

# 15. Маршрутизация със следене състоянието на връзката.

Йерархична маршрутизация

# Link State vs. distance vector

Информацията в един distance vector (DV) е сравнима с пътният знак.

Докато link state (LS) протоколите са пътна карта.

LS рутерът има пълна картина на мрежата и и по-трудно ще вземе неправилно решение.

DV маршрутизират по слухове.

# Link State. Особенности.

LS – подава на съседите си информация за директно свързаните връзки и състоянието им (затова е link state).

Тази информация се подава от рутер на рутер, всеки я копира, без да я променя. Всеки рутер има идентична информация за мрежата.

Всеки рутер сам за себе си изчислява най-добрите пътища до съответните префикси.

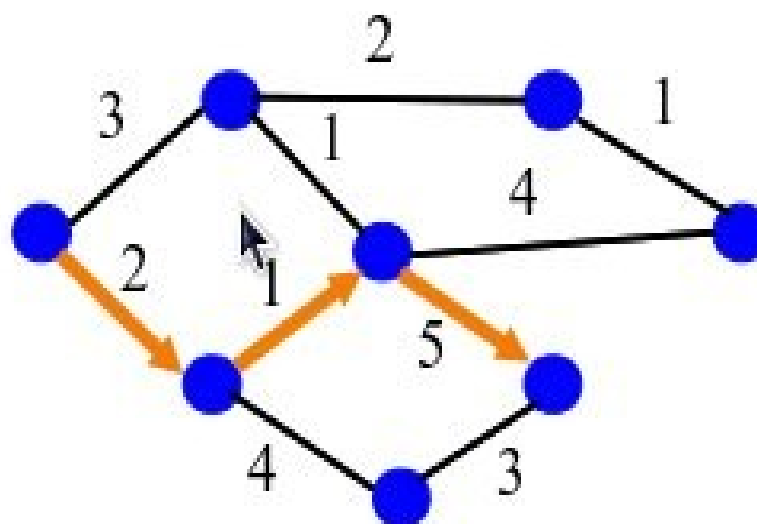
# Алгоритъм на Dijkstra

LS протоколите - *shortest path first* или *distributed database*, се базират на алгоритъма на E. W. Dijkstra - *shortest path*.

Такива протоколи са:

- Open Shortest Path First (**OSPF**) за IP;
- Intermediate System to Intermediate System (**IS-IS**) на **ISO** за CLNS и IP;
- Optimized Link State Routing Protocol (**OLSR - rfc3626**).

# Алгоритъм на Dijkstra



# Пет основни действия

При маршрутизацията със следене състоянието на връзката (**link state routing**) всеки маршрутизатор трябва да извършва следните пет основни действия:

1. Откриване на **съседните маршрутизатори** и техните мрежови адреси.
2. Измерване на **стойностите на връзките** до съседните маршрутизатори.
3. Конструирание на пакети с **информация за състоянието** на връзките.

# Пет основни действия

4. Изпращане на тези пакети до всички останали маршрутизатори.
5. Изчисляване на най-късия път до всеки маршрутизатор в мрежата.

# Резултат от тези действия

В резултат на тези пет действия се събира и разпространява до всички маршрутизатори **информация за цялата топология** на мрежата.



# Откриване на съседните маршрутизатори

След включването на един маршрутизатор неговата първа задача е да научи **кои са съседите** му.

Това се постига чрез **изпращане на “ехо” пакет** по всяка от изходящите линии на маршрутизатора.

От своя страна, **всеки от съседите отговаря** като съобщава името си. Това име трябва да бъде уникално в мрежата. (Например, **IP адрес** на някой от интерфейсите.)

# Откриване на съседните маршрутизатори

Ако два или повече маршрутизатора са свързани в мрежа с общодостъпно предаване (например Ethernet), откриването на съседите е малко по-сложно.

Един възможен начин за представяне на връзките между тях е да се въведе допълнителен възел, който да отговаря на общата среда за предаване.

