14. Разпределена маршрутизация с дистантен вектор

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 следвашия възел (next hop) и изходящия интерфейс и

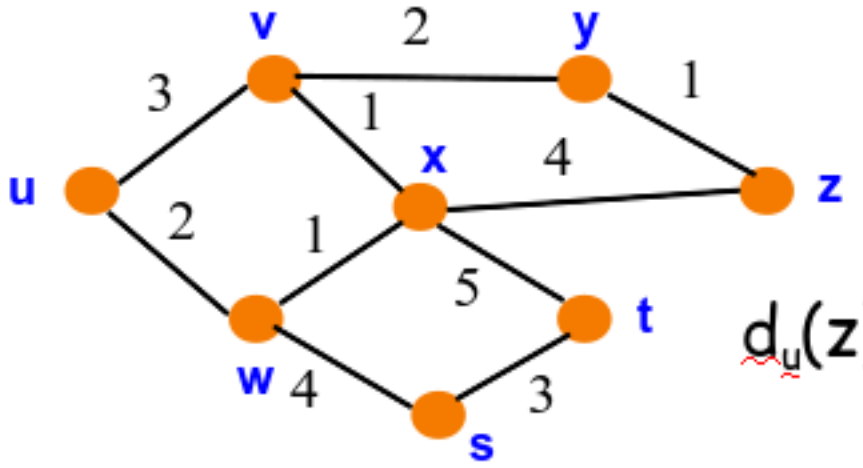
Разстояние (метрика), напр. брой възли до дестинацията (hop count).

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

DV използва само:

1. Посока или интерфейс, по който да се изпрати 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 периодично 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

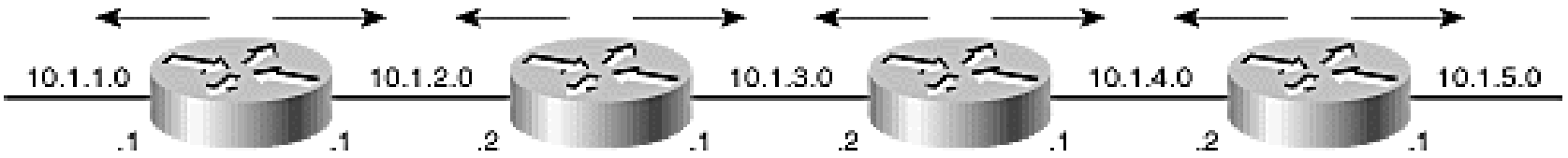
DV алгоритмите:

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

Но

- Периодическите update-и отнемат пропускателна способност от потребителите.

Distance Vector в действие



Router A

	NET	VIA	HOPS
t_0	10.1.1.0	--	0
	10.1.2.0	--	0
t_1	10.1.1.0	--	0
	10.1.2.0	--	0
	10.1.3.0	10.1.2.2	1
t_2	10.1.1.0	--	0
	10.1.2.0	--	0
	10.1.3.0	10.1.2.2	1
	10.1.4.0	10.1.2.2	2
t_3	10.1.1.0	--	0
	10.1.2.0	--	0
	10.1.3.0	10.1.2.2	1
	10.1.4.0	10.1.2.2	2
	10.1.5.0	10.1.2.2	3

Router B

	NET	VIA	HOPS
t_0	10.1.2.0	--	0
	10.1.3.0	--	0
t_1	10.1.2.0	--	0
	10.1.3.0	--	0
	10.1.1.0	10.1.2.1	1
	10.1.4.0	10.1.3.2	1
t_2	10.1.2.0	--	0
	10.1.3.0	--	0
	10.1.1.0	10.1.2.1	1
	10.1.4.0	10.1.3.2	1
	10.1.5.0	10.1.3.2	2
t_3	10.1.2.0	--	0
	10.1.3.0	--	0
	10.1.1.0	10.1.2.1	1
	10.1.4.0	10.1.3.2	1
	10.1.5.0	10.1.3.2	2

Router C

	NET	VIA	HOPS
t_0	10.1.3.0	--	0
	10.1.4.0	--	0
t_1	10.1.3.0	--	0
	10.1.4.0	--	0
	10.1.2.0	10.1.3.1	1
	10.1.5.0	10.1.4.2	1
t_2	10.1.3.0	--	0
	10.1.4.0	--	0
	10.1.2.0	10.1.3.1	1
	10.1.5.0	10.1.4.2	1
	10.1.1.0	10.1.3.1	2
t_3	10.1.3.0	--	0
	10.1.4.0	--	0
	10.1.2.0	10.1.3.1	1
	10.1.5.0	10.1.4.2	1
	10.1.1.0	10.1.3.1	2

