to C grows to 8

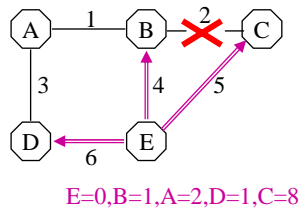
## A sends a new distance vector



## A sends a new distance vector

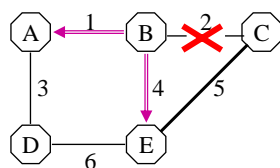


## E sends a new distance vector



x to C	Link from x	Distance
A→C	1	10
B→C	4	9
C→C	-	0
D→C	6	9
E→C	5	8

## B send its DV but the tables are already OK



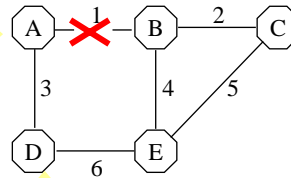
x to C	Link from x	Distance
A→C	1	10
B→C	4	9
C→C	-	0
D→C	6	9
E→C	5	8

- Each update round increased the costs by 2
- The process progresses in a random order, because it is genuinely parallel in nature.
- During the process, the state of the network is bad. DV-packets may be lost due to the overload created by bouncing user messages

## Counting to infinity occurs when failures break the network into isolated islands (1)

- Link 1 is broken, and the network has recovered.
- All link costs = 1

A to	Link	Distance
D	3	1
A	-	0
B	3	3
E	3	2
C	3	3

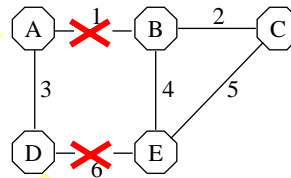


D to	Link	Distance
D	-	0
A	3	1
B	6	2
E	6	1
C	6	2

## Counting to infinity occurs when failures break the network into isolated islands (2)

- Also link 6 breaks.
- D updates its routing table but has not yet sent its distance vector.

A to	Link	Distance
D	3	1
A	-	0
B	3	3
E	3	2
C	3	3

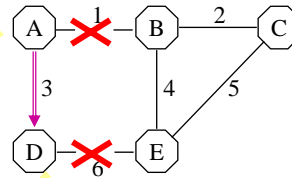


D to	Link	Distance
D	-	0
A	3	1
B	6	Inf.
E	6	Inf.
C	6	Inf.

## Counting to infinity occurs when failures break the network into isolated islands (3)

- A sends its distance vector first:  
A=0,B=3,D=1,C=3,E=2
- D adds the information sent by A into its routing table.

A to	Link	Distance
D	3	1
A	-	0
B	3	3
E	3	2
C	3	3

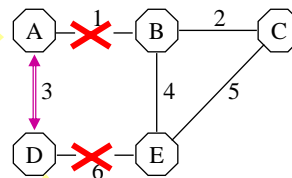


D to	Link	Distance
D	-	0
A	3	1
B	3	4
E	3	3
C	3	4

## Counting to infinity occurs when failures break the network into isolated islands (4)

- The result is a loop. Costs are incremented by 2 on each round.
- We need to define infinity as a cost greater than any normal route cost.

A to	Link	Distance
D	3	1
A	-	0
B	3	5
E	3	4
C	3	5



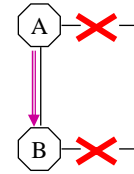
D to	Link	Distance
D	-	0
A	3	1
B	3	4
E	3	3
C	3	4

## The first method to avoid loops is to send less information

### The split horizon rule:

If node A sends to node X through node B, it does not make sense for B to try to reach X through A

⇒ A should not advertise to B its short distance to X



### Implementation choices:

1. Split horizon
  - A does not advertise its distance to X towards B at all
  - ⇒ the loop of previous example can not occur
2. Split horizon with poisonous reverse
  - A advertises to B:  $X = \text{inf}$ .
  - ⇒ two node loops are killed immediately

## Split horizon example

A to	Link	Distance
B	3	1
A	-	0
D	3	2
C	1	1

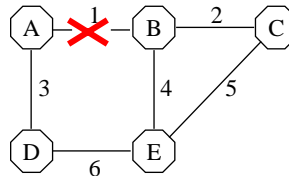
- Normally:
  - A sends:  $A=0, B=1, C=1, D=2$
- Split horizon:
  - A sends:  $A=0, C=1$
- Split horizon with poisonous reverse:
  - A sends:  $A=0, B=\text{inf}, C=1, D=\text{inf}$ ,