# link state packets

След като събере необходимата информация за състоянието на връзките си, **следващата задача** на маршрутизатора е да конструира пакет, който **съдържа тази информация**.

**Пакетът трябва да съдържа** уникалното име на подателя, пореден номер, срок на годност и списък със съседите на подателя, като за всеки съсед е указана цената на връзката до него.

Определянето на момента, в който трябва да бъдат подготвени и изпратени пакетите, е важна задача.

# link state packets

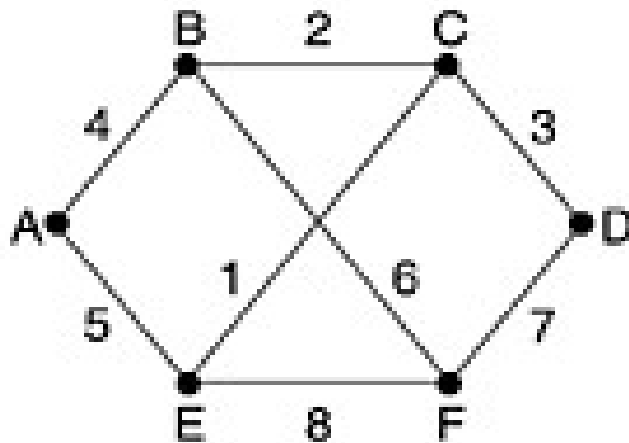
Един възможен начин е това да става през определени равни интервали от време.

Друга по-добра възможност е пакетите да се подготвят и изпращат само при промяна в топологията на мрежата - след отпадане или поява на нов съсед или промяна в цената на някоя връзка.

Нека да разгледаме следната примерна мрежа. Ребрата имат етикети със съответното време-закъснение.

# link state packets

Пакетите със състоянието за връзките за шестте маршрутизатора изглеждат по следния начин:



Link		State		Packets	
A	B	C	D	E	F
Seq.	Seq.	Seq.	Seq.	Seq.	Seq.
Age	Age	Age	Age	Age	Age
B 4	A 4	B 2	C 3	A 5	B 6
E 5	C 2	D 3	F 7	C 1	D 7
	F 6	E 1		F 8	E 8

# Разпространяване на **link state packets**

Най-съществената част на алгоритъма е надеждното доставяне на пакетите с информацията за състоянието на връзката до всички маршрутизатори.

За **разпространението** на пакетите се използва методът на наводняването (**flooding**). При него всеки пакет се изпраща по всички линии, освен линията по която е пристигнал.

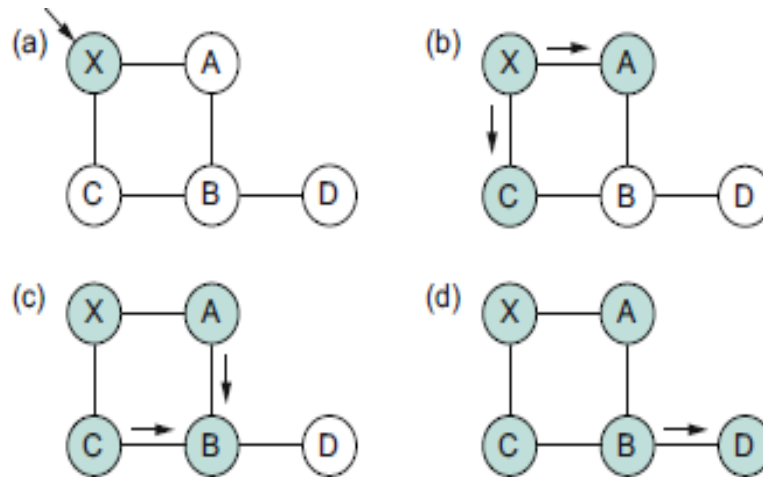
# Разпространяване на link state packets

Обработката на всеки пристигнал пакет започва с **проверка** дали пакетът има **по-голям пореден номер** в сравнение с най-големия пореден номер, който е пристигнал до този момент от този източник.

**Ако номерът е по-голям**, информацията от пакета се **записва** в таблицата с информация за състояние на връзките и пакетът се предава по останалите линии.

Ако номерът е **по-малък** или равен, пакетът се **отхвърля**.

# Разпространяване (Flooding) на link state packets





# Пореден номер

Този алгоритъм има някои проблеми.

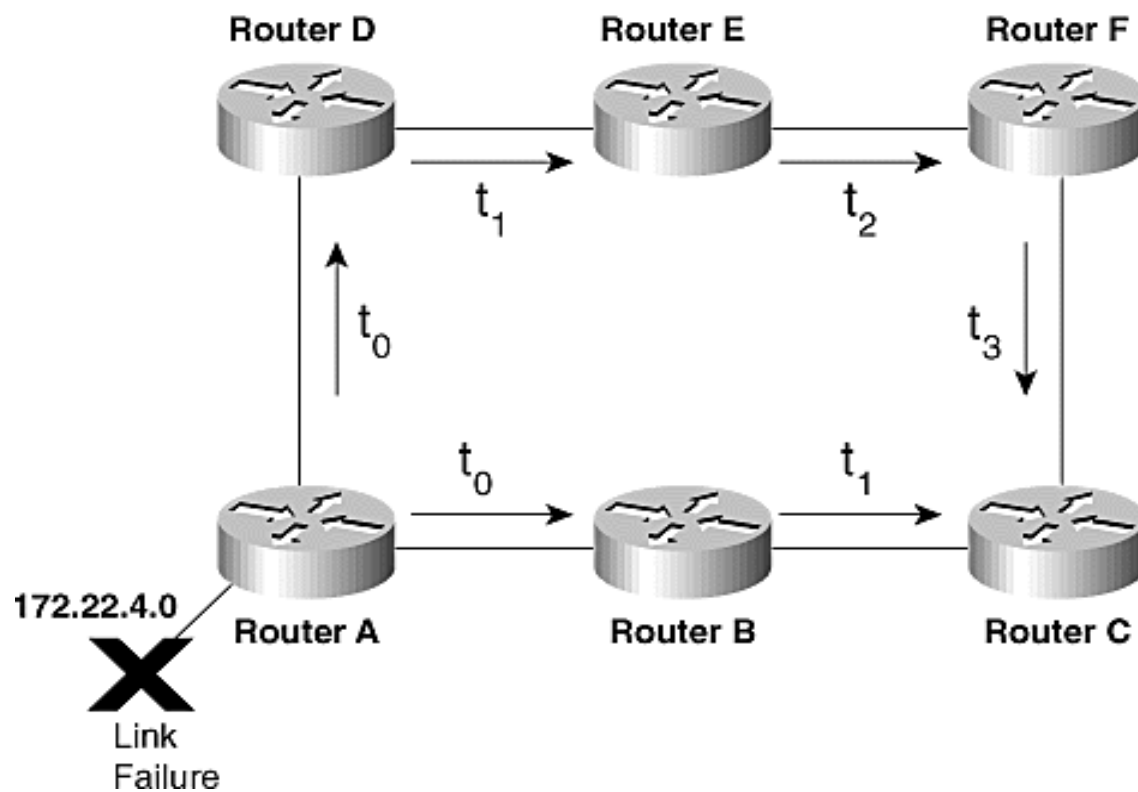
Ако поредният номер не е достатъчно голям, той може да се превърти.

Затова се използват 32-битови поредни номера.

Ако пристига 1 пакет/s, то за превъртане на номера ще са необходими около 137 години.

Ако пристига 1 пакет/10 s (на практика) - 1361 години.

# Пореден номер. Пример.



# Пореден номер. Пример.

Събитие: префикс 172.22.4.0 (рутер А) отпада.  
А разпространява (flood-ва) LSA на съседни В и D. В и D на техните съседни и т.н.

