

13. Разпределена  
маршрутизация с дистантен  
вектор. Маршрутен протокол  
RIP в IPv4 и IPv6.

# Малко история

Маршрутизиращ протокол с дистантен вектор (**distance vector protocol**) е използван отначало в **ARPANET**.

По-късно в Интернет намира широко приложение като **RIP** (Routing Information Protocol).

Основите на тези алгоритми са поставени от Белман (1957 г.), Форд и Фолкерсън (1962 г.).

Затова са известни като алгоритми **Белман-Форд** или **Форд-Фолкерсън**.

Само на **Cisco Systems** – **IGRP** и **EIGRP**.

# Основни принципи

**Distance Vector** – рутерите се анонсират (рекламират) като **вектори**:

**Посока** - адреса на следващия възел (**next hop**) и изходящия интерфейс и

Разстояние (**метрика**), напр. брой възли до дестинацията (**hop count**).

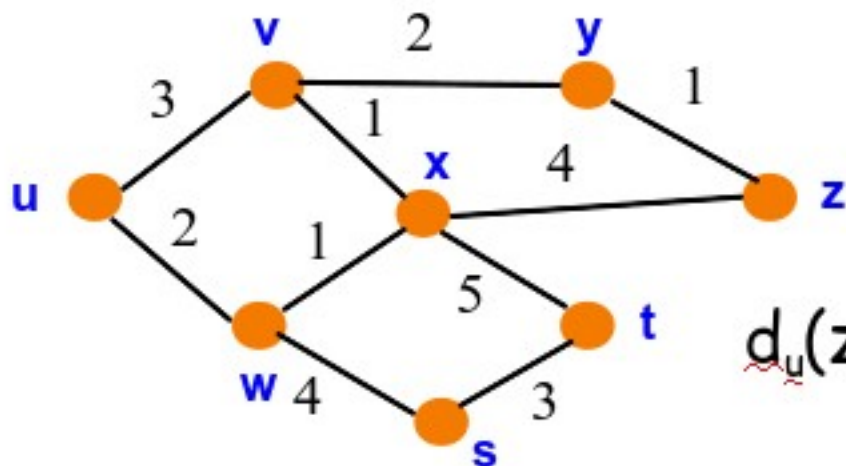
# Основни принципи

Маршрутизаторите (рутери) в тези случаи **не знаят целия път** до крайната точка (дестинация).

DV използва само:

1. Посока или интерфейс, по който да се отправи пакета.
2. Разстоянието до дестинацията.

# Алгоритъм Белман-Форд



$$d_u(z) = \min\{c(u,v) + d_v(z), c(u,w) + d_w(z)\}$$

$d_v(z)$  – дистантния вектор от  $v$  до  $z$

(Всеки възел **периодически изпраща** до съседите си своя дистантен вектор)

# Общи положения

При маршрутизацията с дистантен вектор (**distance vector routing**) всеки маршрутизатор изгражда и поддържа маршрутна таблица, в която всеки ред съдържа **адрес на местоназначение**, адрес на следващата стъпка към това местоназначение по най-добрия известен до момента път и дължината на този път (**метрика**).

**Периодически** маршрутизаторите изпращат на съседите си цялата или част от маршрутната таблица.

# Предимства и недостатъци

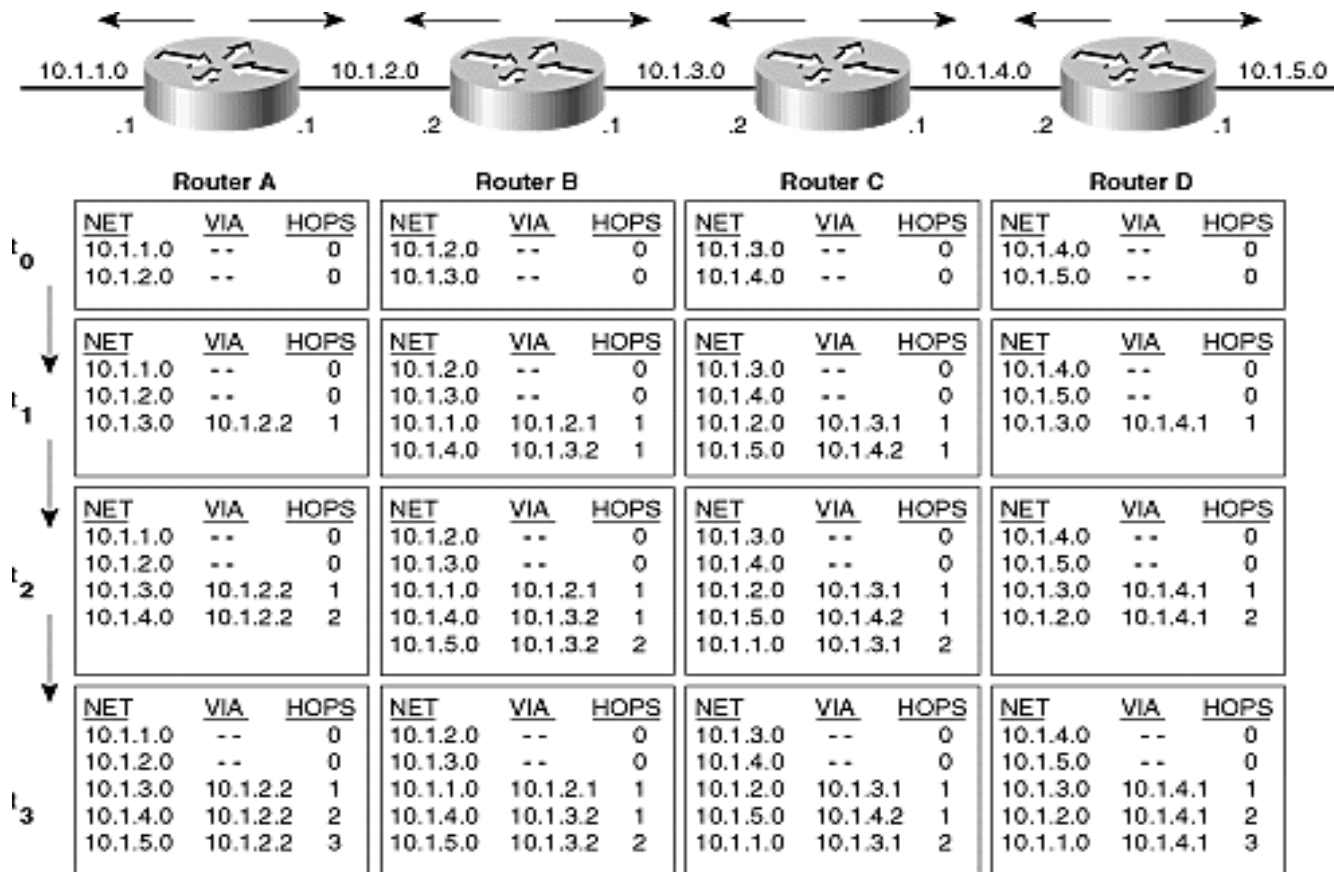
DV алгоритмите:

- не товарят процесора и паметта;
- лесни са за реализация и поддръжка;

Но

- Периодическите **update-и** отнемат пропускателна способност от потребителите.

# Distance Vector в действии





# Метрика

Предполага се, че всеки маршрутизатор знае **метриката** на връзките до своите съсед.

Ако метриката е брой стъпки или маршрутизатори до местоназначението (**хопове**), разстоянието до всеки съсед е 1.

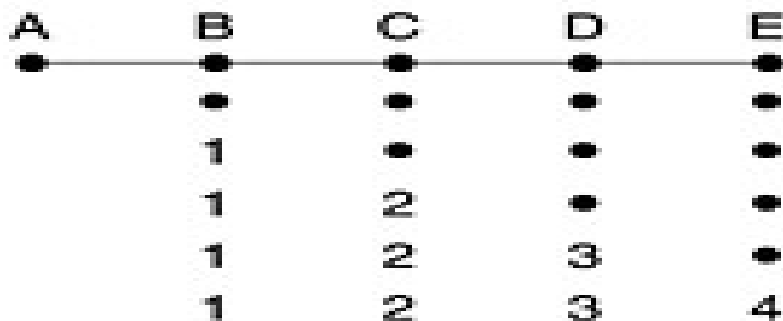
Ако метриката е **натоварване на възела**, разстоянието до всеки съсед е броя на пакетите в изходящата опашка към този пакет.

Ако метриката е **времезакъснение**, маршрутизаторът периодично изпраща “ехо” пакети до съседните му маршрутизатори и измерва закъснението на техния отговор.

# Недостатък

Сериозен недостатък на маршрутизиращите алгоритми с дистантен вектор е **ниската им скорост на сходимост**.

Добрите новини се разпространяват бързо в мрежата, но лошите новини обикновено изискват **твърде голям брой** периодични съобщения за да достигнат до всички маршрутизатори.



# Добавяне на обект

Нека маршрутизаторът **A** в началото не е включен в мрежата.

Всички останали маршрутизатори знаят това - в маршрутната им таблица към направлението **A** е записано  $\infty$  (достатъчно голямо число, трябва да е поне с единица повече от диаметъра на мрежата). Това е отразено на първия ред по-горе.

След **включването на A** останалите маршрутизатори научават за това събитие чрез няколко обмена на своите вектори на разстоянията, всеки от които се извършва едновременно между всички съседни маршрутизатори.

# Добавяне на обект

При първата обмяна  $B$  научава от  $A$  за път с дължина 0 до  $A$  и записва в своята таблица, че  $A$  е на разстояние 1.

В този момент останалите маршрутизатори все още не са научили за включването на  $A$ . Това е отразено на втория ред по-горе.

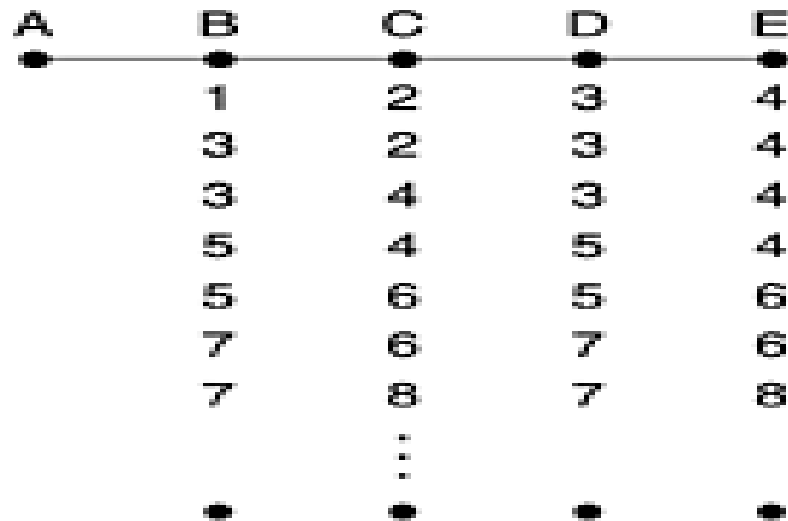
При следващия обмен  $C$  научава, че от  $B$  съществува път до  $A$  с дължина 1 и записва в своя вектор път до  $A$  през  $B$  на разстояние 2 и т.н.