Router D

	NET	VIA	HOPS
t_0	10.1.4.0	--	0
	10.1.5.0	--	0
t_1	10.1.4.0	--	0
	10.1.5.0	--	0
	10.1.3.0	10.1.4.1	1
t_2	10.1.4.0	--	0
	10.1.5.0	--	0
	10.1.3.0	10.1.4.1	1
	10.1.2.0	10.1.4.1	2
t_3	10.1.4.0	--	0
	10.1.5.0	--	0
	10.1.3.0	10.1.4.1	1
	10.1.2.0	10.1.4.1	2
	10.1.1.0	10.1.4.1	3

Метрика

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

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

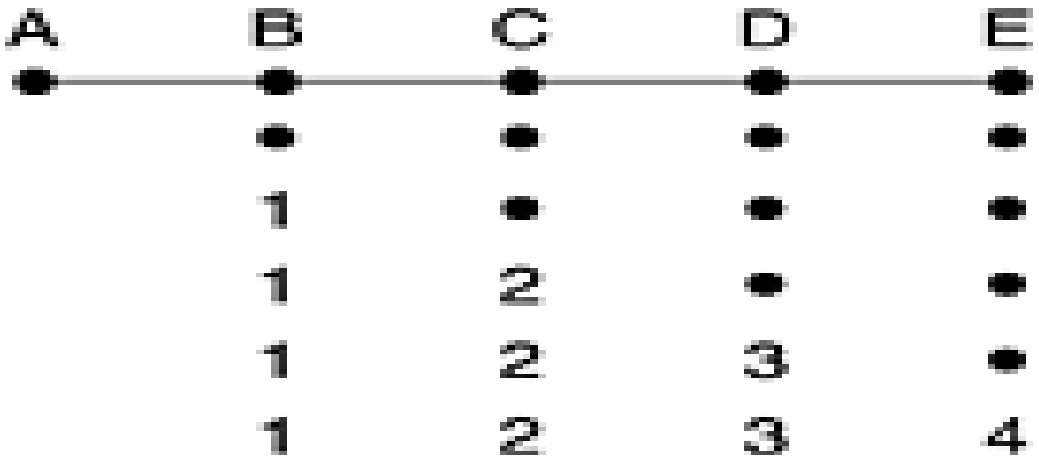
Ако метриката е натоварване на възела, разстоянието до всеки съсед е броя на пакетите в изходящата опашка към този пакет.

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 ∞ (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.