*Note that A sends different DVs on link 1*

*→ 2-node loops are impossible!*

## Three-node loops are still possible (1)

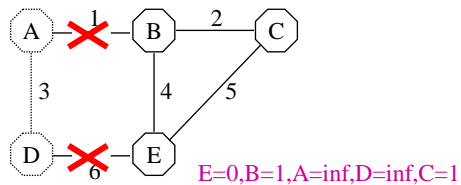
- Link 1 is broken, and the network has recovered.
- All link costs = 1



x to D	Link from x	Distance
B→D	4	2
C→D	5	2
E→D	6	1

## Three-node loops are still possible (2)

- Also link 6 fails.
- E sends its distance vector to B and C  
 $E=0, B=1, A=\text{inf}, D=\text{inf}, C=1$

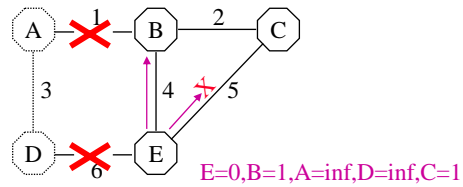


x to D	Link from x	Distance
B→D	4	2
C→D	5	2
E→D	6	Inf.



## Three-node loops are still possible (3)

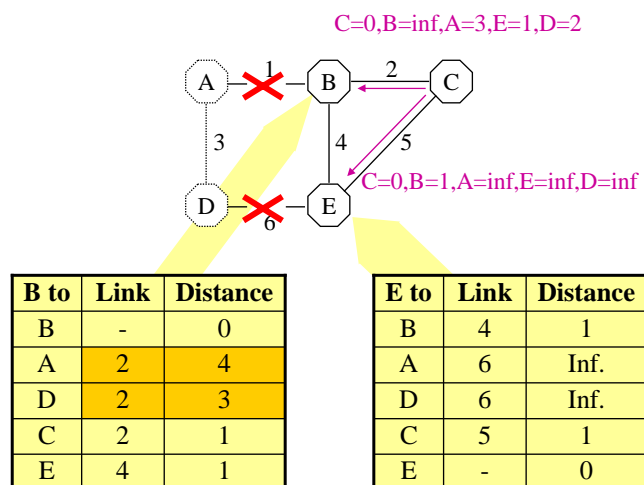
- Also link 6 fails.
- E sends its distance vector to B and C  
 $E=0, B=1, A=\text{inf}, D=\text{inf}, C=1$
- ... But the DV sent to C is lost



x to D	Link from x	Distance
B→D	4	Inf.
C→D	5	2
E→D	6	Inf.

## Three-node loops are still possible (4)

- Now C sends its poisoned DV

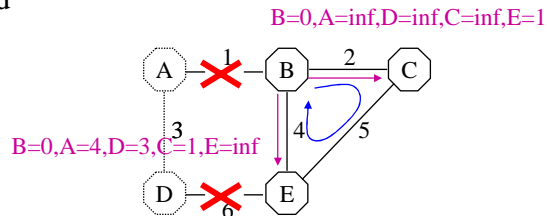


B to	Link	Distance
B	-	0
A	2	4
D	2	3
C	2	1
E	4	1

E to	Link	Distance
B	4	1
A	6	Inf.
D	6	Inf.
C	5	1
E	-	0

## Three-node loops are still possible (5)

- B generates its poisoned distance vectors
- The three node loop is ready
- On link 5 cost=4 is advertised. C's knowledge about the distance to D grows ...
- Routes to D do not change except that the costs keep growing, nodes count to infinity. This finally breaks the loop.



x to D	Link from x	Distance
B→D	2	3
C→D	5	2
E→D	4	4

## How often should a DV-protocol advertise?

Time of advertisement is a compromise:

- + immediate delivery of changed info
- + recovery from packet loss
- + need to monitor the neighbors
- sending all changes at the same time
- traffic load created by the protocol

+ = Faster  
- = Slower

- Entries in the routing tables have **refresh** and **obsolescence** timeouts
- In RIP, the refresh timeout is 30 seconds and the obsolescence timeout is 180 seconds

## The second method to avoid loops is to use triggered updates

- A **triggered update** happens when an entry in the routing table is modified (e.g. when a link breaks)
- Triggered updates reduce the probability of loops
- Triggered updates also speed up counting to infinity
- RIP advertises
  - when the refresh timer expires, and
  - when a change occurs in an entry
- Loops are still possible, e.g. because of packet loss

## The Bellman-Ford algorithm

## Bellman-Ford algorithm (1)

- DV-protocols are based on the Bellman-Ford algorithm
- Centralized version:
  - Let  $N$  be the number of nodes and  $M$  the number of links.
  - $L$  is the link table with  $M$  rows,  $L[l].m$  - link cost  
 $L[l].s$  - link source  
 $L[l].d$  - link destination
  - $D$  is an  $N \times N$  matrix, such that  $D[i,j]$  is the distance from  $i$  to  $j$
  - $H$  is an  $N \times N$  matrix, such that  $H[i,j]$  is the link that  $i$  uses to send to  $j$

$D$	1	..	$i$	..	$N$
1	distance from $i$ to $j$				
:					
$j$					
:					
$N$					

Both directions are presented separately in the link table and distance table!

A column  $\equiv$  DV of the corresponding node

## Bellman-Ford algorithm (2)

- Initialized distance and link matrices

	1	..	$i$	..	$N$
1	$\mathbf{0}$ $\infty$ $\infty$ $\infty$ $\infty$				
:					
$j$					
:					
$N$					

Distance matrix  $D$

	1	..	$i$	..	$N$
1	$-1$ $-1$ $-1$ $-1$ $-1$				
:					
$j$					
:					
$N$					

Link matrix  $H$

*From  $i$  to  $j$*

## Bellman-Ford algorithm (3)

1. Initialization: *(previous slide)*  
If  $i=j$ , then  $D[i,j] = 0$ , else  $D[i,j] = \text{inf}$ .  
Initialize  $\forall H[i,j] = -1$ .
2.  $\forall$  links  $l$  and  $\forall$  destinations  $k$ :
  - i. set  $i = L[l].s$ ,  $j = L[l].d$
  - ii. calculate  $d = L[l].m + D[j,k]$
  - iii. if  $d < D[i,k]$ , set  $D[i,k] = d$ ;  $H[i,k] = l$ .
3. If at least one  $D[i,k]$  changed, go to step 2, else stop.

## Bellman-Ford algorithm (4)

- First in D-matrix appear one hop link distances, then two hop link distances, etc.
- Number of steps  $\leq N$
- Complexity:  $O(M \cdot N^2)$
- Complexity of the distributed version:  $O(M \cdot N)$

## RIP protocol

## RIP-protocol properties (1)

- Simple protocol. Used before standardization.
- RIP version 1
  - RFC 1058 in 1988
- RIP is used inside an autonomous system
  - Interior routing protocol
- RIP works both on shared media (Ethernet) and in point-to-point networks.
- RIP runs on top of UDP (port 520) and IP.

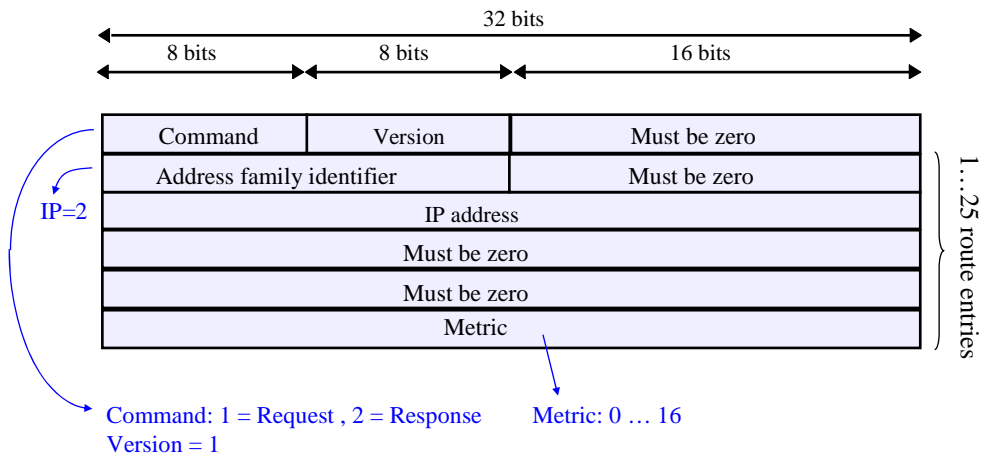
## RIP-protocol properties (2)

- An entry in the routing table represents a network, a sub-network or a host:
  - <netid,0,0> represents a network
  - <netid,subnetid,0> represents a sub-network
  - <netid,subnetid,host> represents a host (used only in exceptional cases)
  - <0.0.0.0> represents a route out from the autonomous system
- The mask must be manually configured.
- Sub-network entries are aggregated to a network entry on interfaces belonging to another network.

## RIP-protocol properties (3)

- Distance = hop count = number of links on a path (route).
  - No other metrics
- Distance 16 = infinite.
- RIP advertises once in 30s.
  - If an entry is 180s old  $\Rightarrow$  distance is set to infinite
  - Advertisements must be randomized to avoid bursts of RIP updates.
- RIP also sends 1-5 s after an update (triggered updates).
- RIP uses poisoned vectors.

## RIP message format



S-38.2121 / Fall-06 / RKa, NB

DV-47

## RIP routing table

A routing table entry contains

- Destination IP address (network)
- Distance to destination
- Next hop IP address
- “Recently updated” flag
- Several timers (refresh, obsolescence...)

S-38.2121 / Fall-06 / RKa, NB

DV-48

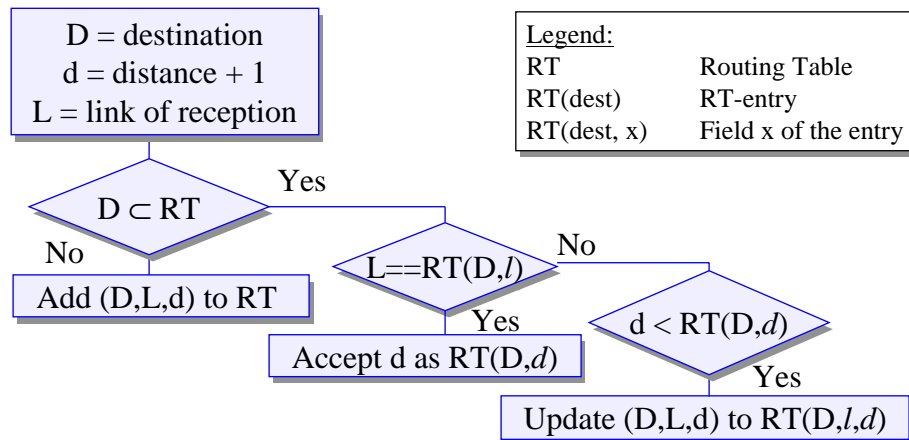


## Routing table example

- Example Kernel routing table

```
# netstat -nr
Kernel routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
127.0.0.1 * 255.255.255.255 UH 1 0 2130 lo0
191.72.1.0 * 255.255.255.0 U 1 0 3070 eth0
191.72.2.0 191.72.1.1 255.255.255.0 UG 2 0 1236 eth0
191.72.3.0 191.72.1.2 255.255.255.0 UG 2 0 3212 eth0
```

## Processing of Received Distance Vectors



*Note: this is simplified, shows only the principle!*

## RIP response messages

- Distance vectors are sent in response messages
- Periodic updates (30 seconds period)
  - All routing table entries
  - Different DV on different links because of poisoned vectors
  - More than 25 entries ⇒ several messages
- Triggered updates after changes
  - Contains changed entries
  - 1-5 seconds delay, so that the message contains all updates that are related to the same change
- Destinations with infinite distance can be omitted if the next hop is same as before.

## RIP request messages

- The router can request routing tables from its neighbors at startup
  - Complete list
  - Response similar to normal update (+ poisoned vectors)
- Partial routing table
  - For debugging
  - No poisoned vectors

## Silent nodes

- When only RIP was used, hosts could listen to routing traffic and maintain their own routing tables
  - Which router is closest to the destination?
  - Which link, if several available?
- These were "silent nodes", that only listened to routing traffic without sending
- Nowadays there are too many routing protocols
  - RIP-2, OSPF, IGRP, ...

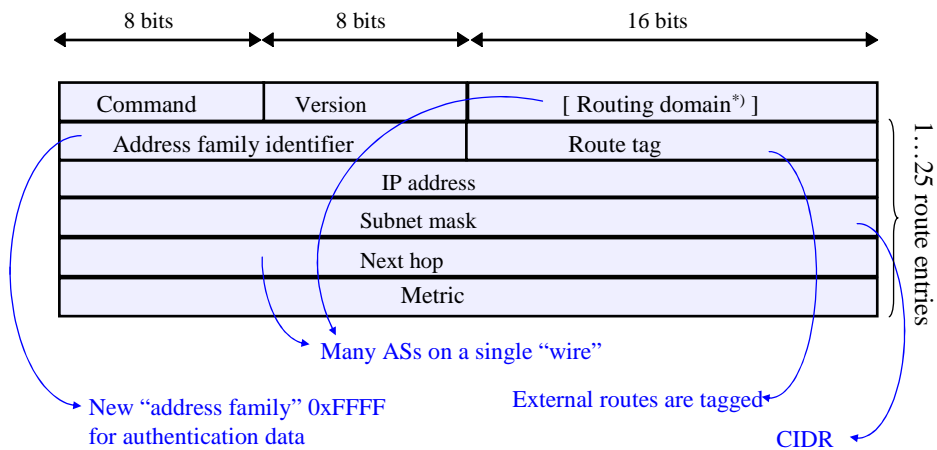
*Historical*

## RIP version 2

## RIP version 2

- RFC-1388 (1387,1389)
- Why?
  - Simple and lightweight alternative to OSPF and IS-IS
- RIP-2 is an update that is partially interoperable with RIP-1
  - A RIP-1 router understands some of what a RIP-2 router is saying.
- Improvements
  - Authentication
  - Support for CIDR
  - Next hop –field
  - Subnet mask
  - Support for external routes
  - Updates with multicast