На рутер С: пристига LSA от В в момент  $t_1$ , и  
влиза в топол. БД на С и се отправя към F.

В  $t_3$ , пристига същото копие на LSA по пътя  
A-D-E-F-C.

Рутер С вече има LSA в топол. БД и няма да го  
отправи към В. Поредният номер на LSA от F е  
равен на LSA от В.

# Пореден номер. Пример.

Второ събитие: 172.22.4.0 пада и след това се вдига.

Рутер А изпраща LSA за падането с номер 166.

След това нов LSA за вдигането с номер 167.

Рутер С получава “падналото LSA” и след това “вдигнато LSA” по пътя A-B-C, но по-късно LSA от A-D-E-F-C path.

Какво щеше да стане, ако нямаше пореден номер (sequence number)?

Но LSA 167 вече е в топол. БД и закъсняло LSA 166 не може да заблуди рутер С.

# Поле за срок на годност

В полето за срок на годност маршрутизаторът-подател указва продължителността на интервала от време в секунди, през който пренасяната от него информация трябва да се счита за валидна.

Всеки маршрутизатор, който получи даден пакет намалява с единица стойността на това поле преди да го предаде към своите съседни.

# Поле за срок на годност

Освен това, след като маршрутизаторът запише данните от пакета в своята таблица, той продължава да намалява срока на годност на тези данни на всяка следваща секунда.

Ако **срока** на годност стане **0**, **данните се изтриват**.

По този начин се **премахва опасността остаряла информация** за състоянието на връзките **да се разпространява** и използва прекалено дълго време от маршрутизаторите.

# Прилагане на SPF алгоритъма

Е. W. Dijkstra: “Да се конструира дърво с минимална обща дължина между всичките  $n$  възела.”

(Дървото е граф с един и само един път между всеки два възела.)

Как става:

Дъгите се разделя на три множества:

# Прилагане на SPF алгоритъма

- I. Дъгите, присвоени на “**строящото се**” дърво;
  - II. Дъгите, от които се избира следващата за **добавяне в I**;
  - III. Останалите дъги.
- Възлите** се делят на две множества:
- A. Свързаните с дъги от **множество I**,
  - B. Останалите (една и само една **дъга от II** води до един от тези възли).



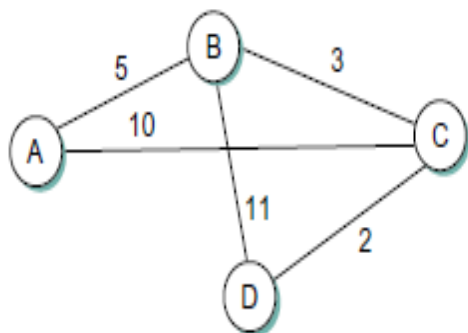
# Прилагане на SPF алгоритъма

*SPF Tree Database.* БД, съответства на **множество I.**

*Кандидатска БД.* БД, съответства на **множество II.**

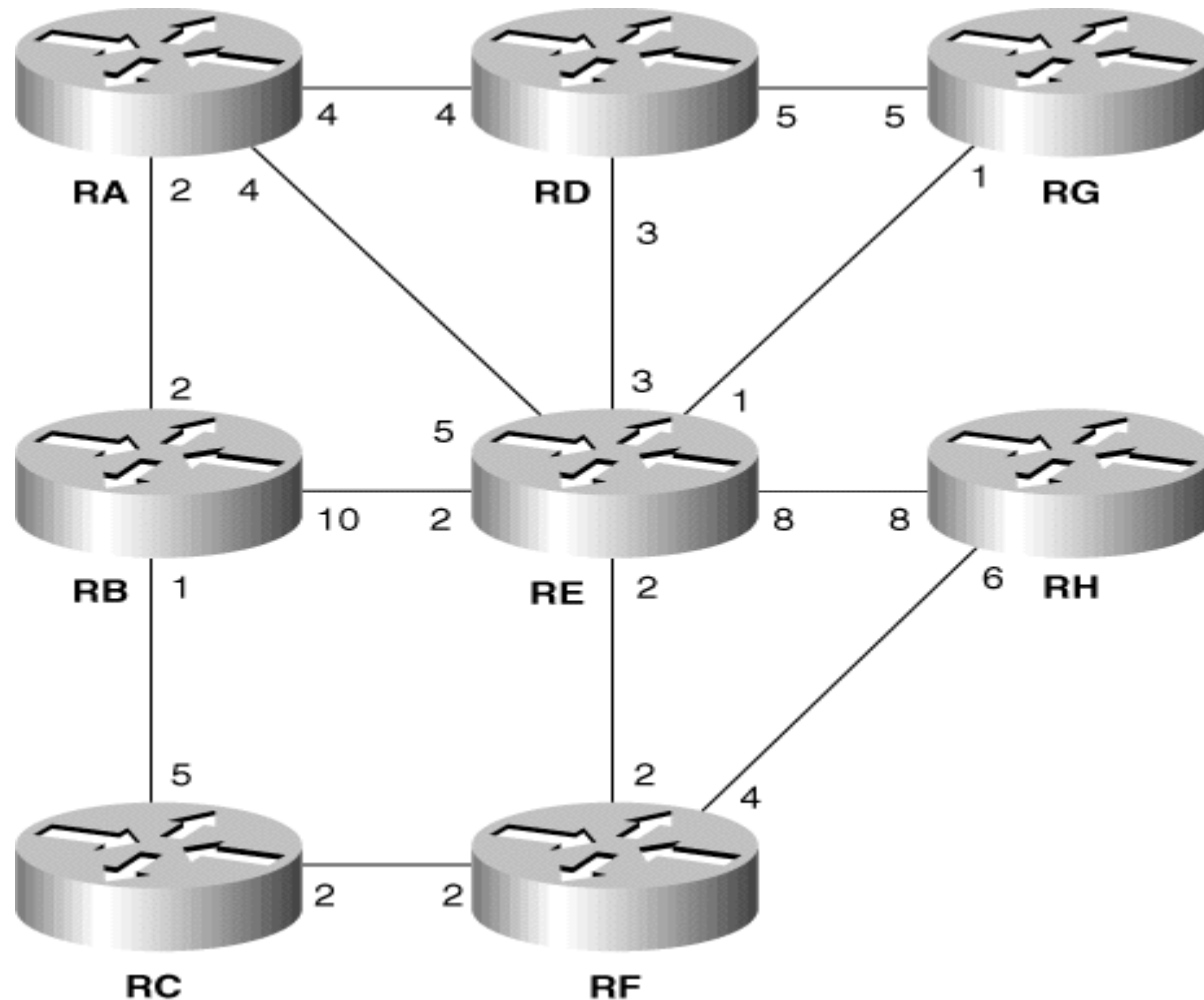
*Link State Database.* Топологичната БД, съответства на **множество III.**

# SPF алгоритъм. Прост пример.



1	(D,0,-)	
2	(D,0,-)	(B,11,B) (C,2,C)
3	(D,0,-) (C,2,C)	(B,11,B)
4	(D,0,-) (C,2,C)	(B,5,C) (A,12,C)
5	(D,0,-) (C,2,C) (B,5,C)	(A,12,C)
6	(D,0,-) (C,2,C) (B,5,C)	(A,10,C)
7	(D,0,-) (C,2,C) (B,5,C) (A,10,C)	

# SPF алгоритъм. По-сложен пример.



# Link State Database. Извлечение.

Router ID	Neighbor	Cost
RA	RB	2
RA	RD	4
RA	RD	4
RB	RA	2
RB	RC	1
RB	RE	10
RC	RB	5
RC	RF	2
RD	RA	4

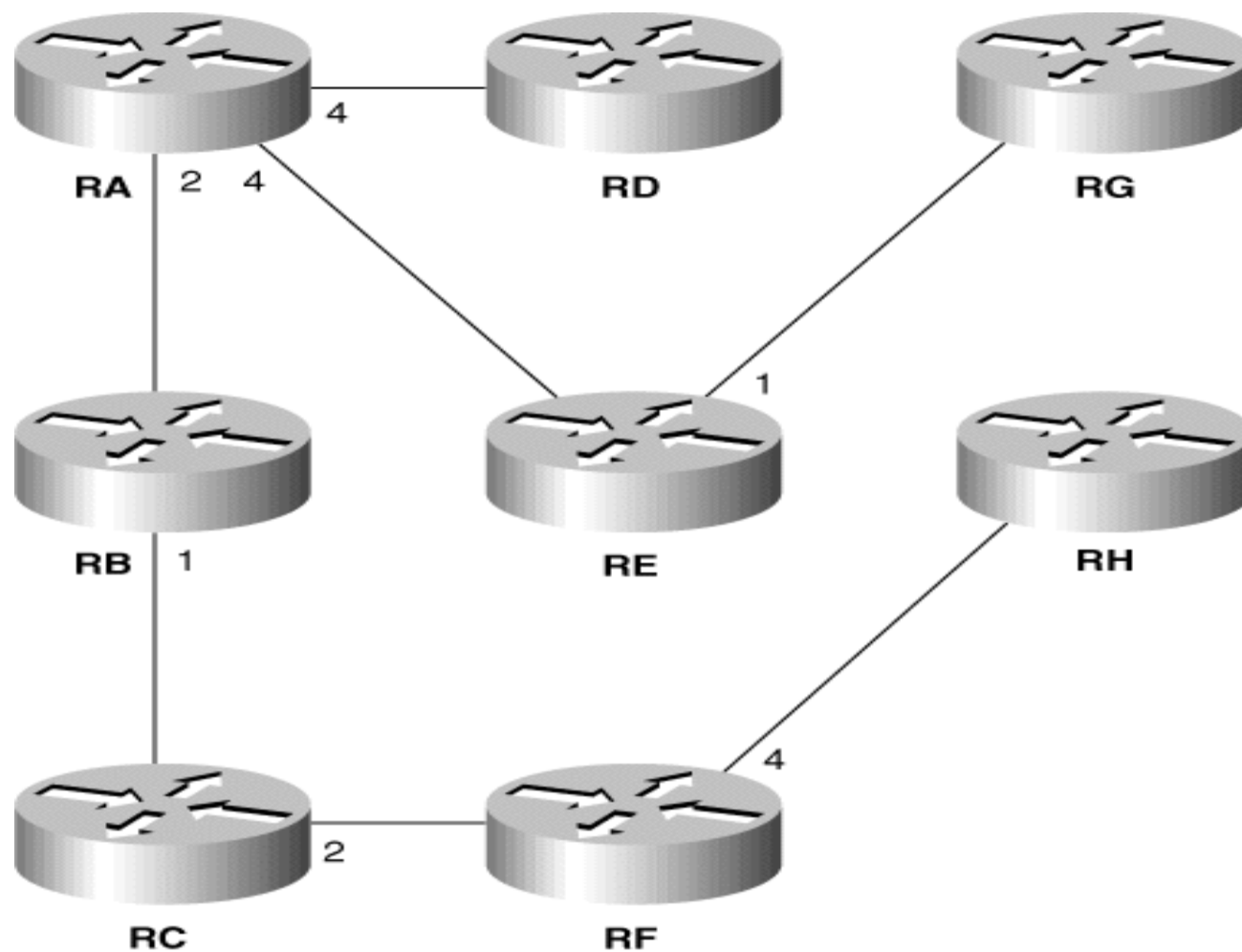
Канд. БД	Стойност до Root	SPF Tree	Описание
		RA,RA,0	Router A-root.
RA,RB,2 RA,RD,4 RA,RE,4	2 4 4	RA,RA,0	Добавят се съседите на RA
RA,RD,4 RA,RE,4 RB,RC,1 RB,RE,10	4 4 3	RA,RA,0 RA,RB,2	(RA,RB,2) с най-ниска стойност
RA,RD,4 RA,RE,4 RC,RF,2	4 4 5	RA,RA,0 RA,RB,2 RB,RC,1	(RB,RC,1) с най-ниска стойност от канд. БД