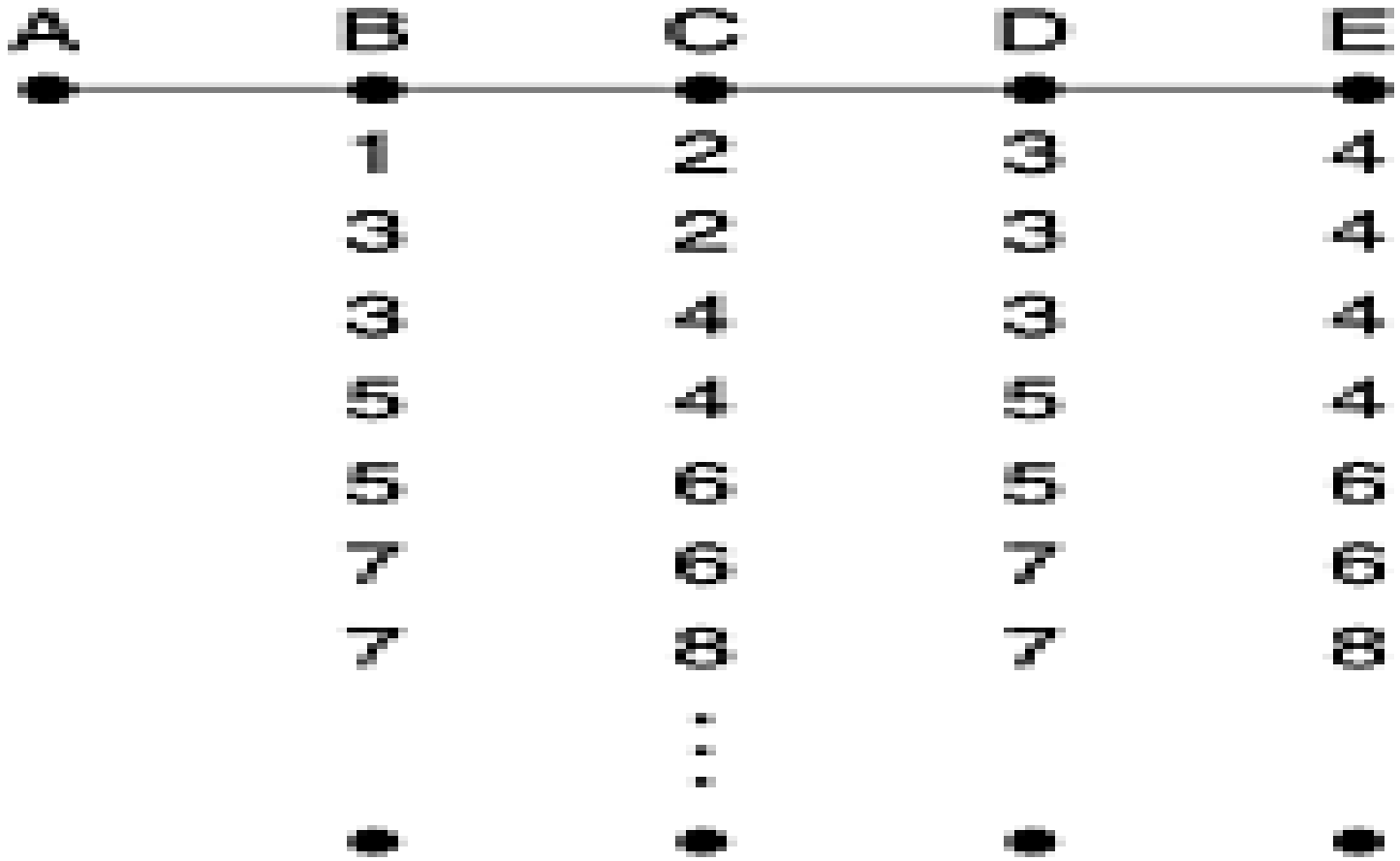
В този момент останалите маршрутизатори все още не са научили за включването на *A*. Това е отразено на втория ред по-горе.

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.

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.

Изключване 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*.

Изключване 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 машрутизатора не достигнат ∞.

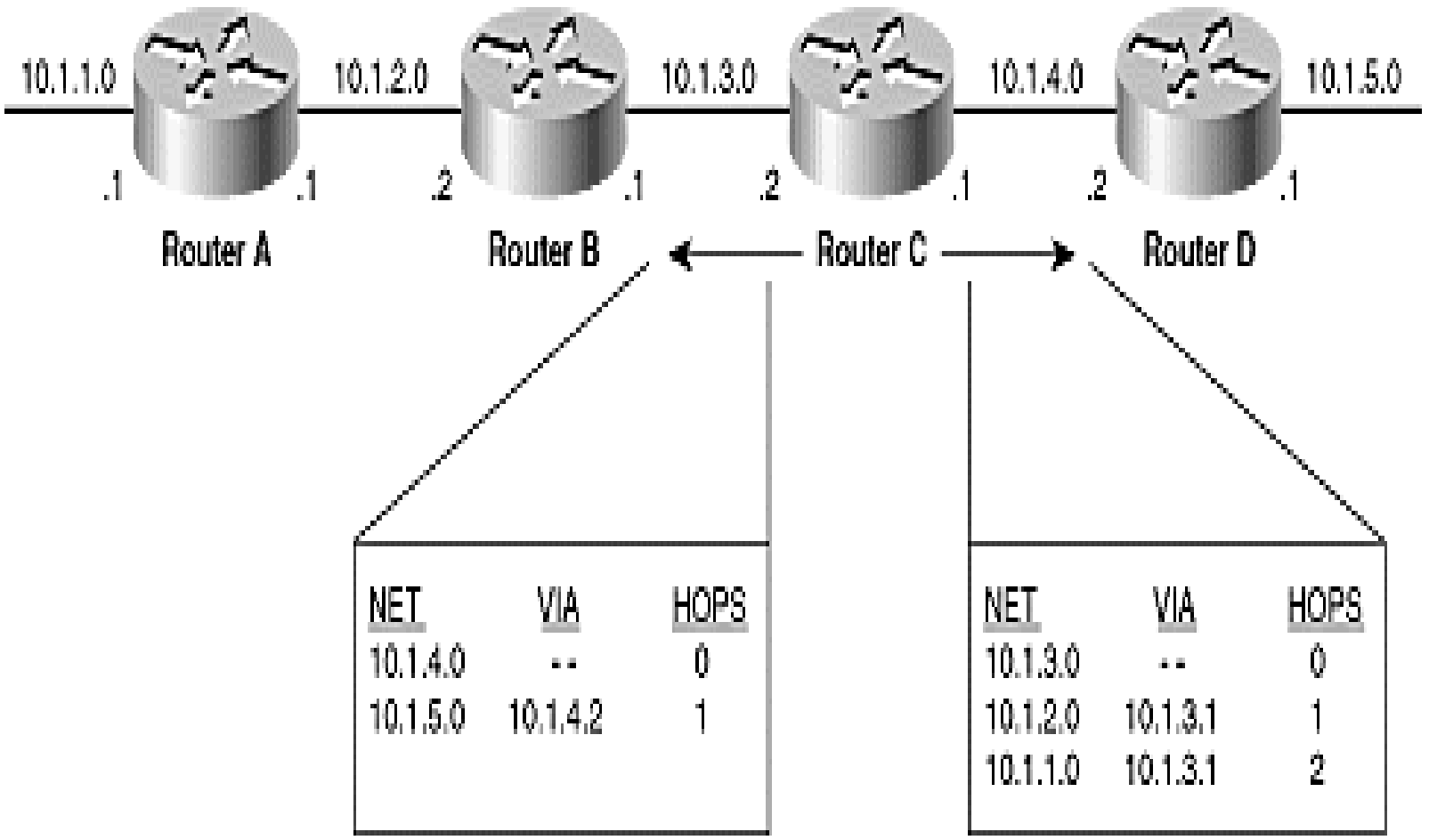
Този проблем се нарича **броене до безкрайност (count to infinity)**.