# Добавяне на обект

По-общо в мрежа с диаметър  $k$  хопа са необходими най-много  $k$  размени на съобщения за разпространяване на новината за появил се по-добър път.

# Исключване на обект

Да разгледаме друг пример.



# Иключване на обект

Нека всички маршрутизатори в началото са включени в мрежата.

Да предположим, че  $A$  спира да работи или се прекъсва връзката от  $A$  до  $B$ , което от гледна точка на  $B$  е същото.

При първия обмен  $B$  не получава информация от  $A$ , но получава информация от  $C$ , че има път до  $A$  с дължина 2.

$B$  не знае, че пътя от  $C$  до  $A$  минава през него - от негова гледна точка би могъл да съществува друг независим път от  $C$  до  $A$ , затова  $B$  записва в таблицата си в реда за  $A$  път с дължина 3 и следваща стъпка  $C$ .

# Исключване на обект

$D$  и  $E$  не променят маршрутните си таблици при първия обмен на векторите на разстоянията.

На следващия обмен  $C$  научава за два възможни пътя до  $A$ , и двата с дължина 4, единият през  $B$ , другият през  $D$ .

$C$  избира и записва в маршрутната си таблица единия от тях в зависимост от реда на обработването на съобщенията от  $B$  и  $D$ .



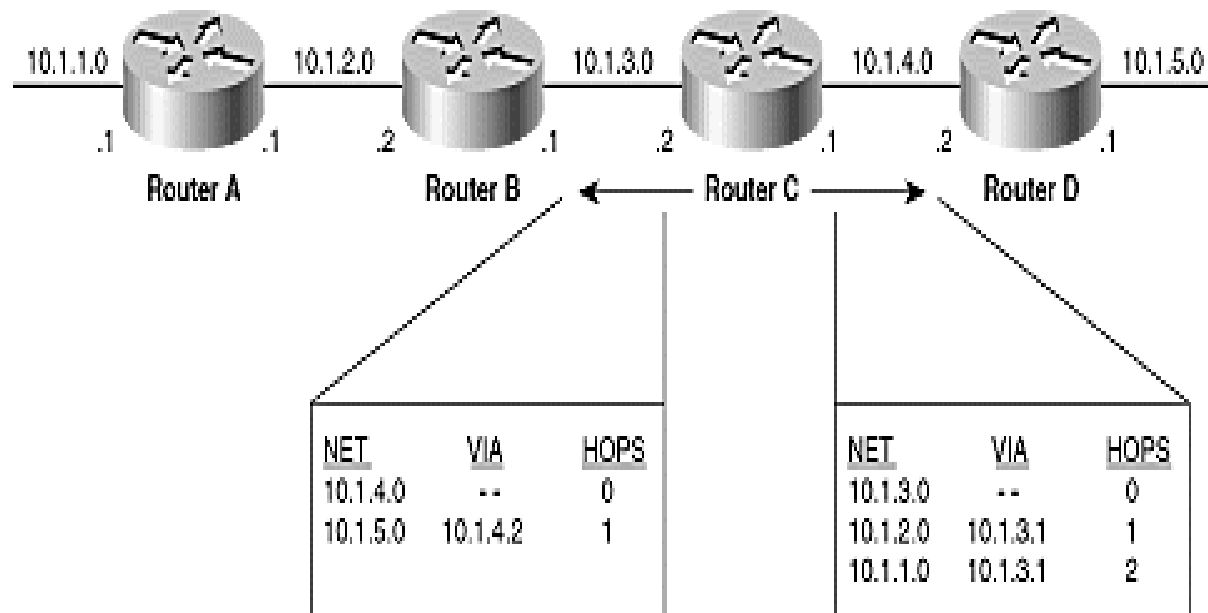
# Count to Infinity

Резултатът от продължаващия обмен е отразен в следващите редове по-горе. Той ще продължи, докато стойностите по направленията към  $A$  и в **четирите маршрутизатора не достигнат  $\infty$** .  
Този проблем се нарича **броене до безкрайност (count to infinity)**.