RA,RE,4 RC,RF,2 RD,RE,3 RD,RG,5	4 5 7 9	RA,RA,0 RA,RB,2 RB,RC,1 RA,RD,4	(RA,RD,4) и (RA,RE,4) са с 4 от RA; (RC,RF,2) = 5. (RA,RD,4) се добавя. (RD,RE,3) отпада.
RC,RF,2 RD,RG,5 RE,RF,2 RE,RG,1 RE,RH,8	5 9 6 5 12	RA,RA,0 RA,RB,2 RB,RC,1 RA,RD,4 RA,RE,4	(RA,RE,4) се добавя. Съседите на RE в Канд. БД. “Най-скъпият” път до RG отпада.
RE,RF,2 RE,RG,1 RE,RH,8 RF,RH,4	6 5 12 9	RA,RA,0 RA,RB,2 RB,RC,1 RA,RD,4 RA,RE,4 RC,RF,2	(RC,RF,2) се добавя, съседите му – в Канд. БД. “Най-скъпият” път до RH отпада.

# Пресмятане на SPF

RF,RH,4	RA,RA,0 RA,RB,2 RB,RC,1 RA,RD,4 RA,RE,4 RC,RF,2 RE,RG,1	(RE,RG,1) се добавя. Всички съседни на RG са в дървото, нищо не се добавя в Канд. БД.
	RA,RA,0 RA,RB,2 RB,RC,1 RA,RD,4 RA,RE,4 RC,RF,2 RE,RG,1 RF,RH,4	(RF,RH,4) се добавя. Няма повече кандидати. <b>Shortest path tree е готово.</b>

# Дървото на най-късия път





# Изчисляване на новите маршрути

Изчислените маршрути се записват в маршрутните таблици.

Необходимата памет за съхраняване на информацията за състоянието на връзките за мрежа с  $n$  маршрутизатори, всеки от които има по  $k$  съседни е пропорционална на  $nk$ .

Така големите по размер мрежи изискват използване на маршрутизатори с голям обем памет.

# Йерархична маршрутизация

С увеличаването на размерите на мрежата нараства обемът на маршрутните таблици, което изисква **повече памет и процесорно време** за тяхната обработка.