Split horizon

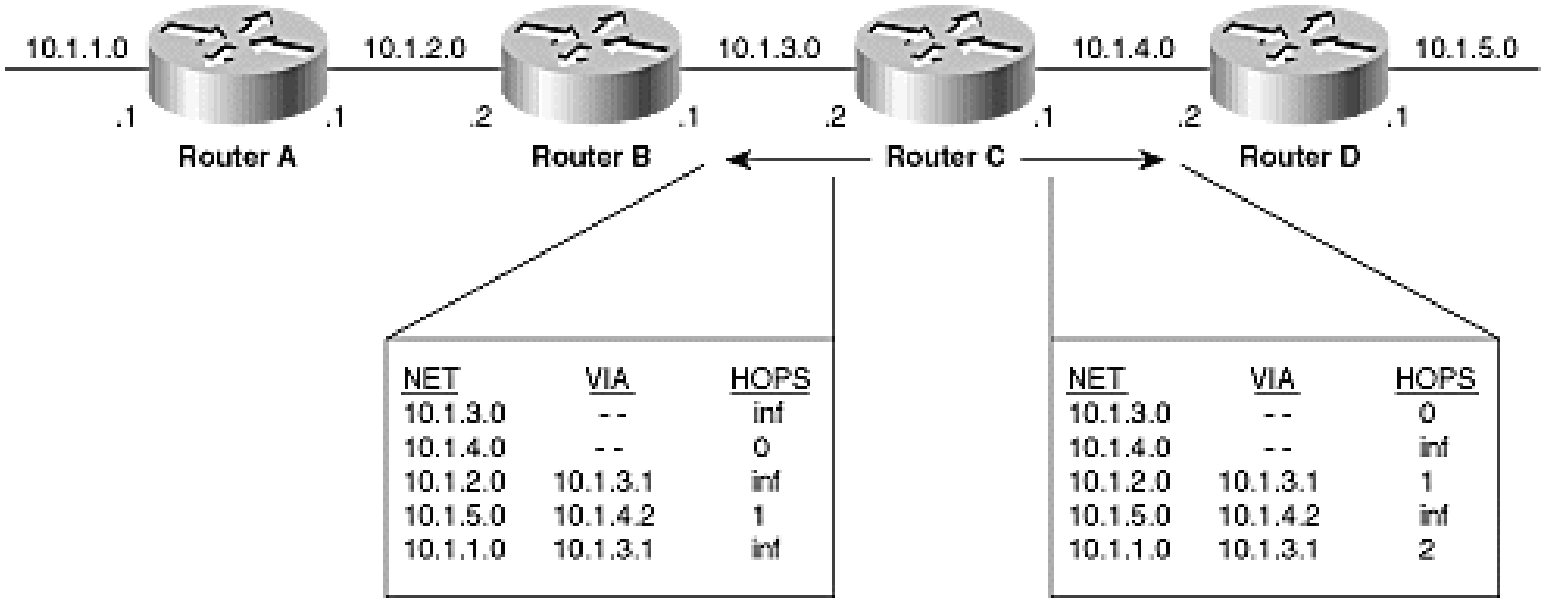
Едно частично негово решение е т.н. **разделяне на хоризонта** (**split horizon**).

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



Poison-Reverse (обратно поправяне)

Тази технология решава същите задачи, като Split horizon. Портовете, които са източник на информацията се маркират като недостижими (infinite distance).

Частично решение - **split horizon**

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

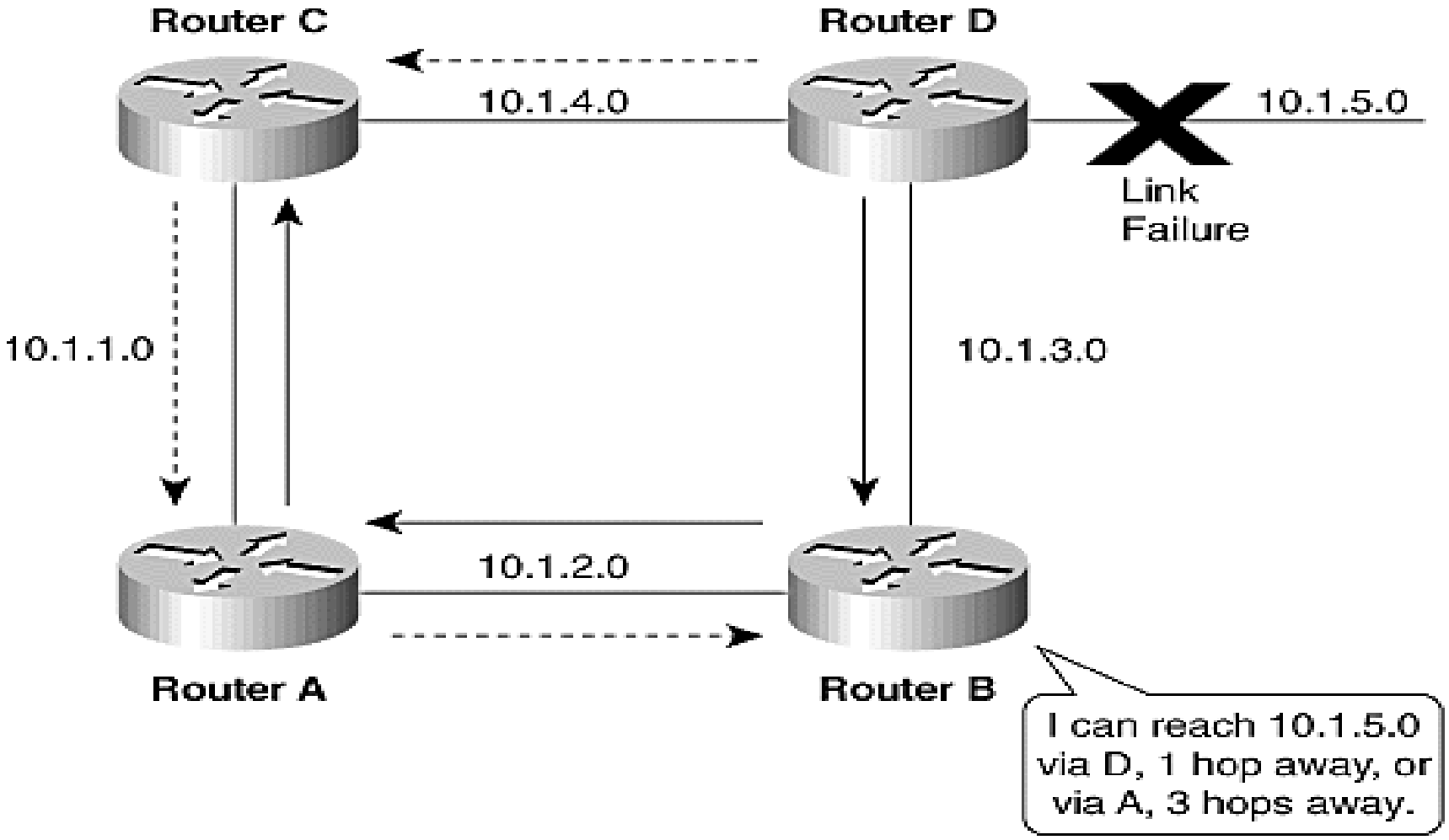
Но...

