

Rețele de calculatoare

#8

Cerințele de proiectare ale
nivelului rețea

Algoritmi de dirijare în rețea

Adrian Runceanu

<https://www.runceanu.ro/adrian/>

Curs 8

- 1. