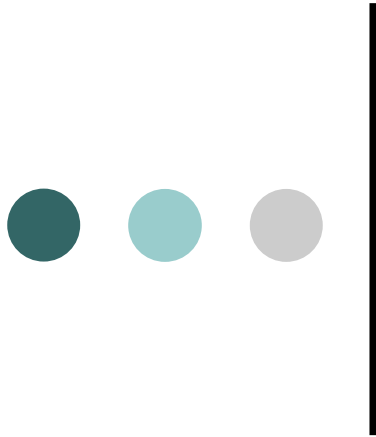


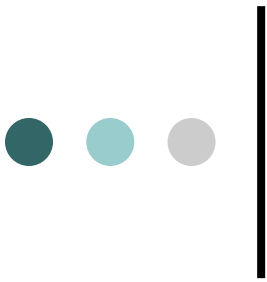
Multimedijalno inženjerstvo – master strukovne studije



Digitalni komunikacioni sistemi: **Lekcija 5: Rutiranje I**

zima 2018/2019

Branimir M. Trenkić



Rutiranje

- adresiranje, prosleđivanje, rutiranje -
 - rutiranje na osnovu vektora udaljenosti -
 - rutiranje na osnovu stanja veze -
- [Danas: bez otkaza u mreži]

Rutiranje i komutator

- Lekcija 2, slajd #19:

Komutator – osnovna funkcija

- Osnovna funkcija komutatora

- Multipleksiranje i demultipleksiranje** okvira koji pripadaju različitim host-host prenosnim sesijama (konekcijama)
- Određivanje linka** ili linkova po kojima dati okvir treba **proslediti**

- Ova funkcija je od esencijalne važnosti jer će često fizički link biti raspodeljen između više istovremenih konekcija

Rutiranje i komutator

- Lekcija 2, slajd #20:

Komutator - tri problema

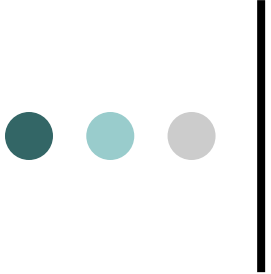
- Što se tiče te funkcije - mogu se uočiti **tri problema**:
 1. Prosleđivanje: Kada okvir dođe u komutator, komutator ga obrađuje u smislu (I) **određivanja korektnog odlaznog linka** i odlučuje (II) **kada treba slati** taj okvir na taj link
 2. Rutiranje
 3. Dodeljivanje resursa

Rutiranje i komutator

- Lekcija 2, slajd #21:

Komutator - tri problema

- **Rutiranje**: svaki komutator na neki način **mora da** odredi (**spozna**) **topologiju mreže**, kako bi bio u stanju **da formira korektnu strukturu podataka** na bazi koje će **vršiti prosleđivanje**
- **Proces** u kojem **komutatori sarađujući spoznaju topologiju mreže**, adaptirajući je u slučaju bilo kakvog otkaza u mreži – **naziva se rutiranje**
- Ovaj proces se ne odvija prilikom prenosa okvira već „u pozadini“



Rutiranje i komutator

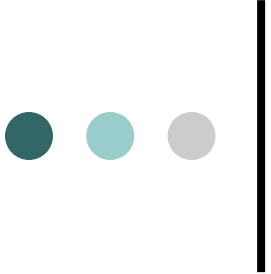
- Problem koji danas analiziramo je:

*„Šta **komutatori** (**i krajnje stanice**) u paketskim mrežama **treba da urade** kako bi se obezbedilo da paket poslat sa neke izvorišne stanice S , u mreži, dođe do njegovog krajnjeg odredišta D ?“*

- Reč „obezbedi“ implicira ***neku vrstu garancija***

Rutiranje i komutator

- Ovaj ***problem je dosta složen*** (više pod-problema) ***i nosi sa sobom mnoge izazove***
- Problem nalaženja putanje kroz mrežu je ***izazov iz sledećih razloga:***
 - Problem distribucije informacija
 - Efikasnost
 - Otkazi u mreži



Rutiranje i komutator

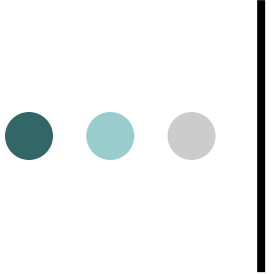
Distribucija informacija

- Svaka stanica ***poseduje saznanja samo o svojim lokalnim povezanostima***, t.j. ***neposrednim susedima*** u topologiji mreže
 - Pouzdano utvrđivanje i tog saznanja ***zahteva određene aktivnosti***
- Na nivou mreže mora se obezbediti način koji će omogućiti da se na osnovu ovih saznanja obezbedi ***globalna povezanost***

Rutiranje i komutator

Efikasnost

- **Pronađene putanje** moraju biti „**dobre**“
- Ne smeju biti neobično dugačke, što bi povećalo ukupno kašnjenje u prenosu
- Konkretno, predpostavićemo da **svaki link ima određenu cenu** (kojom se može modelovati npr. kašnjenje)
- **Problem** nalaženja putanje između izvorišta i krajnjeg odredišta paketa treba **svesti** na nalaženje putanje koja **minimizira ukupnu cenu**



Rutiranje i komutator

Otkazi u mreži

- U mrežama može doći do otkaza rada linkova i stanica
- Sa ovim događajima povezan je i ***vremenski period oporavka*** ovih elemenata

Rutiranje i komutator

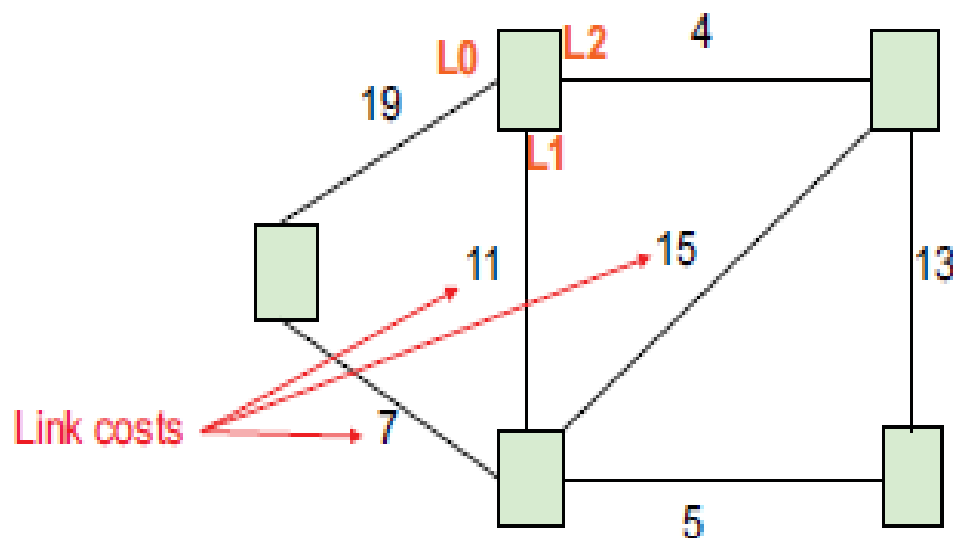
- *U rešavanju ovog problema* učestvuje „cela mreža“, t.j. sve stanice u mreži (krajnje + komutatori)
- *Ključnu ulogu igraju komutatori* i mi ćemo se fokusirati na taj deo rešavanja problema
- S obzirom da se informacije potrebne za rešavanje problema nalaze širom mreže (na različitim komutatorima), rešenje zahteva *sadejstvo svih komutatora u mreži*
- **Problem:** Distribuirane metode za nalaženje putanje u mreži

Rutiranje osnovni pojmovi

Problem: Distribuirane metode za nalaženje putanje u mreži

○ Rešenje iz tri dela:

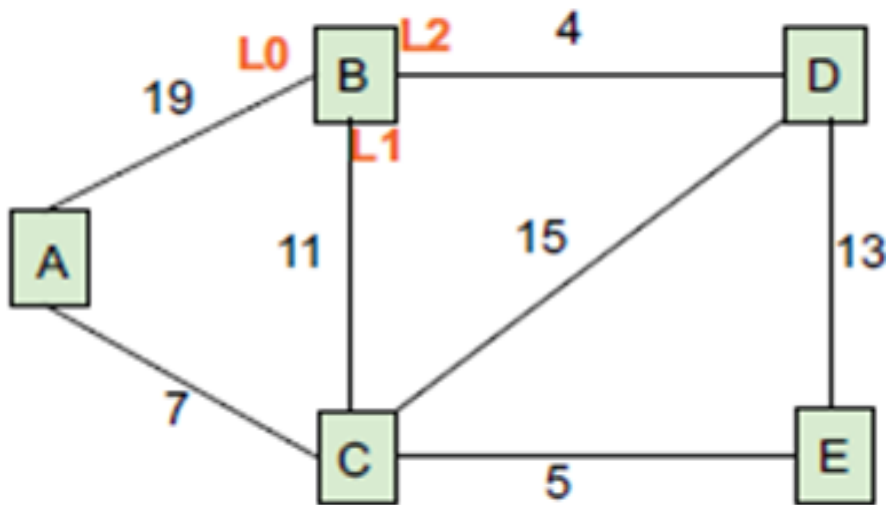
- *Adresiranje*
- *Prosleđivanje*
- *Rutiranje*



Rutiranje osnovni pojmovi

Problem: Distribuirane metode za nalaženje putanje u mreži

- **Adresiranje** (imenovanje tačaka u mreži)
 - Jedinstveni identifikator za **globalno adresiranje**
 - Identifikator linka **do suseda**



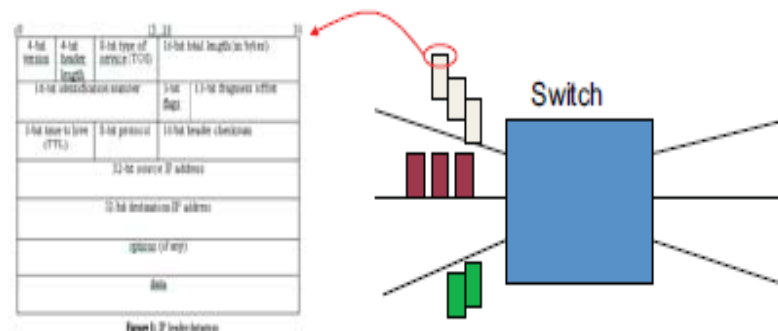
Rutiranje osnovni pojmovi

Problem: Distribuirane metode za nalaženje putanje u mreži

- **Prosleđivanje** (obrada paketa u procesu komutacije)
- **Rutiranje** (formiranje i ažuriranje struktura podataka koje omogućuju realizaciju prosleđivanja)
 - To su sve **funkcije sloja mreže**

Prosleđivanje

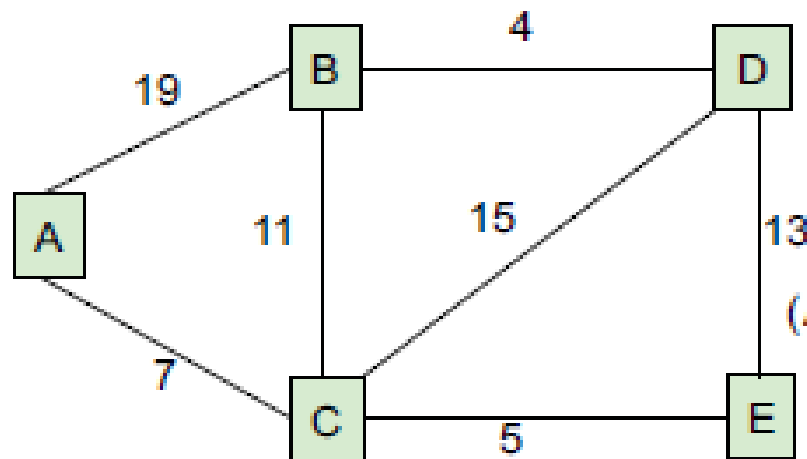
- Tri konceptijski vrlo jednostavne funkcije
- ***lookup(dst_addr)*** u ruting tabeli vraća ***rutu*** (engl. *route*) (t.j. ***odlazni link***) za paket
- ***enqueue(packet, link_queue)***
- ***send(packet)*** po odlaznom linku
- Neke dodatne provere ***pre*** funkcije ***enqueue***
 - ***Dekrementiranje brojača skokova*** (TTL polje); ako je = 0, odbaciti paket
 - ***Rekalkulacija kontrolne sume*** (u IP, kontrolne sume zaglavlja)



Rutiranje na osnovu najkraće putanje

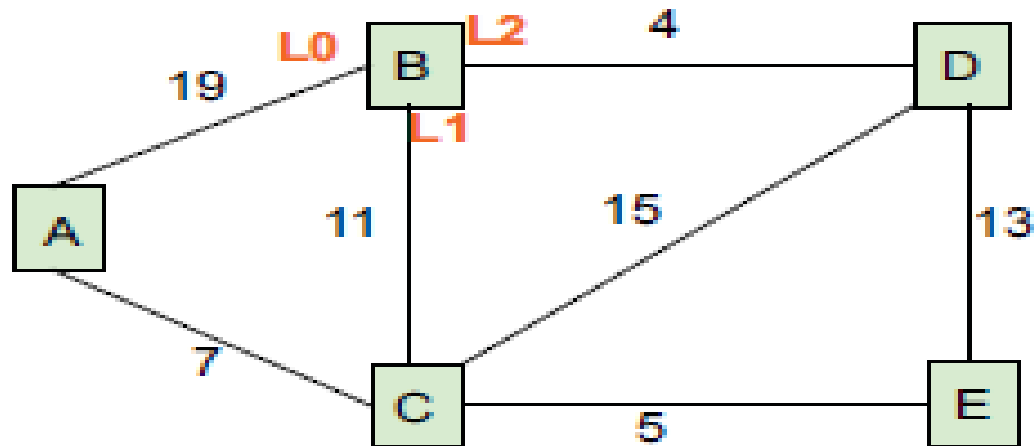
- Svaki komutator (ruter) želi na nađe **putanju (rutu) sa minimalnom ukupnom cenom** do drugih čvorova

- Koristimo izraz „**najkraća putanja**“ iako **nas zanima minimalna cena** (a ne minimalni # skokova)



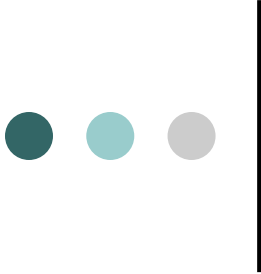
- Nekoliko mogućih **distribuiranih pristupa**:
 - **Vektorski protokoli**, protokol **na bazi vektora udaljenosti** (distance vector, **DV**)
 - **Protokoli na bazi stanja linka** (link-state, **LS**)

Struktura tabelle rutiranja



Routing table @ node B

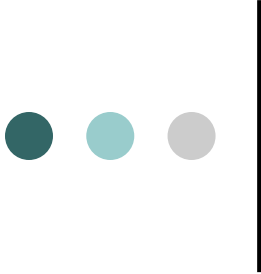
Destination	Link (next-hop)	Cost
A	ROUTE L1	18
B	Self	0
C	L1	11
D	L2	4
E	ROUTE L1	16



Distribuirano rutiranje: Zajednički pristup

○ *Određivanje „živih“ suseda*

- *Zajedničko za obe kategorije* protokola (DV + LS)
- *HELLO protokol* (periodično)
 - Slanje HELLO paketa *ka svim susedima* kako bi saznao *ko je na drugom kraju odlaznog linka*
 - Koristi primljene HELLO pakete za *formiranje liste suseda* koja se sastoji od *uređene trojke* (*vremenska oznaka, adresa suseda, link*)
 - Ponavlja se *periodično*: ako se ne čuju neko izvesno vreme – desio se otkaz linka, tako da se taj sused uklanja sa liste suseda



Distribuirano rutiranje: Zajednički pristup

- **Korak oglašavanja** (periodično)
 - **Slanje** informacija **do svih suseda**
 - Koriste se **za identifikaciju povezanosti** & **cene dostupnosti** neke tačke u mreži
- **Integracioni korak**
 - **Formiranje ruting tabele** na osnovu informacija iz oglašavanja
 - Problem suočavanja sa starim podacima

Rutiranje na bazi vektora udaljenosti

○ **DV oglašavanje**

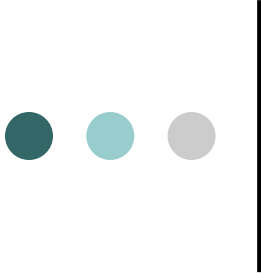
- Slanje informacija iz stavki ruting tabele:
(dest, cost)
- **Inicijalno** samo **(self, 0)**

○ **DV integracioni korak (Bellman-Ford)**

- Za svaku stavku **(dest, cost)** iz oglašavanja suseda
 - Proračun cene dostupnosti tačke **(dest)** preko **suseda koji je oglasio stavku**:
 - Rezultat:- **(dest, my_cost)**
 - **$my_cost = cost_in_advertisement + link_cost$**

Rutiranje na bazi vektora udaljenosti

- DV **integracioni korak** (*Bellman-Ford*)
 - Da li ruter **već šalje pakete** ka **dest preko tog suseda**?
 - Videti **da li se link slaže** sa onim što se već nalazi u ruting tabeli
 - Ako se linkovi poklapaju, **ažurirati cenu u ruting tabeli** sa **my_cost**
 - Inače, **ako je my_cost manji** od cene u ruting tabeli
 - **Sused nudi bolju putanju!** Koristi je...
 - **Ažuriraj ruting tabelu** tako da se paketi **do dest** sada šalju **preko ovog suseda**



Rutiranje na bazi vektora udaljenosti

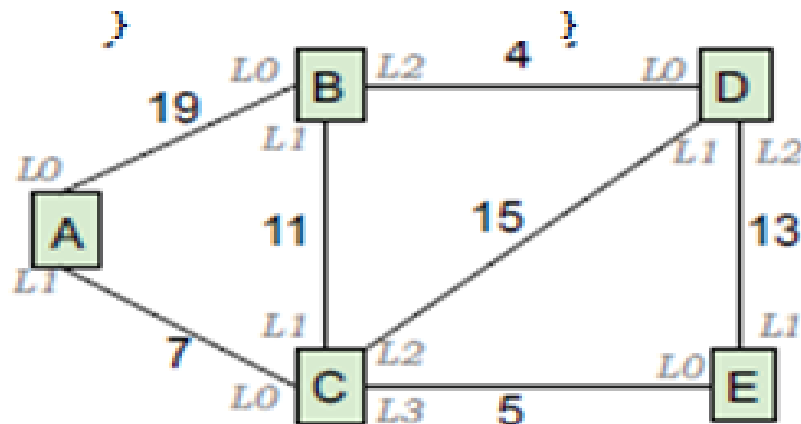
- DV ***integracioni korak*** (Bellman-Ford)
 - ***Ako odredište dest još uvek nije u tabeli,***
ažurira se tabela tako što je ruter ***dodaje tabeli*** i
sortira tabelu saglasno odredišnoj adresi

DV Primer

Inicijalno:

```
{ 'A': (None, 0),  
  'B': (L0, 19),  
  'C': (L1, 7)  
}
```

```
{ 'A': (L0, 19),  
  'B': (None, 0),  
  'C': (L1, 11),  
  'D': (L2, 4)  
}  
{ 'B': (L0, 4),  
  'C': (L1, 15),  
  'D': (None, 0),  
  'E': (L2, 13)  
}
```



```
{ 'A': (L0, 7),  
  'B': (L1, 11),  
  'C': (None, 0),  
  'D': (L2, 15),  
  'E': (L3, 5)  
}  
{ 'C': (L0, 5),  
  'D': (L1, 13),  
  'E': (None, 0)  
}
```

Ruter A: ažurirati rute do B_C, D_C, E_C

Ruter B: ažurirati rute do A_C, E_C

Ruter C: bez ažuriranja

Ruter D: ažurirati rutu do A_C

Ruter E: ažurirati rute do A_C, B_C

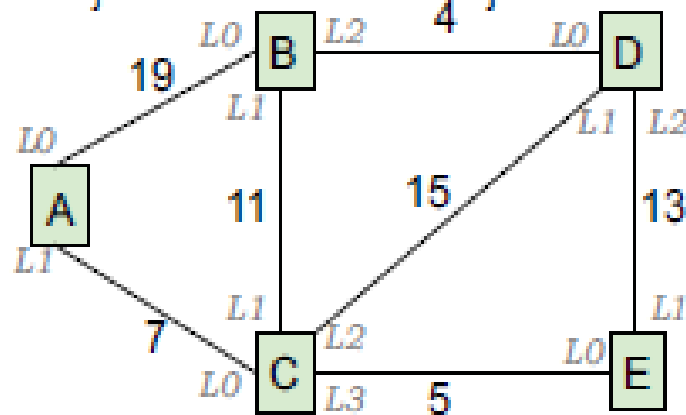
DV Primer

Konačno stanje:

```
{ 'A': (None, 0),
  'B': (L1, 18),
  'C': (L1, 7),
  'D': (L1, 22),
  'E': (L1, 12) }
```

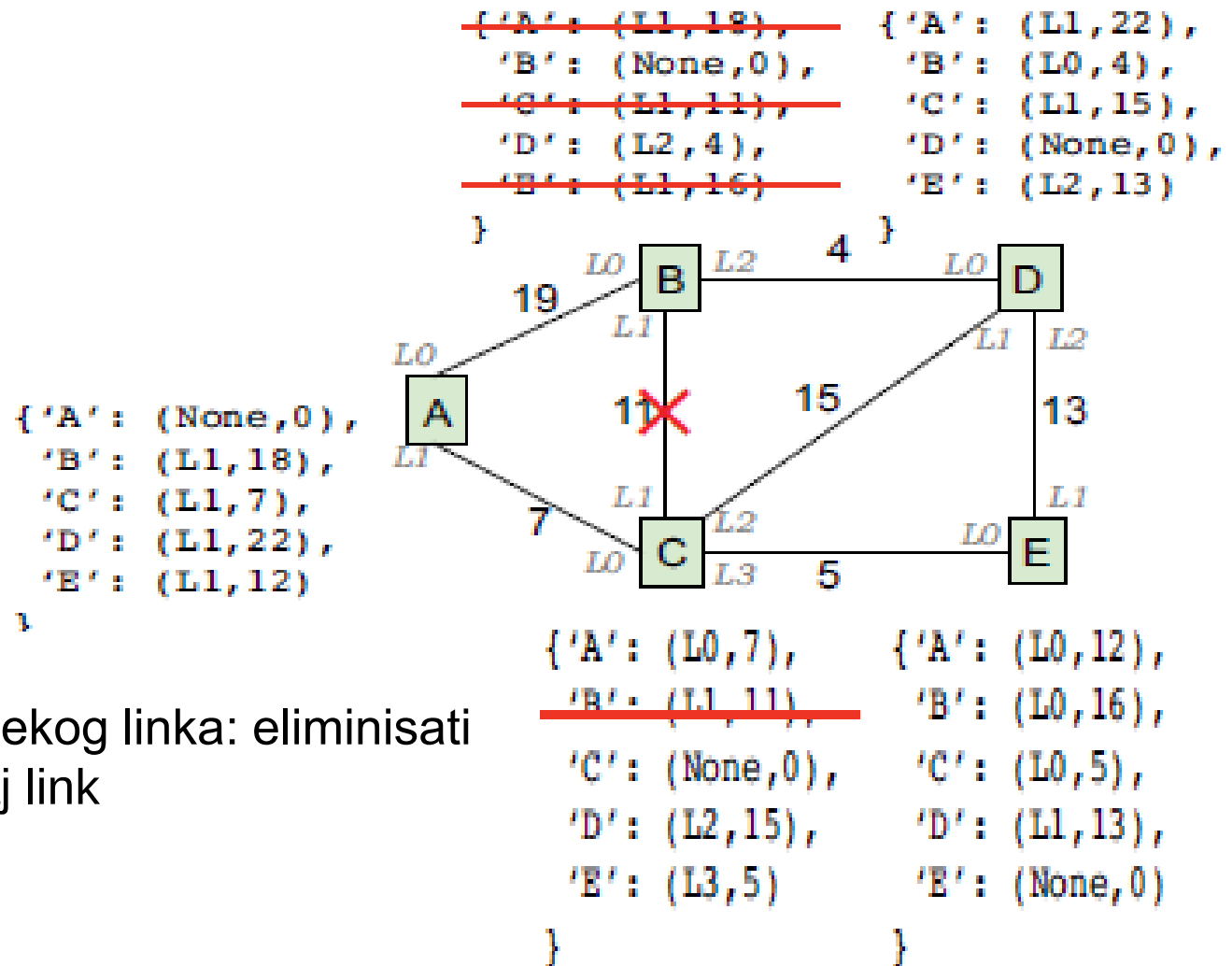
Ruter A: bez ažuriranja
 Ruter B: bez ažuriranja
 Ruter C: bez ažuriranja
 Ruter D: bez ažuriranja
 Ruter E: bez ažuriranja

```
{ 'A': (L1, 18),
  'B': (None, 0),
  'C': (L1, 11),
  'D': (L2, 4),
  'E': (L1, 16) }
{ 'A': (L1, 22),
  'B': (L0, 4),
  'C': (L1, 15),
  'D': (None, 0),
  'E': (L2, 13) }
```



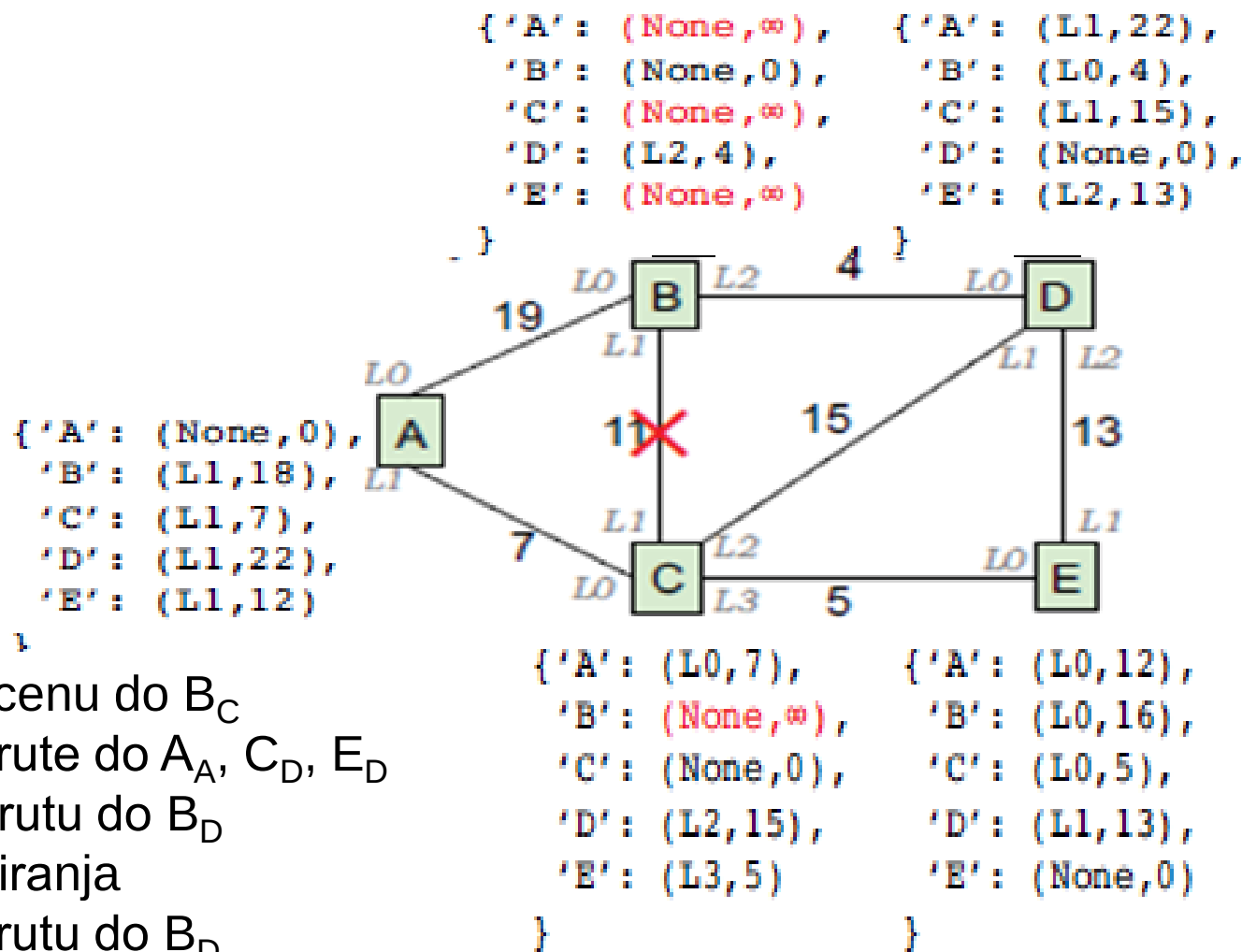
```
{ 'A': (L0, 7),
  'B': (L1, 11),
  'C': (None, 0),
  'D': (L2, 15),
  'E': (L3, 5) }
{ 'A': (L0, 12),
  'B': (L0, 16),
  'C': (L0, 5),
  'D': (L1, 13),
  'E': (None, 0) }
```

DV Primer: Otkaz linka



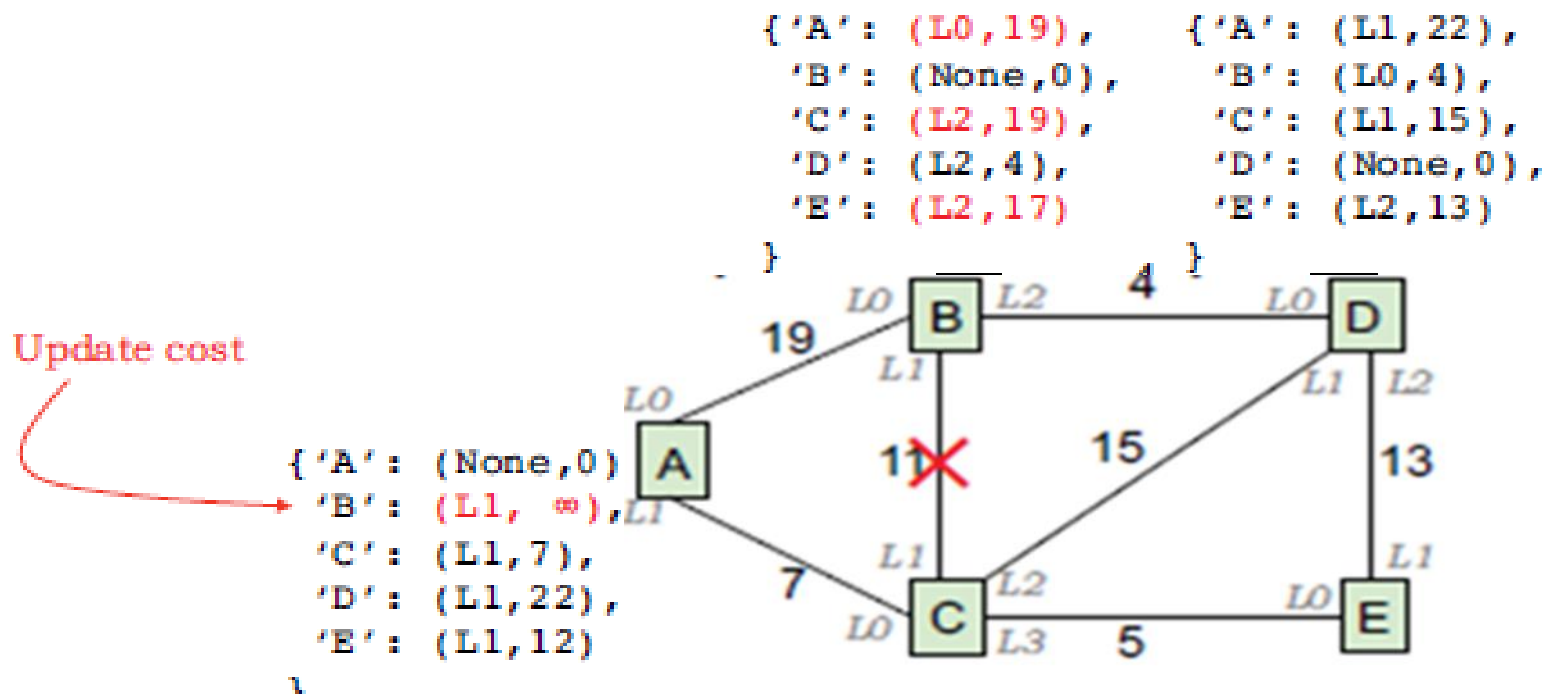
U slučaju otkaza nekog linka: eliminisati rute koje koriste taj link