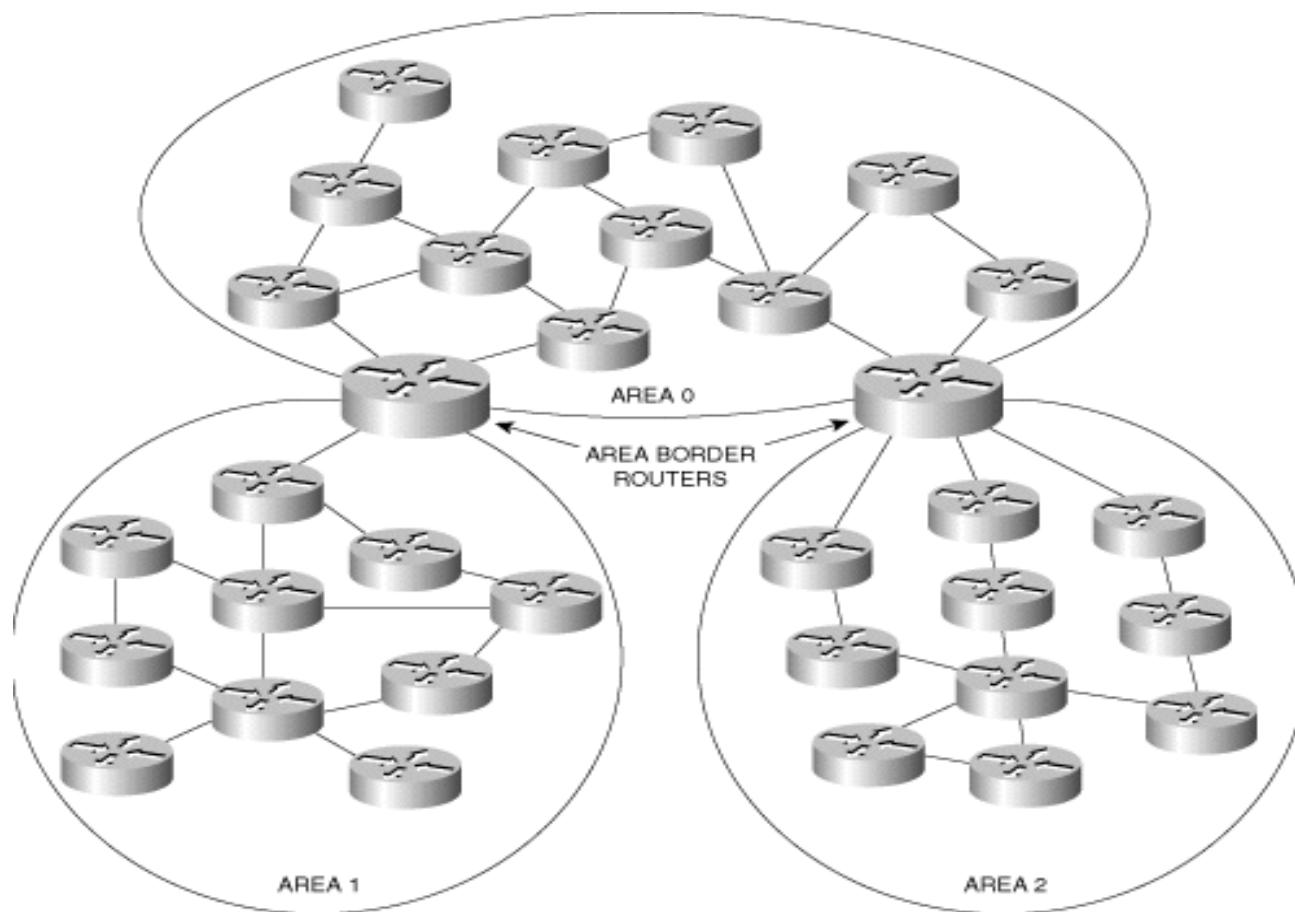
Това налага **въвеждането на йерархично маршрутизиране**, при което мрежата се разделя на **области**.

Маршрутизаторите в една област **знаят всичко** за вътрешната структура на **своята област**, но **не знаят** вътрешната структура на **останалите области**.

За по-големи мрежи може да е необходима **йерархия с повече от две нива**.

Йерархична маршрутизация е най-подходяща при протоколите със следене на състоянието на връзката.

# Области в LS протоколите



# Области в LS протоколите

Област (*area*) е подмножество от рутери в общата мрежа. (DV не може, не са рекурсивни.)

Разделянето на области се налага:

- колкото е **по-голяма БД** със състояние на връзките, изисква **повече памет** от DV.
- сложният алгоритъм - повече CPU време и цикли от DV.
- “*flooding*” на LS пакети vs. *bandwidth*, но OSPF и др. - частични updates, когато е необходимо.

# LS vs. DV

## Размяна на съобщения

LS: с  $n$  възела,  $E$  връзки,  $O(nE)$  изпратени съобщения

DV: разменят се само  $m/y$  съседи

## Скорост на сходимост

LS: висока

DV: ниска

Зацикляния

Проблем "Count-to-infinity"

Стабилност: рутер не е в ред?

LS:

Възелът рекламира неточна "*link cost*"

Всеки възел изчислява собствената си таблица

DV:

DV възел рекламира **неточен** "*path cost*"

Таблицата на даден възел се използва от други (**грешката се разпространява**)