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

Двете технологии предотвратяват образуването на цикли/примки, включващи само два рутера. Ако са 3 или повече – не помагат.

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 $(\text{hop count})_{max}$

C “казва” на **A**: 5-hop маршрут до 10.1.5.0.

A “казва” на **B**: 6-hop маршрут до 10.1.5.0.

B “мисли”: “пътят на **A** до 10.1.5.0 се е удължил, но е единствен, ще го ползвам!"

B променя: hop count = 7, обновява **D** и “**въртележката**” се завърта наново.

Това е “броене до безкрайност” - *counting-to-infinity*: **hop count до 10.1.5.0** ще продължи да нараства до **∞**.

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 hop count (=15)

16-hop маршрут е недостижим (poisoned reverse),

но конвергенция: ≈ 7 минути при update = 30 s.

Недостатъчно, затова:

triggered updates и holddown timers

Triggered Updates

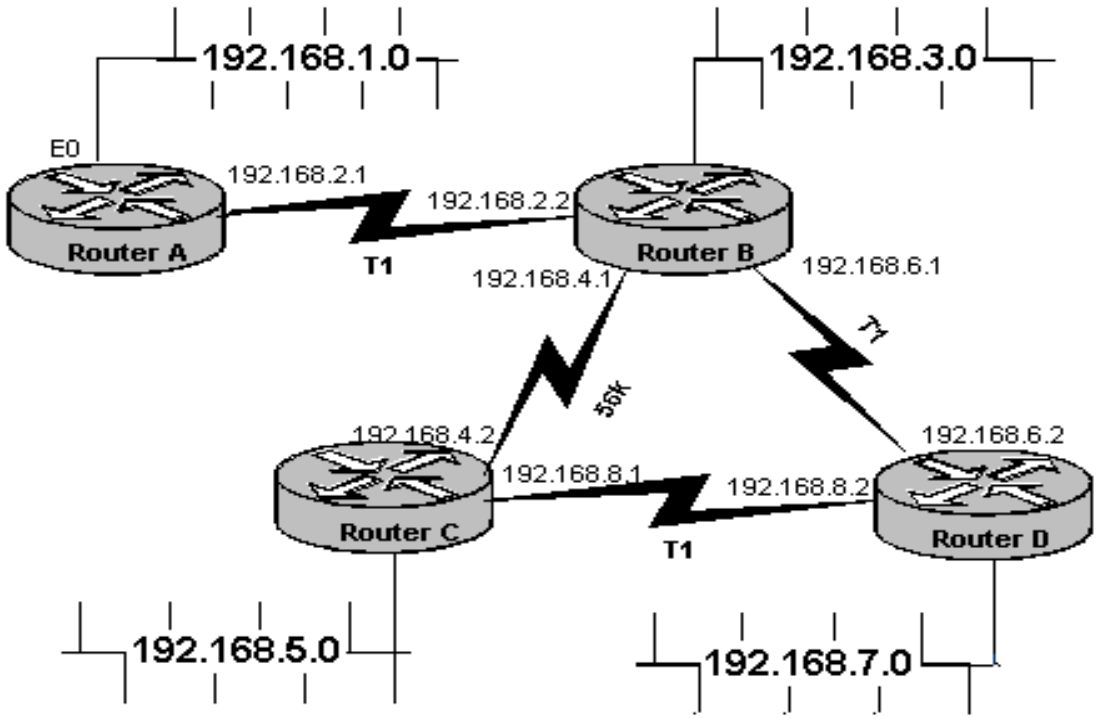
Ако (за добро или лошо) метриката се промени, рутерът веднага изпраща “update”, без да чака нулиране на таймера.

Реконвергирането ще стане много по-бързо, “counting to infinity” проблемът 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 B 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.

Защита срещу това: holddown таймер или timeout според RFCs 1058 u 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 изтичне 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 изтичне на garbage-collection timer, маршрутът окончателно се премахва от таблиците.