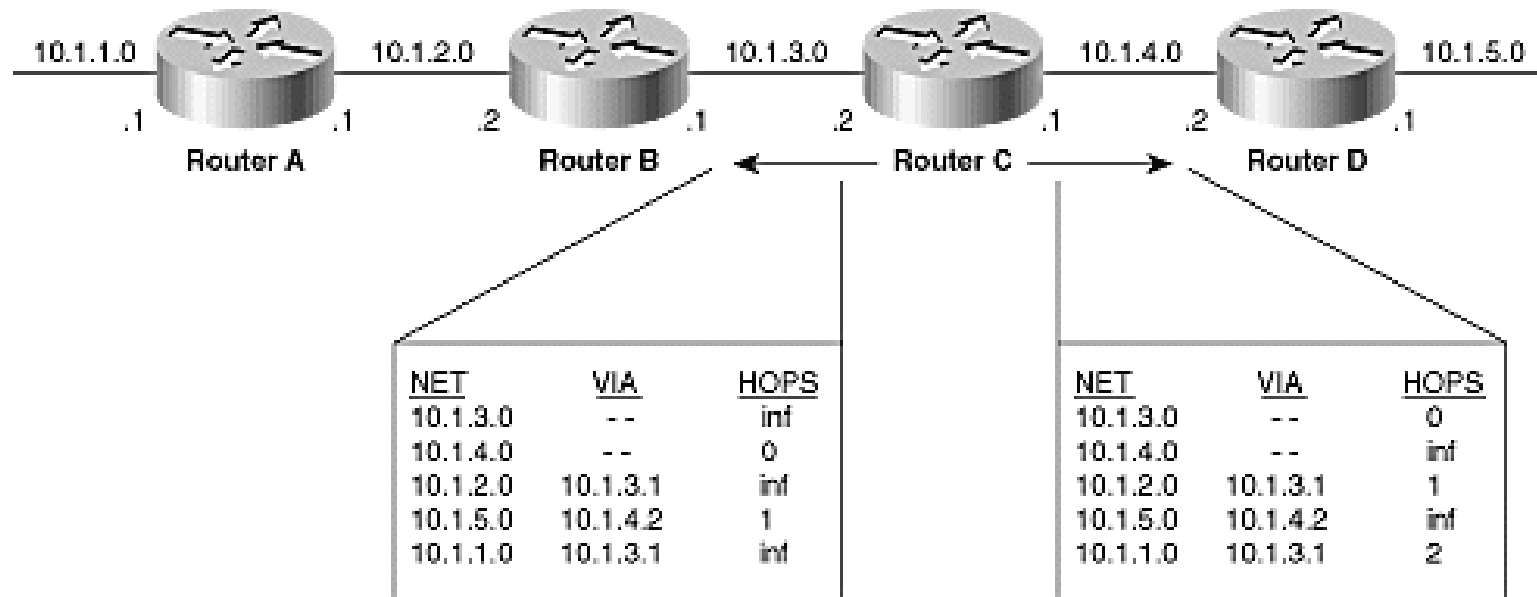
# Split horizon

Едно частично негово решение е т.н. **разделяне на хоризонта (split horizon)**.  
При него се въвежда ново правило - ако в маршрутната таблица на  $X$  в реда за  $Y$  е записана следваща стъпка  $Z$ , то  $X$  не изпраща към  $Z$  информация за маршрута към  $Y$ .

# Split horizon



# Split horizon wi poisoned reverse



# Частично решение - **split horizon**

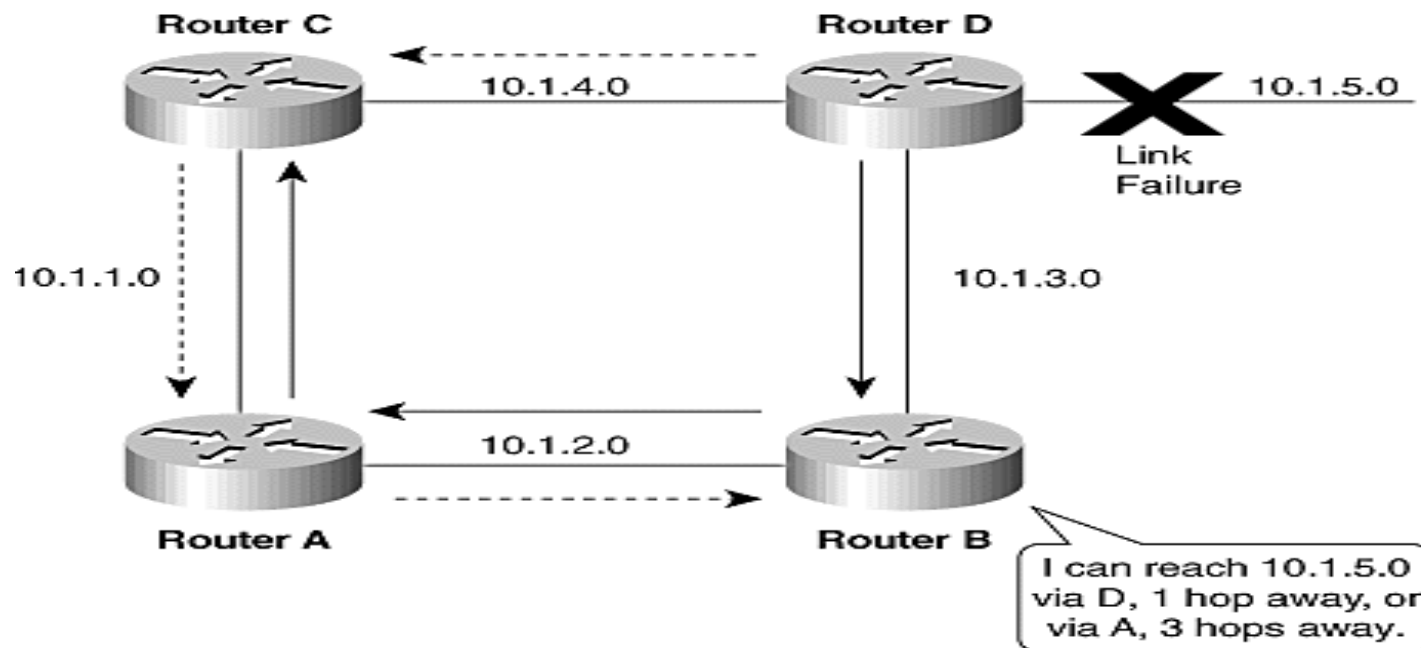
Решението е “**Split horizon wi poisoned reverse**“, което се прилага при съвременните **DV** протоколи

Но...

Въвеждането на разделяне на хоризонта не винаги решава напълно проблема **броене до безкрайност**.

Да разгледаме следната **топология**:

# Броене до безкрайност



# Въвеждане на $(\text{hop count})_{\max}$

10.1.5.0 пада. Рутер **D** изпраща “updates” до съседите си:

- **C** (прекъсната линия)
- **B** (непрекъсната).

**B** маркира маршрута през **D** “недостижим”, но **A** анонсира резервен най-добър път до 10.1.5.0 на 3 хопа: влиза в таблицата на **B**.

**B** “подава” 10.1.5.0 на 3 хопа на **D**.

**D** “обновява” **C** с 4-hop маршрут до 10.1.5.0.

# Въвеждане на $(\text{hop count})_{\max}$

**С** “казва” на **А**: 5-hop маршрут до 10.1.5.0.

**А** “казва” на **В**: 6-hop маршрут до 10.1.5.0.

**В** “мисли”: “пътят на **А** до 10.1.5.0 се е удължил, но е единствен, ще го **ползвам!**”

**В** променя:  $\text{hop count} = 7$ , обновява **D** и “**въртележката**” се завърта наново.

Това е “броене до безкарайност” -

*counting-to-infinity*: **hop count до 10.1.5.0** ще продължи да нараства до  $\infty$ .

А всички **прилагат split horizon?!**



# Въвеждане на $(\text{hop count})_{\text{max}}$

Начинът да се избегне този проблем е дефиниране на  $(\text{hop count})_{\text{max}}$

Ограничаваме  $\text{hop count} (=15)$

16-hop маршрут е недостижим ( $\text{poisoned reverse}$ ),

но конвергенция:  $\approx 7$  минути при  $\text{update} = 30 \text{ s}$ .

Недостатъчно, затова:

$\text{triggered updates}$  и  $\text{holddown timers}$

# Triggered Updates

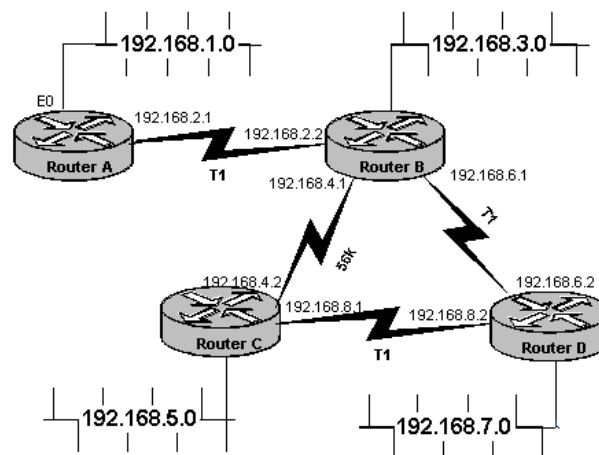
Ако (за добро или лошо) метриката се промени, рутерът веднага изпраща “update”, без да чака нулиране на таймера.

Реконвергирането ще стане много по-бързо, “counting to infinity” проблема се редуцира.

Допълнително прецизиране на метода:

обновяване само на мрежите, които са предизвикали (triggered) проблема, а не цялата таблица с маршрути. Така се спестява процесорно време и пропускателна способност на мрежата.

# Holddown (timeout)



Рутер В рекламира **192.168.1.0** като **Network Unreachable** към D. Рутер D поставя 192.168.1.0 в **holddown**. Но **рутер C** рекламира маршрутната си таблица към D, в която 192.168.1.0 е “**жива**”. Но в рутер D 192.168.1.0 е в holddown, т.е D няма да приеме информацията за 192.168.1.0, идваща от C.

# Holddown

Triggered Updates не достига едновременно до всички маршрутизатори.

Маршрутизатор, още **неполучил съобщението**, би изпратил **редовно периодично** съобщение за обновяване с **неточна** информация до друг маршрутизатор, който вече има актуалната информация за променен маршрут.

# Holddown

Защита срещу това: **holddown** таймер или **timeout** според RFCs 1058 и 2453.

След изпращане или получаване на **triggered update** (например “**unreachable**” маршрут или **hop count** - от **2** на **4**) маршрутизаторът стартира **holddown** таймер.

До неговото нулиране **не приемат** съобщения за **обновяване** на променения маршрут.

# Дефиниране на Holddown

**Компромис:** редуцира се вероятността за вкарване на “лоша” информация в таблицата за сметка на времето за реконвергенция.

Да се внимава с определянето на стойността **holddown**:

**По-късият** период е по-неефективен;

**По-дългият** – по-зле за нормалното рутиране.

# Flush timer (garbage-collection time)

След **изтичане на timeout**, маршрутът вече не е валиден.

Но се държи в таблицата за кратък период от време, за да научат съседите, че ще бъде изхвърлен.

След **изтичане на garbage-collection timer**, маршрутът окончателно се премахва от таблиците.

# RIP. Общи положения.

**RIP** (routing information protocol) е широко използван маршрутизиращ протокол с вектор на разстоянието (distance vector).

Той е подходящ предимно за малки мрежи, в които относително рядко настъпват промени в топологията.

Всеки ред в маршрутната таблица на RIP маршрутизаторите съдържа информация за **направлението**, **следваща стъпка към това направление** и **метрика**.



# RIP. Общи положения.

**Метриката** обозначава разстоянието в стъпки до местоназначението, т.е. метриката използвана от RIP протокола е **брой хопове**.

**Максималният** брой хопове в една RIP мрежа е **15**.

За обмен на маршрутна информация: порт **520/UDP**

MAC header	IP header	UDP header	RIP header	Data :::
------------	-----------	------------	------------	----------

# RIP таймери

RIP на всеки 30 секунди изпраща копие на маршрутната таблица към съседните маршрутизатори.

Таймерът за невалиден маршрут (**hold down time**) е 180 s.

Определя интервала от време, след който даден маршрут се счита за невалиден, ако маршрутизаторът не е получил съобщения за него.

# RIP таймери

Когато даден път бъде отбелязан като невалиден, се изпращат съобщения с тази информация към съседните маршрутизатори и се преустановява използването му.

Тези съобщения се изпращат до изтичането на таймера за изтриване на маршрут (**flush timer**).

След което пътят се изтрива окончателно от маршрутната таблица.

# Формат на RIP пакетите

Първата версия на **RIPv1** не поддържа **subnet маски**, т.е. VLSM, респ. CIDR.

Втора версия на протокола - **RIPv2**,  
поддържа **VLSM**, респ. **CIDR**. Форматът  
на пакетите на версия RIPv2 е следния:

# Формат на RIPv2 пакетите

Command	Version	Routing domain
Address family		Route tag
IP address		
Netmask		
Next hop IP address		
Metric		

# Формат на RIPv2 пакетите

Първите три полета **Command**, **Version** и **Routing domain** представляват заглавната част на пакета, а останалите шест полета съдържат **данни за маршрути** и комбинация от тях може да се повтаря до 25 пъти в един RIPv2 пакет.

За пренасяне на информацията от по-големи маршрутни таблици се използват няколко RIPv2 пакета.

Полето **Command** указва дали пакетът съдържа **заявка или отговор**.

# Формат на RIPv2 пакетите

Полето **Version** указва версията на протокола, за RIPv2 тази стойност е 2.

Полетата **Routing domain** и **Route Tag** не се използват и се запълват с нули.

Полето **Address family** е равно на 2, ако следва IP адрес. Ако имаме заявка за цялата маршрутна таблица, е 0.

# Сходимость на RIP

При промяна в топологията на мрежата се налага всички маршрутизатори да преизчислят своите вектори на разстоянията и да достигнат до непротиворечиво описание на новата топология.

За увеличаване на скоростта на сходимость на RIP се използват различни методи, например разделяне на хоризонта (*split horizon*).

Тези методи намаляват вероятността за поява на цикли в маршрутите, но не могат да гарантират отсъствието им.



# Count to infinity

Максималният брой **хопове** в RIP е **15**.

Всяко **местоназначение**, което е на разстояние **над 15 хопа** се приема за **недостижимо**.

Това прави **невъзможно** прилагането на **RIP** в мрежи с **повече от 15 рутера**.

Но **ограничава** ситуацията “броене до безкрайност” (**Count to infinity**), при която могат да се получат **цикли** в маршрутите.

# Версии на RIP

RIPv1 (RFC 1058) прилага само **classful** маршрутизация.

Т.е периодичните **updates** не носят **subnet** информация.

Не е възможно да имаме подмрежи от един и същи клас с различни маски. С други думи, всички подмрежи от даден клас трябва да бъдат с еднакви маски.

# Версии на RIP

RIPv2 е разработен през 1994 г. и има възможност да носи subnet информация, да поддържа CIDR.

За поддържане на обратна съвместимост с версия 1 запазено е ограничението от 15 хопа.

За сигурност е въведена аутентикация с явен текст, подобрена с MD5 (RFC 2082).

# Версии на RIP

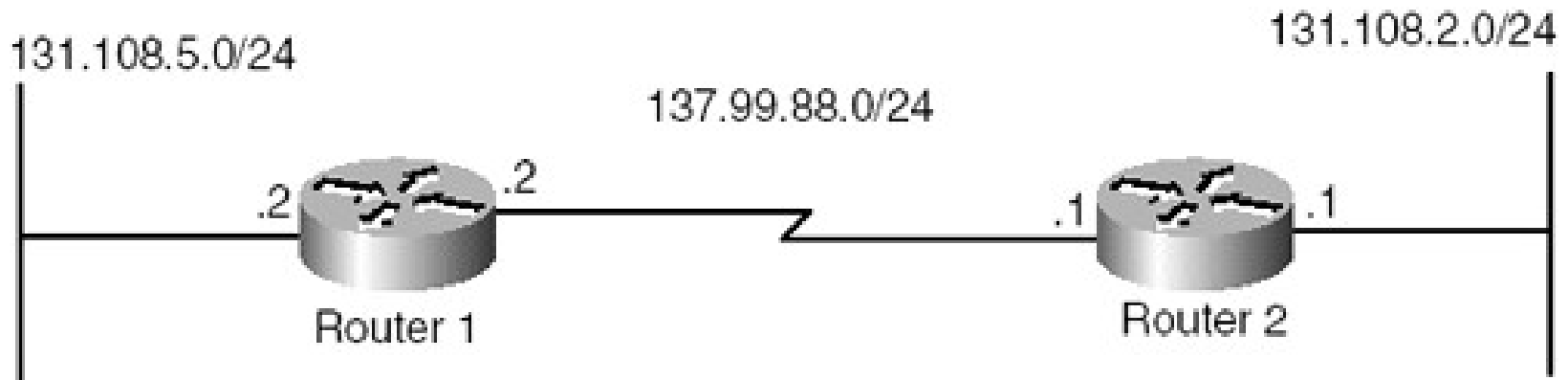
За да не се товарят хостове, които не са участници в RIP, RIPv2 “мултикаства” обновленията на адрес 224.0.0.9, за разлика от RIPv1, който е broadcast.

# RIPv2. Discontiguous Networks.

Discontiguous network е мрежов префикс, разделен от друг (различен) префикс.

На следващия слайд 131.108.0.0 е разделена от подмрежа на 137.99.0.0; т.е 131.108.0.0 е discontiguous network.

# RIPv2. Discontiguous Networks.

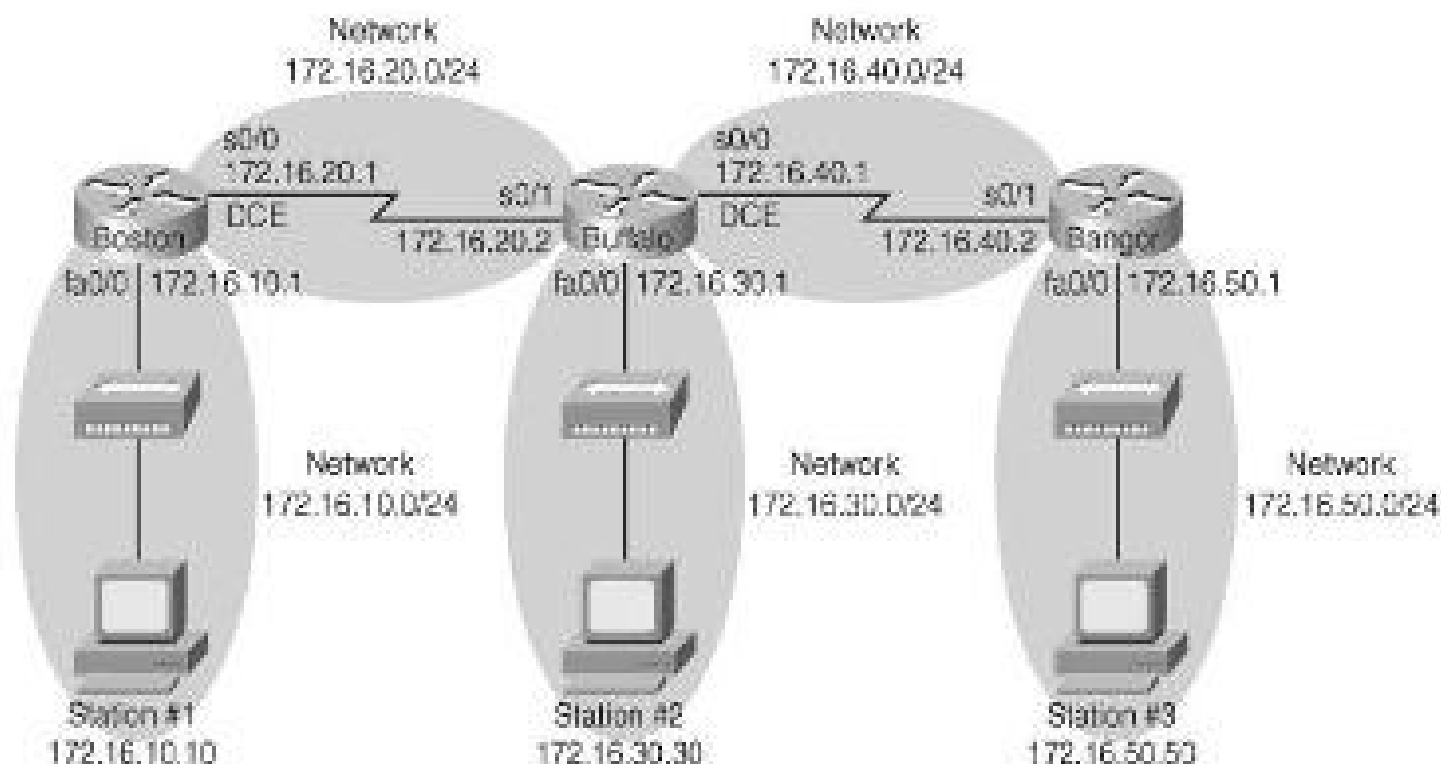


RIPv1 прави **summary** към **classful** границата.

Router 1 (2) изпраща **update** към Router 2 (1) през 137.99.88.0:

Router 1 (2) конвертира **131.108.5.0/24** (**131.108.2.0/24**) в **131.108.0.0/16**.

# Конфигуриране на RIP



# Boston Router

```
Boston>en
```

```
Boston#config t
```

```
Boston(config)#router rip
```

```
Boston(config-router)#version 2
```

```
Boston(config-router)#network 172.16.0.0
```

!Advertises directly connected networks  
(classful address only)

```
Boston(config-router)#no auto-summary
```

!Turns off autosummarization



# Buffalo Router

```
Buffalo>en
```

```
Buffalo#config t
```

```
Buffalo(config)#router rip
```

```
Buffalo(config-router)#version 2
```

```
Buffalo(config-router)#network 172.16.0.0
```

```
Buffalo(config-router)#no auto-summary
```

# Bangor Router

```
Bangor>en
```

```
Bangor#config t
```

```
Bangor(config)#router rip
```

```
Bangor(config-router)#version 2
```

```
Bangor(config-router)#network 172.16.0.0
```

```
Bangor(config-router)#no auto-summary
```

# Конфигуриране на RIP

```
Router(config)# router rip
```

```
Router(config-router)# network 10.0.0.0
```

```
Router(config-router)# exit
```

```
Router(config)# interface ethernet1
```

```
Router(config-if)# ip address 10.1.1.1  
255.255.255.0
```

```
Router(config-if)# no ip split-horizon
```

```
Router(config-if)# exit
```

# Конфигуриране на RIP. Други команди.

```
router rip
```

```
Passive-interface eth0
```

! не изпраща update-и по interface eth0

# RIPng

RIPng (RFC 2080) е разширение на RIPv2 за поддържане на IPv6:

- в маршрутната таблица IPv6 префикс, next-hop IPv6 адрес
- използва порт 521/UDP и мултикаст (FF02::9) за updates
- сигурност: IP AH (Authentication Header) и IP ESP (Encapsulating Security Payload)