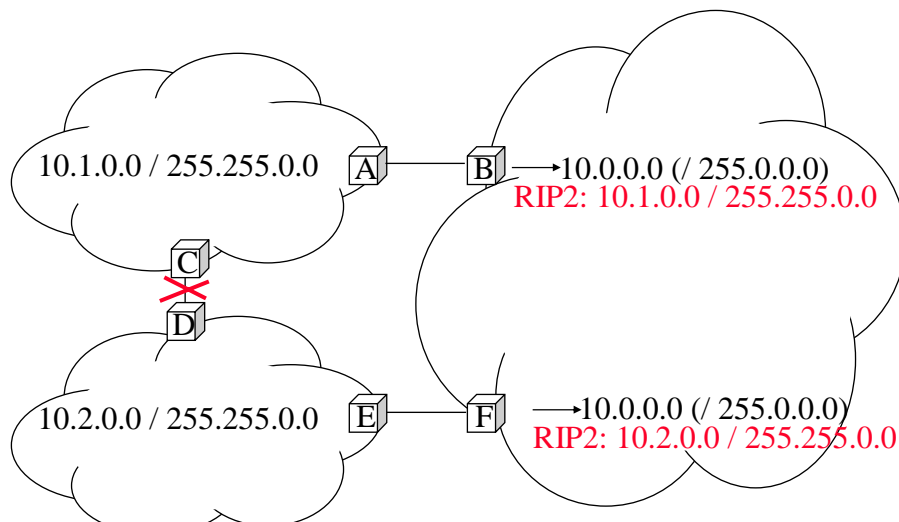
## RIP version 2 – messages



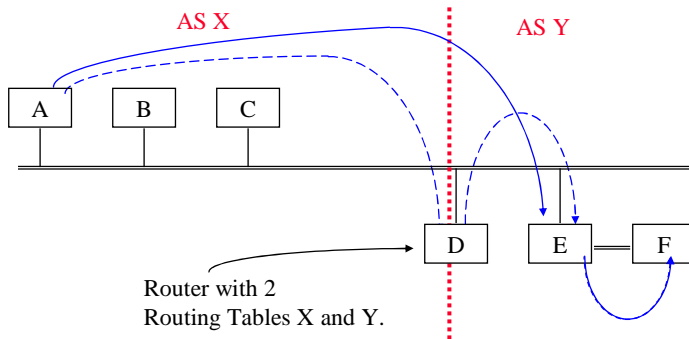
## Routing from one sub-net to another (1)

- In RIP-1 the subnet mask is not known outside the subnet, only network-id is sent in an advertisement out from a subnet
  - ⇒ A host and a subnet can not be distinguished
  - ⇒ All subnets must be interconnected with all other subnets and exterior traffic is received in the nearest router independent of the final destination inside our network
- RIP-2 corrects the situation by advertising both the subnet and the subnet mask
  - Masks of different length within a network
  - CIDR
  - RIP-1 does not understand

## Routing from one sub-net to another (2)



## Routing domain and next hop



Next hop  $\Rightarrow$  D advertises in X: the distance to F is f and the next hop is E!

## Support for local multicast

- RIP-1 broadcasts advertisements to all addresses on the wire
  - Hosts must examine all broadcast packets
- RIP-2 uses a multicast address for advertisements
  - 224.0.0.9 (all RIP-2 routers)
  - No real multicast support needed, since packets are only sent on the local network
- Compatibility problems between RIP-1 and RIP-2

## Acknowledged updates (extension)

- When RIP is used on ISDN links a new call is established per 30s
  - ⇒ Expensive.
- Slow network ⇒ queue length are restricted.
  - RIP sends its DVs 25 entries/message in a row ⇒ RIP messages may be lost.
- A correction proposal: ack all DVs: no periodic updates
  - ⇒ If there are no RIP message: assume that neighbor is alive and reachable
  - ⇒ Info on all alternative routes is stored.

*Not part of RIP-2*

## The synchronization problem

- Routers have a spontaneous tendency to synchronize their send times. This increases the probability of losses in the net.
- Reason: send interval = constant + time of message packing + processing time of messages that are in the queue.
- Therefore, send instants are randomized between 15s ... 45s.