DV Primer: Otkaz linka



- Ruter A: ažurirati cenu do B_C
- Ruter B: ažurirati rute do A_A, C_D, E_D
- Ruter C: ažurirati rutu do B_D
- Ruter D: bez ažuriranja
- Ruter E: ažurirati rutu do B_D

DV Primer: Otkaz linka



Ruter A: ažurirati cenu do B_B

Ruter B: bez ažuriranja

Ruter C: bez ažuriranja

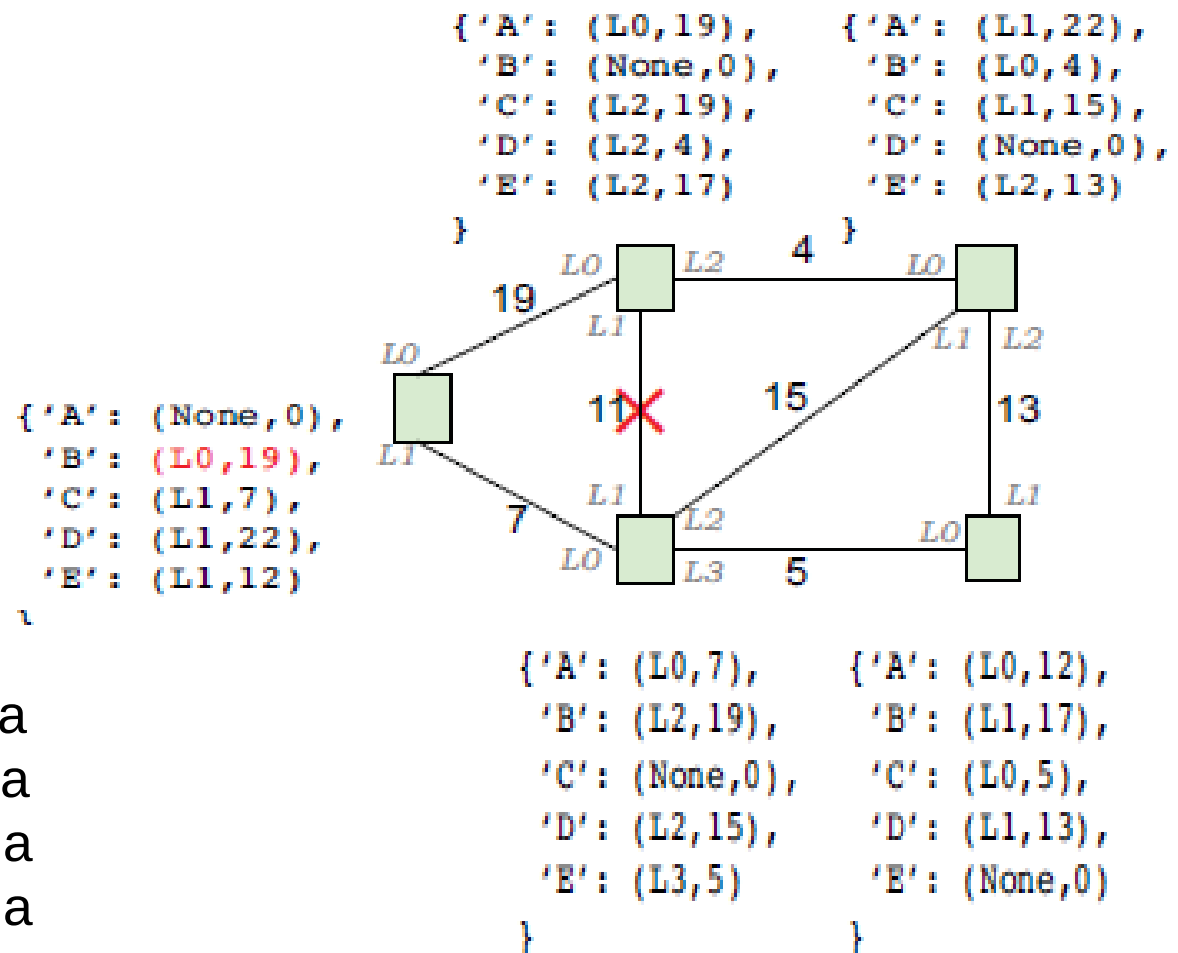
Ruter D: bez ažuriranja

Ruter E: bez ažuriranja

{ 'A': (L0, 7), 'B': (L2, 19), 'C': (None, 0), 'D': (L2, 15), 'E': (L3, 5) }

{ 'A': (L0, 12), 'B': (L1, 17), 'C': (L0, 5), 'D': (L1, 13), 'E': (None, 0) }

DV Primer: konačno stanje



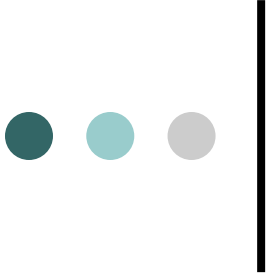
Ruter A: bez ažuriranja

Ruter B: bez ažuriranja

Ruter C: bez ažuriranja

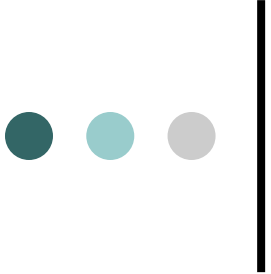
Ruter D: bez ažuriranja

Ruter E: bez ažuriranja



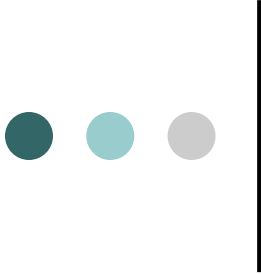
Korektnost i performanse

- Načelo ***optimalnosti sub-struktura*** osnova za korektnost algoritama rutiranja
 - “***Predpostavimo da najkraći put od X do Y prolazi kroz Z. U tom slučaju, deonica od X do Z mora biti najkraći put***”
- Dokaz Bellman-Ford-ovog algoritma ***preko indukcije broja koraka na najkraćem putu*** (putu minimalne cene)
 - Jednostavno za ne-negativne cene i sinhroni model
 - Komplikovano u slučaju distribuiranog asinhronog modela



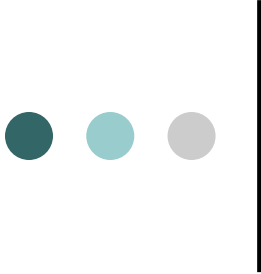
Korektnost i performanse

- Koliko je ***vreme konvergencije protokola*** za rutiranje na bazi vektora udaljenosti?
- ***Vreme je proporcionalno najvećem broju koraka*** (skokova) u odnosu na sve putanje najmanjih cena



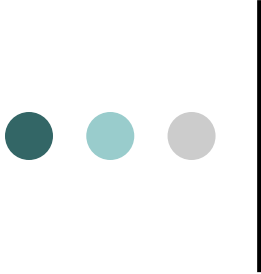
Rutiranje na bazi stanja veze

- Bazira se na **drugačijoj ideji**
- Protokol na bazi vektora udaljenosti:
 - Oglašava se jedino **najbolja cena do svakog** odredišta u mreži
- Protokol rutiranja na bazi stanja veze:
 - U fazi oglašavanja, svaki ruter oglašava jedino **informacije o njegovim susedima** (t.j. cenama linkova do njih)
 - Prijemom ovakvog oglašavanja, ruter difuzijom prosleđuje ovo oglašavanje do svojih suseda. Ovaj proces se naziva **plavljenje (flooding)**



Rutiranje na bazi stanja veze

- **Rezultat plavljenja**:- **mapa celokupne mreže** u svakom ruteru
- Mapa se sastoji od **(I)** čvorova (ruteri) u mreži i **(II)** trenutno operativnih linkova (koji su evidentirani pomoću HELLO protokola)
- Opremljen kompletnom mapom mreže, svaki ruter u mreži može izvršiti **centralizovan proračun nalaženja najkraće rute** (putanje) do svakog odredišta u mreži.



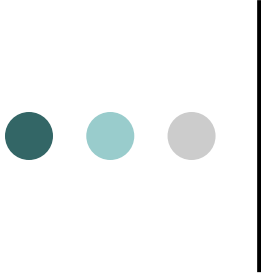
Rutiranje na bazi stanja veze

- Suprotno od ovoga, u protokolu na bazi vektora udaljenosti:
 - a) Sam **proračun ruta je distribuiran**,
 - b) **Bez nekog značajnijeg poznavanja topologije** čitave mreže u bilo kom od rutera
- Protokol rutiranja na bazi stanja veze **distribuirana informaciju o stanju svojih linkova** (odatle potiče i ime) i svojih suseda ka svim ruterima u mreži
- Sve dok ruteri imaju konzistentnu sliku topologije, prosleđivanje će raditi na željeni način

Rutiranje na bazi stanja veze

○ *Korak oglašavanja*

- Šalju se **informacije o njegovim vezama** do suseda (**link-state advertisement**, **LSA paket**):
[origin_addr, seq#, [(nbhr1, linkcost1), (nbhr2, linkcost2),....]]
 - **origin_addr** – adresa rutera koji je formirao LSA
 - **seq#** - redni broj formiranog LSA (od 0, +1)
 - **nbhr** – trenutno aktivni sused
 - **linkcost** – cena odgovarajućeg linka
- Raditi to **periodično** (održavanje u životu, oporavak od izgubljenog LSA)