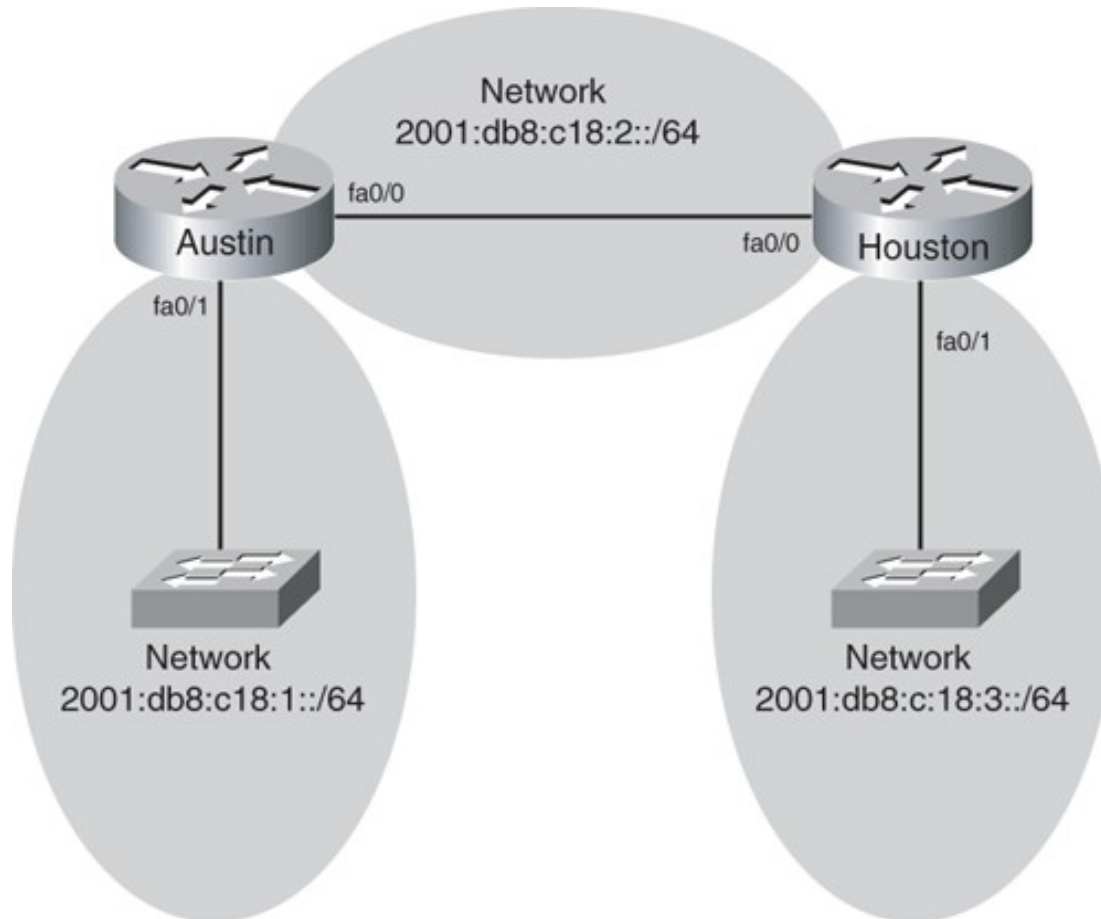
# Конфигуриране на RIPv2 (Cisco)

Не изисква глобална конфигурация.

Освен за целия RIPv2 - таймери, default-route origination, maximum-paths и др.

На един рутер, множество RIPv2 процеси.

# Конфигуриране на RIPvng (Cisco)



# Конфигуриране на RIPvng (Cisco)

```
Austin(config)#ipv6 unicast-routing
```

```
Austin(config)#interface fastethernet 0/0
```

```
Austin(config-if)#ipv6 enable
```

```
Austin(config-if)#ipv6 address
```

```
2001:db8:c18:2::/64 eui-64
```

```
Austin(config-if)#ipv6 rip tower enable
```

```
Austin(config-if)#interface fastethernet 0/1
```

```
Austin(config-if)#ipv6 enable
```

```
Austin(config-if)#ipv6 address
```

```
2001:db8:c18:1::/64 eui-64
```

```
Austin(config-if)#ipv6 rip tower enable
```



# Конфигуриране на RIPvng (Cisco)

```
Houston(config)#ipv6 unicast-routing
```

```
Houston(config)#interface fastethernet 0/0
```

```
Houston(config-if)#ipv6 enable
```

```
Houston(config-if)#ipv6 address  
2001:db8:c18:2::/64 eui-64
```

```
Houston(config-if)#ipv6 rip tower enable
```

```
Houston(config-if)#interface fastethernet  
0/1
```

```
Houston(config-if)#ipv6 enable
```

```
Houston(config-if)#ipv6 address  
2001:db8:c18:3::/64 eui-64
```

```
Houston(config-if)#ipv6 rip tower enable
```

# ripngd Configuration (quagga)

```
[root@rec-gw quagga]# less ripngd.conf.sample
```

```
...
```

```
! debug ripng events
```

```
! debug ripng packet
```

```
!
```

```
router ripng
```

```
network sit1 !sit tunnel-interface
```

```
route 3ffe:506::0/32
```

```
distribute-list local-only out sit1
```

```
!
```

```
!ipv6 access-list local-only permit 3ffe:506::0/32
```

```
!ipv6 access-list local-only deny any
```

# ripngd команды в Terminal Mode

```
#show ip ripng
```

```
#show debugging ripng
```

```
#debug ripng events
```

```
#debug ripng packet
```

```
#debug ripng zebra
```