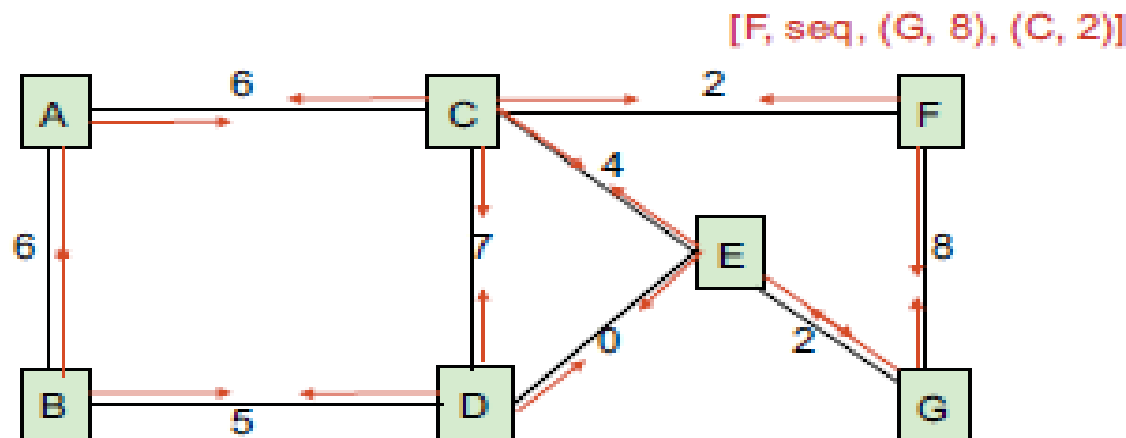


Rutiranje na bazi stanja veze

○ *Korak oglašavanja*

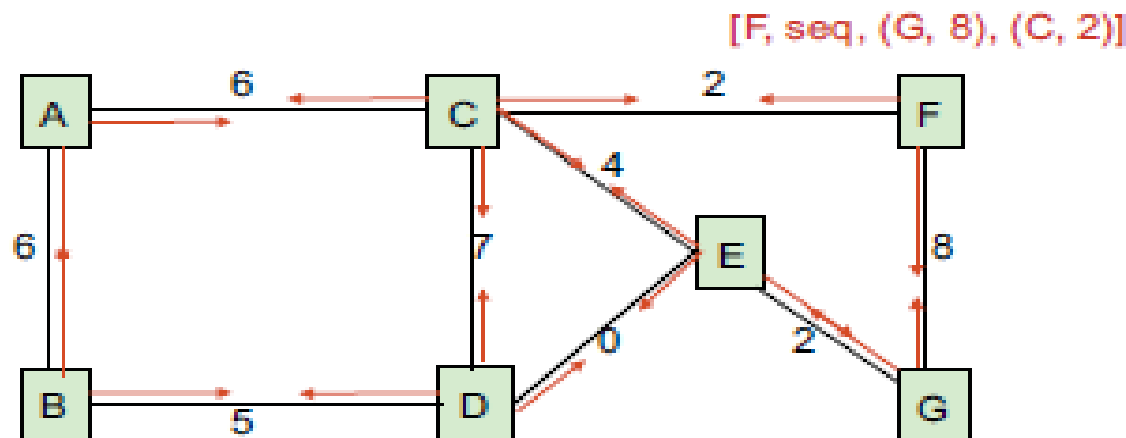
- Svaki susedni ruter koji primi ovakvo oglašavanje ***difuzijom ga prosleđuje*** sada do svojih neposrednih suseda itd.....
- Ovaj proces se naziva plavljenje (***LSA flooding***)
- ***Rezultat plavljenja:- mapa celokupne mreže (link-state baze podataka)*** u svakom ruteru

LSA plavljenje



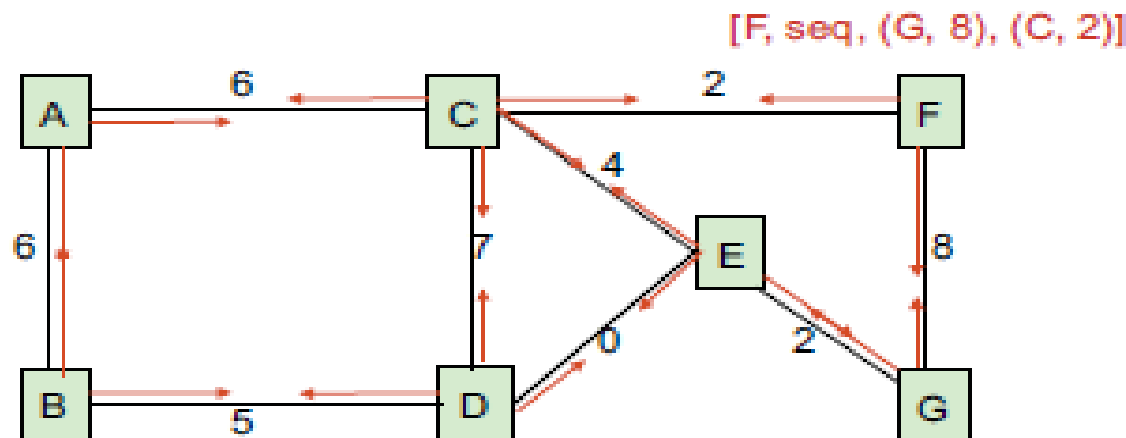
- **Periodično** formiranje LSA
- LSA se prenosi **po svakom linku** u oba smera
 - Ne trudite se da otkrijete odakle (sa kog linka) dolaze LSA paketi

LSA plavljenje

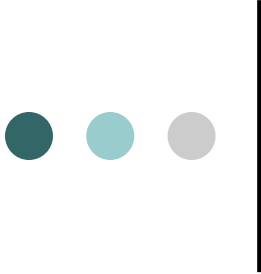


- Svaka stanica ***vršu difuziju primljenog LSA paketa samo jednom***
 - Koristiti seq# u proveru da li je nov LSA paket; sačuvati poslednji seq

LSA plavljenje



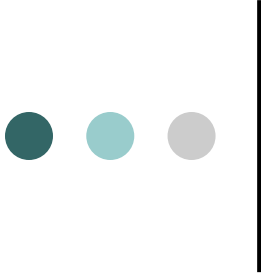
- **Višestruke mogućnosti** za svaki ruter **da primi** bilo koji **LSA paket**
 - **Potrebno vreme**: **broj linkova** potrebnih za prolaz kroz mrežu



Rutiranje na bazi stanja veze

○ *Integracija*

- Ako je **seq#** u primljenom LSA **> seq#** sačuvanog LSA za određenu izvorišnu tačku:
- **Ažurirati LSA** sa novim seq#, **listu suseda**
- **Ukloniti sačuvan LSA paket** ako je seq# previše zastareo
- **Rezultat**: Svaki ruter **formira trenutnu mapu mreže** – **link-state baza podataka**



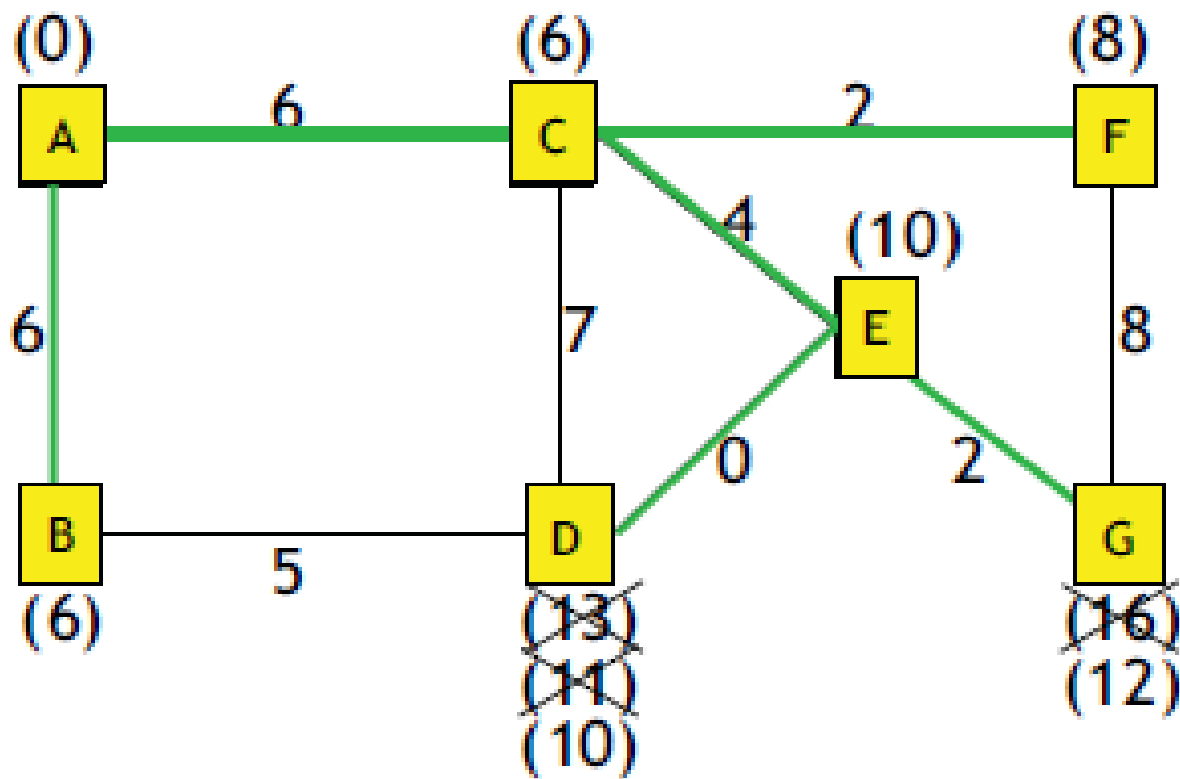
Rutiranje na bazi stanja veze

- **Formiranje ruting tabele**

- **Periodično** svaki ruter izvršava isti **algoritam najkraćeg puta** na bazi njegove trenutne mape – link-state baze podataka (npr. **Dijkstra algoritam**)
- **Uslovi korektnosti** ruting tabele:
 - **Proračun korektno implementiran** u svakoj tački i
 - Svaka tačka ima **identičnu mapu mreže**

Integracioni korak: Dijkstra algoritam (Primer)

- Predpostavimo da se žele **naći putanje od A do ostalih odredišta**

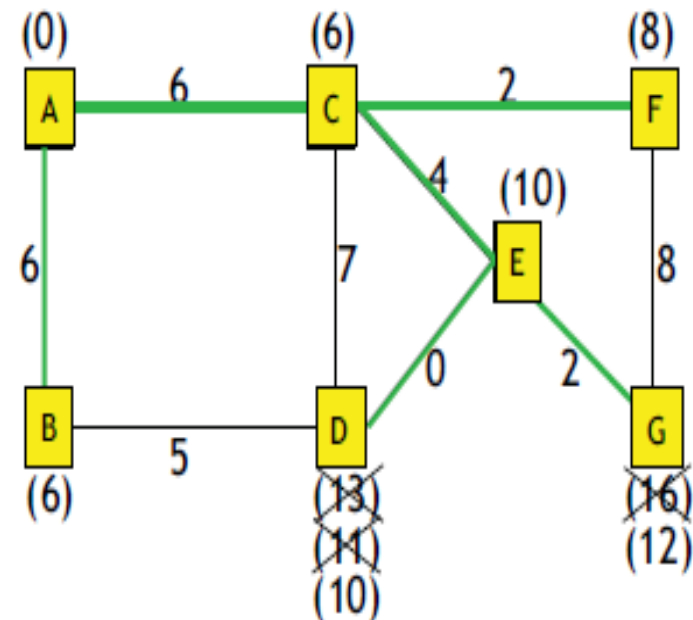


Dijkstra algoritam najkraćeg puta

○ Inicijalizacija:

- `nodeset` = [all nodes] = set of nodes we haven't processed
- `spcost` = {me:0, all other nodes: ∞ } # shortest path cost
- `routes` = {me:--, all other nodes: ?} # routing table

○ Osnovna petlja - dok je skup *nodeset* ne-prazan



- find `u`, the node in `nodeset` with smallest `spcost`
- remove `u` from `nodeset`
- for `v` in [`u`'s neighbors]:
 - `d = spcost(u) + cost(u,v)` # distance to `v` via `u`
 - if `d < spcost(v)`: # we found a shorter path!
 - `spcost[v] = d`
 - `routes[v] = routes[u]` (or if `u == me`, enter link from me to `v`)

Drugi primer

Naći najkraći put od A:

LSAs:

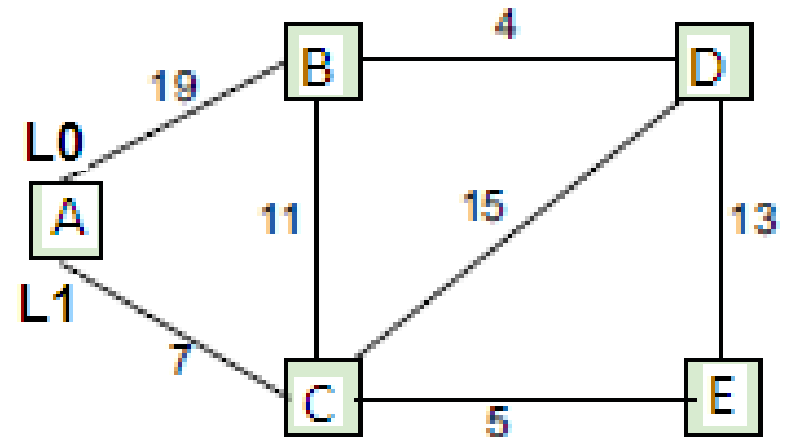
A: [(B, 19), (C, 7)]

B: [(A, 19), (C, 11), (D, 4)]

C: [(A, 7), (B, 11), (D, 15), (E, 5)]

D: [(B, 4), (C, 15), (E, 13)]

E: [(C, 5), (D, 13)]



- find u , the node in $nodeset$ with smallest $spcost$

- remove u from $nodeset$

- for v in $\{u's\ neighbors\}$:

• $d = spcost(u) + cost(u,v)$ # distance to v via u

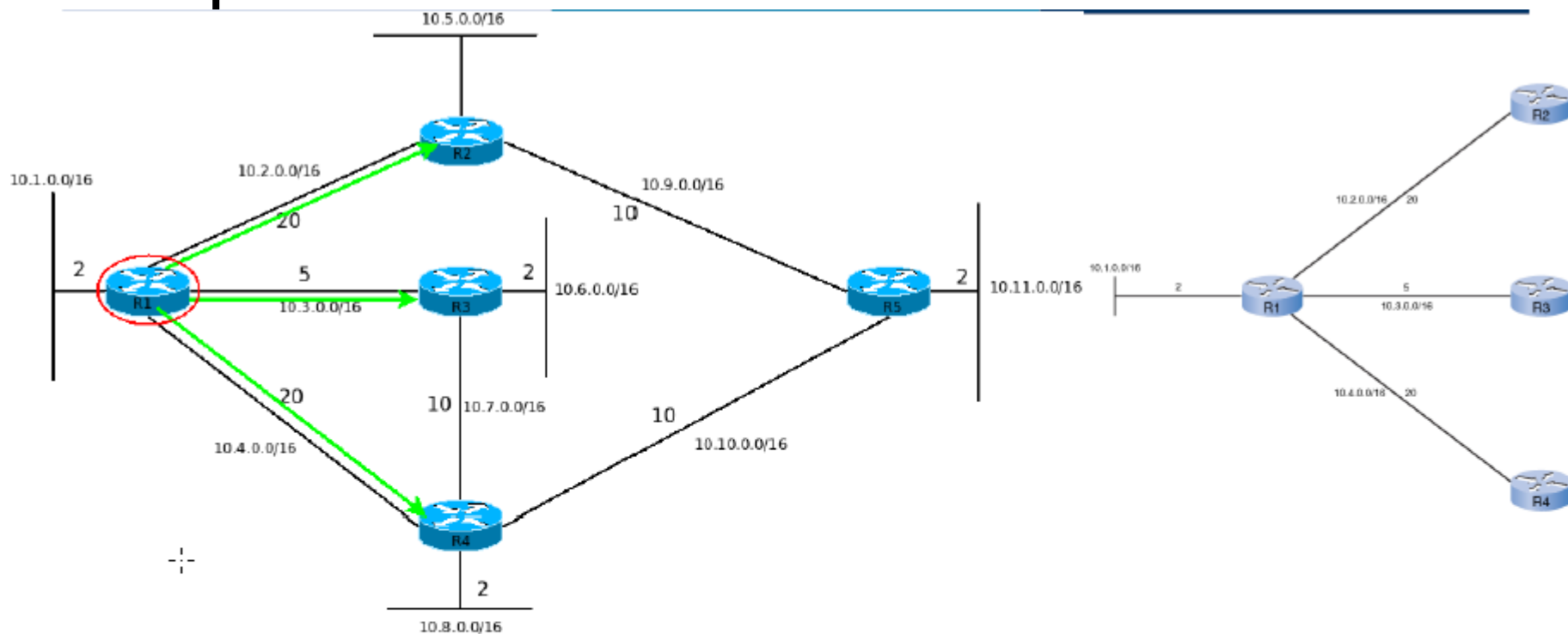
• if $d < spcost(v)$: # we found a shorter path!

- $spcost[v] = d$

- $routes[v] = routes[u]$ (or if $u == me$, enter link from me to v)

Step	u	Nodeset	$spcost$					route				
			A	B	C	D	E	A	B	C	D	E
0		[A,B,C,D,E]	0	∞	∞	∞	∞	--	?	?	?	?
1	A	[B,C,D,E]	0	19	7	∞	∞	--	L0	L1	?	?
2	C	[B,D,E]	0	18	7	22	12	--	L1	L1	L1	L1
3	E	[B,D]	0	18	7	22	12	--	L1	L1	L1	L1
4	B	[D]	0	18	7	22	12	--	L1	L1	L1	L1
5	D	[]	0	18	7	22	12	--	L1	L1	L1	L1

Još jedan primer, Početak R1



SPF стабло

R1, 0

кандидати

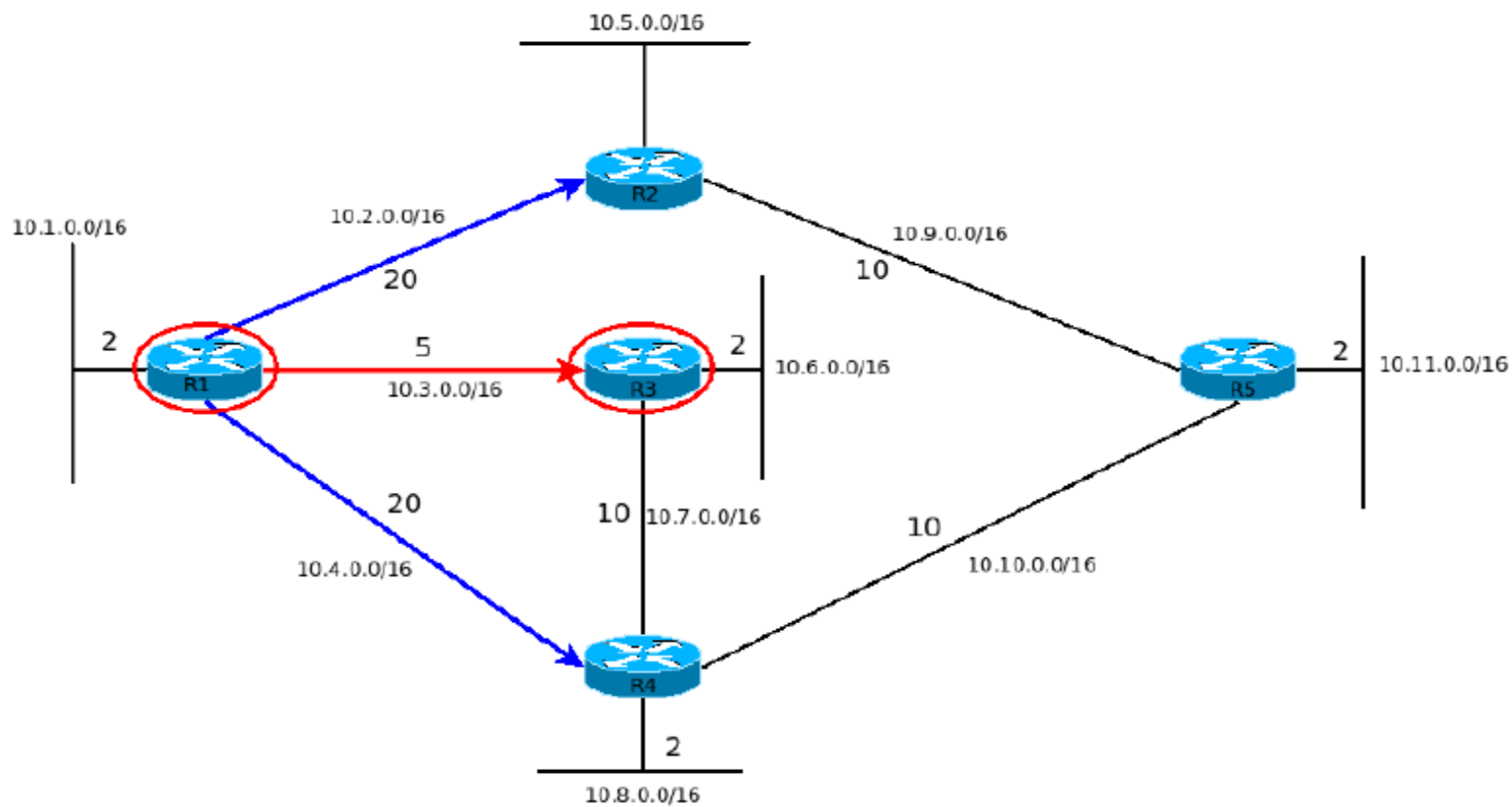
Суседи управо додатог чвора

R1->R2, 20 (додајемо, +0)

R1->R3, 5 (додајемо, +0)

R1->R4, 20 (додајемо, +0)

Kandidati od R1



SPF стабло

R1

кандидати

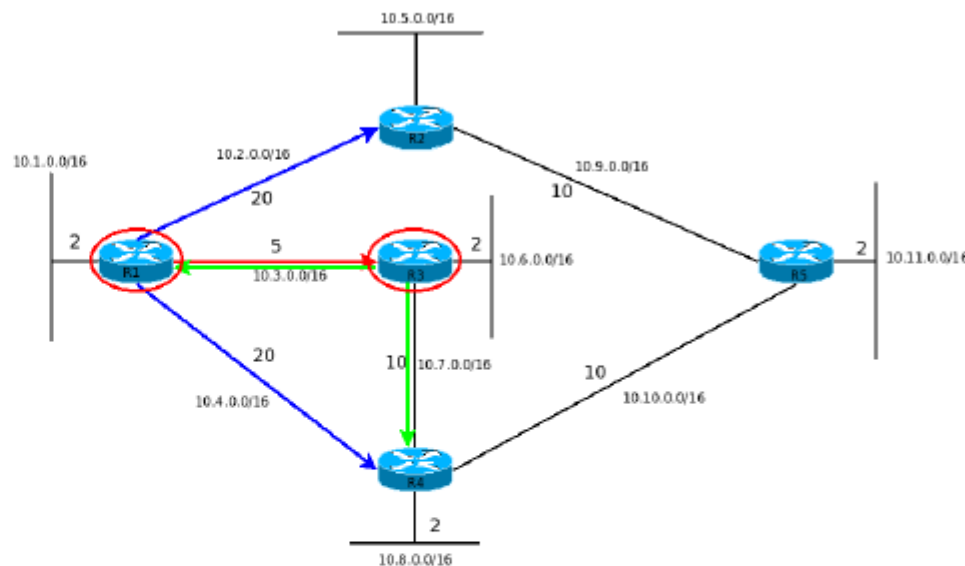
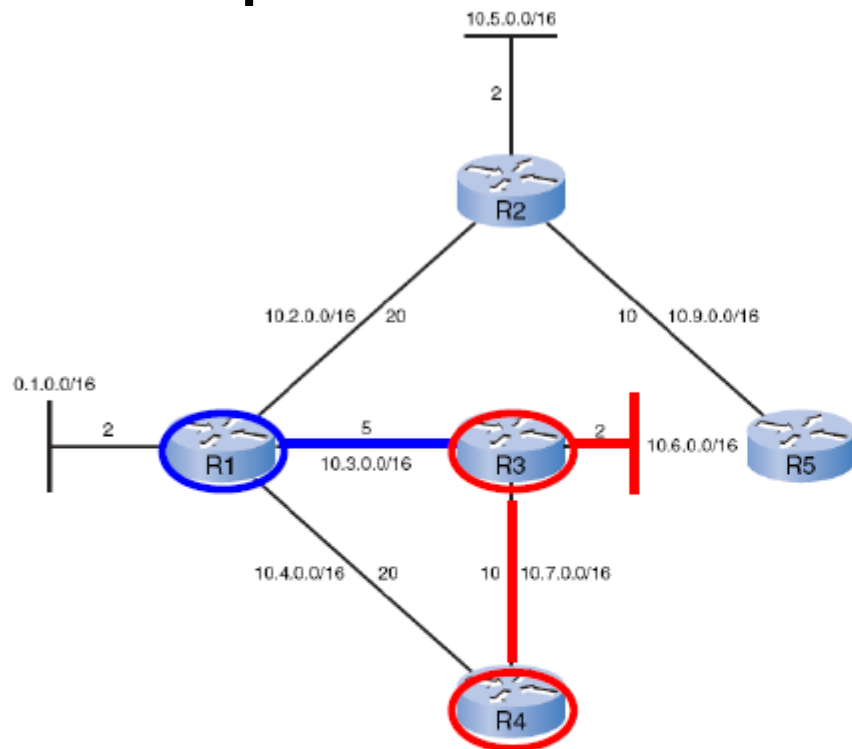
R1->R2, 20

R1->R3, 5 (SPF)

R1->R4, 20

Суседи управо додатог чвора

R1, R3, susedi R3



SPF стабло

R1, 0->
R3, 5

кандидати

R1->R2, 20

~~R1->R3, 5~~

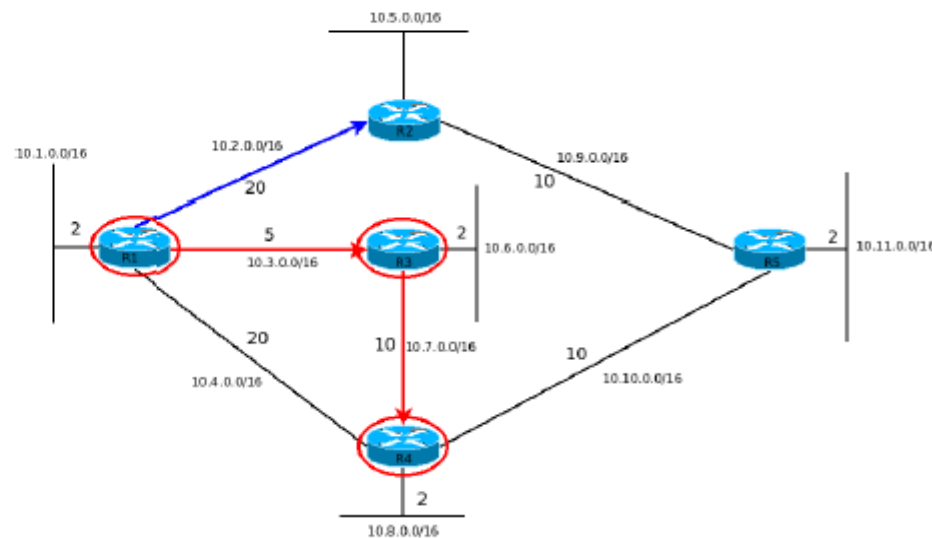
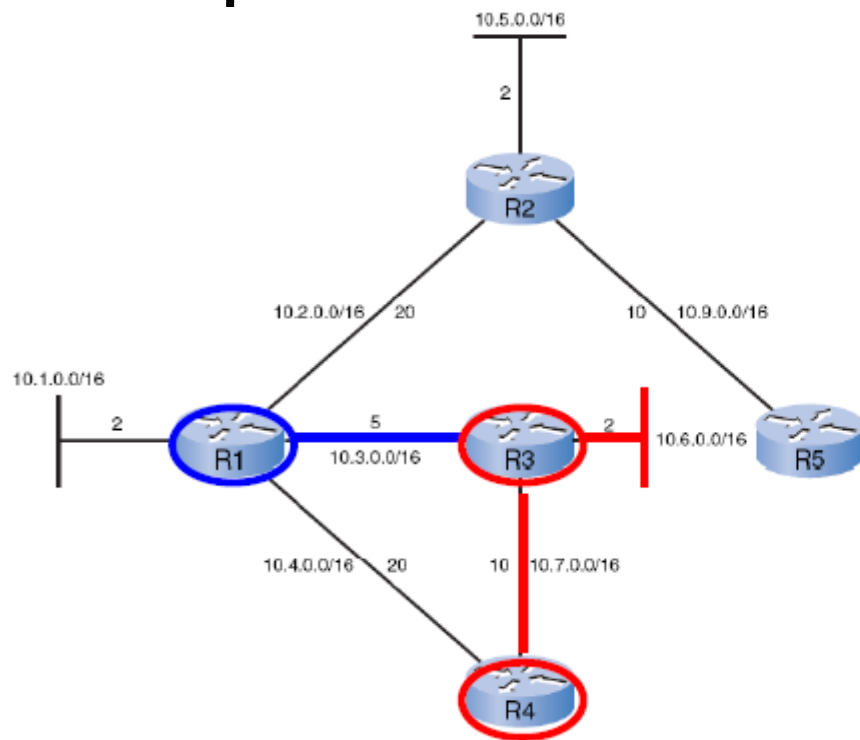
R1->R4, 20

Суседи управо додатог чвора

R3->R1, 5 (не додајемо, чвор постоји)

R3->R4, 10 (додајемо, $10 + 5 = 15 < 20$, уместо досадашњег кандидата за R4)

R1, R3, kandidati od R3



SPF стабло

R1->
R3, 5

кандидати

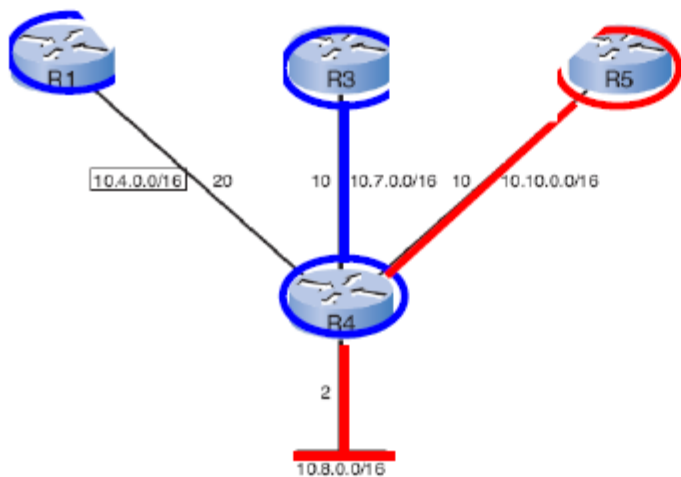
R1->R2, 20

~~R1->R3, 5~~

~~R1->R4, 20~~

R3->R4, 15

Суседи управо додатог чвора

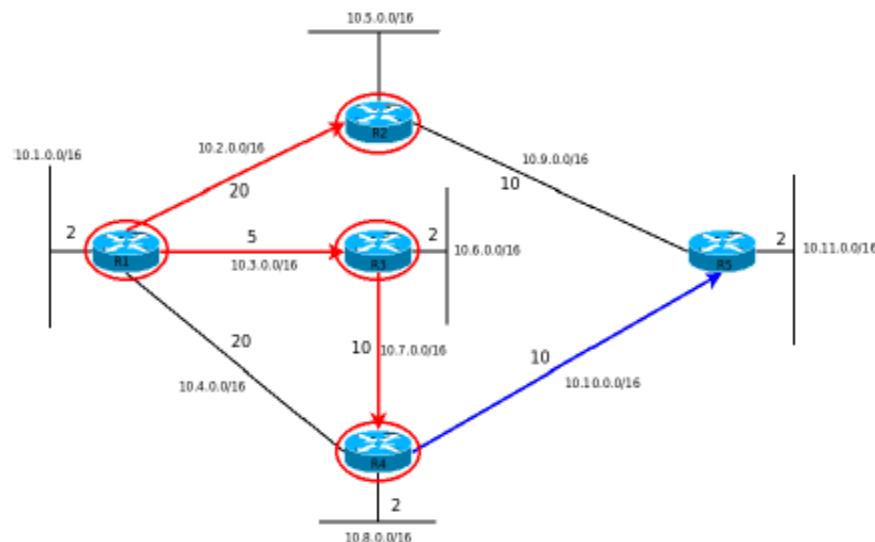
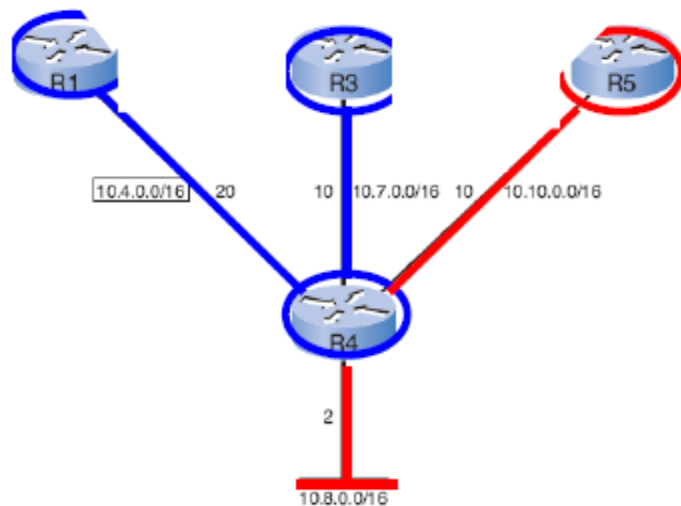


R1->
R3, 5 ->
R4, 15

~~R3 $\rightarrow$ R4, 15~~

R4->R5, 10 (додајемо, +15)

R1, R3, R4 kandidati od R4



кандидати

R1->R2, 20 (SPF)

R1->R3, 5

R1->R4, 20

R3->R4, 15

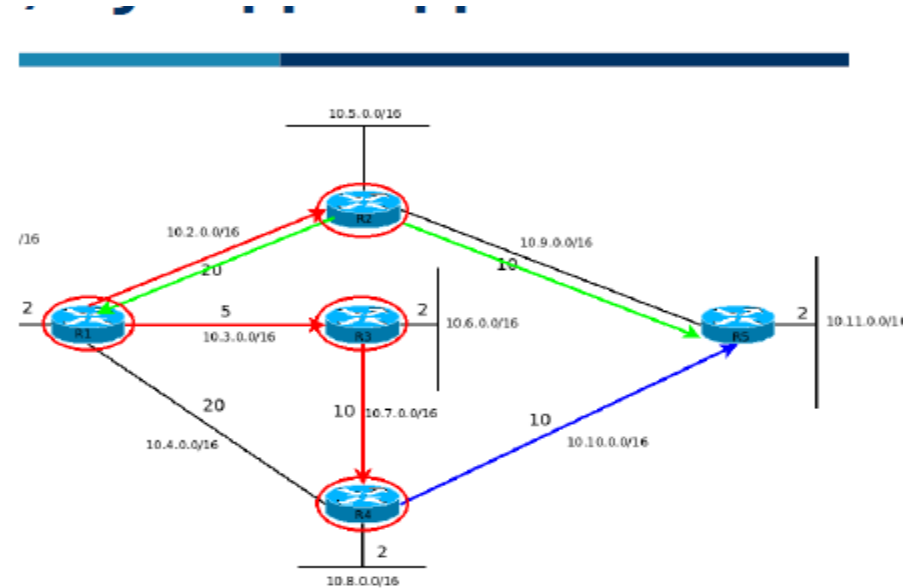
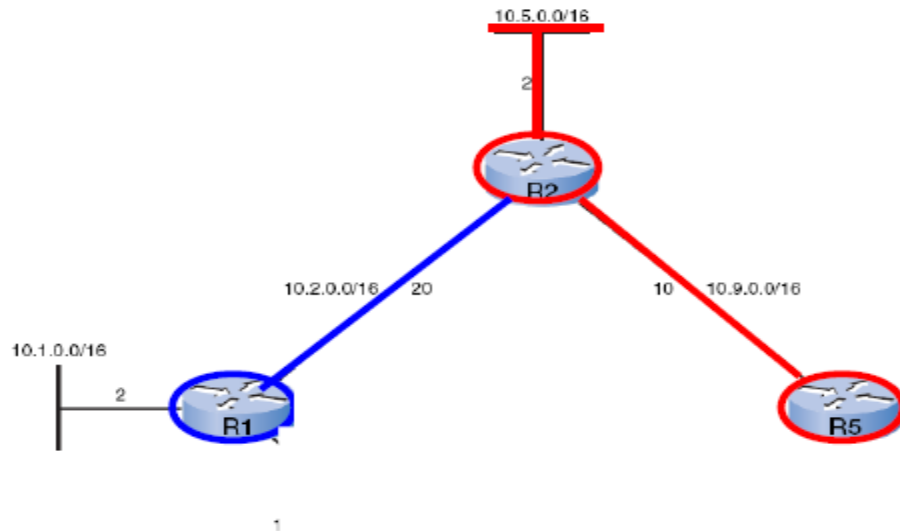
R4->R5, 25

SPF стабло

R1->
R3, 5 ->
R4, 15

Суседи управо додатог чвора

R1, R3, R4; R1, R2 susedi od R2



SPF стабло

R1->
R3, 5 ->
R4, 15

R1->
R2, 20

кандидати

R1->R2, 20

R1->R3, 5

R1->R4, 20

R3->R4, 15

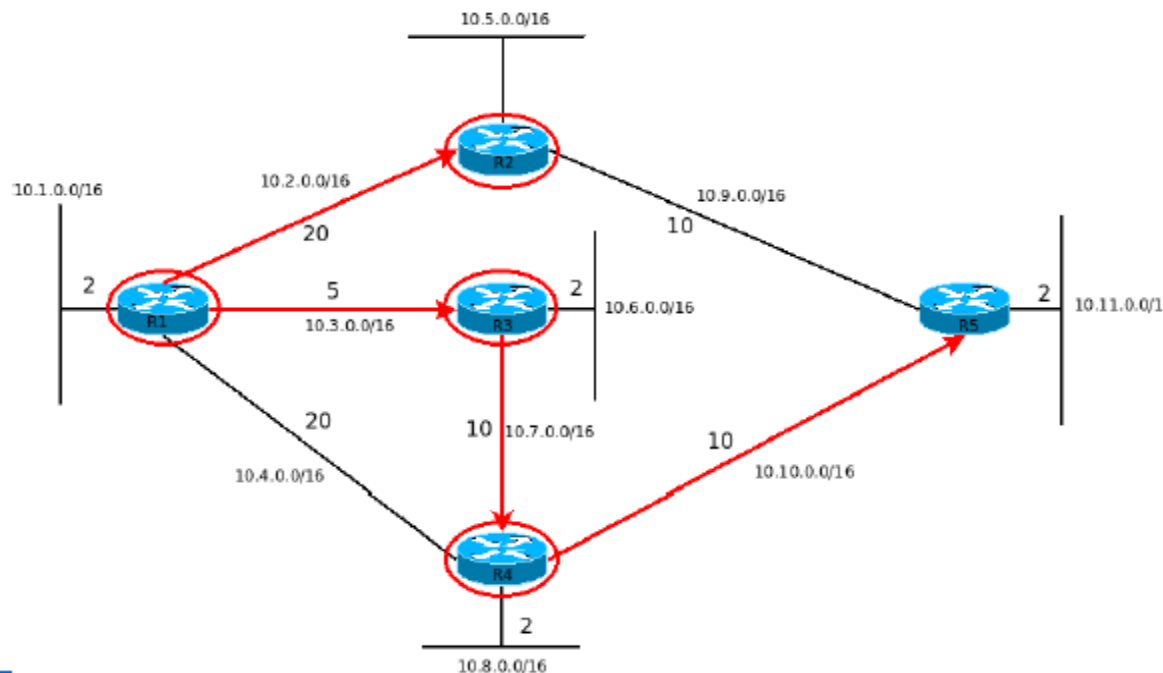
R4->R5, 25

Суседи управо додатог чвора

R2->R1, 20 (не додајемо, чвор постоји)

R2->R5, 10 (не додајемо: $10+20=30>25$, јер постоји краћи пут до R5, постоји бољи кандидат)

R1, R3, R4; R1, R2 kandidati
od R2



SPF стабло

R1->
R3, 5 ->
R4, 15

R1->
R2, 20

кандидати

R1->R2, 20

R1->R3, 5

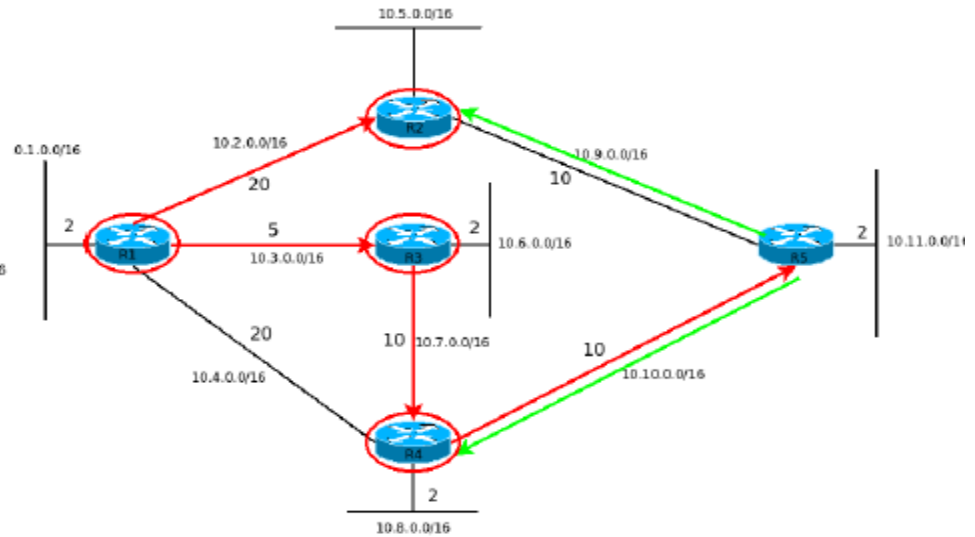
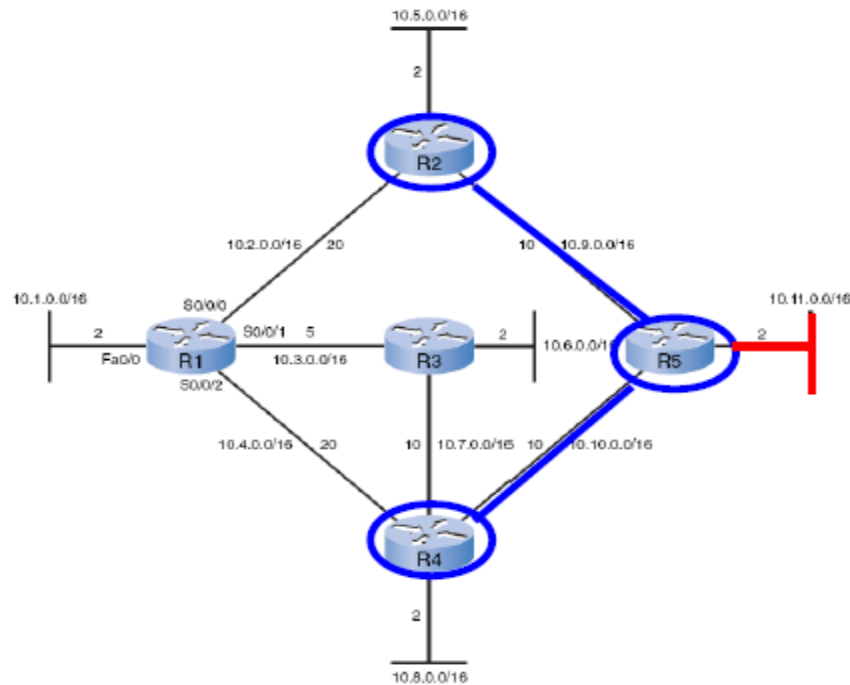
R1->R4, 20

R3->R4, 15

R4->R5, 25 (SPF)

Суседи управо додатог чвора

R1, R3, R4, R5; R1, R2 susedi od R5



SPF стабилно

R1->
R3, 5 ->
R4, 15 ->
R5, 25

R1->
R2, 20

кандидати

R1->R2, 20

R1->R3, 5

R1->R4, 20

R3->R4, 15

R4->R5, 25 (SPF)

Суседи управо додатог чвора

R5->R2, 10 (не додајемо, чвор постоји)

R5->R4, 10 (не додајемо, чвор постоји)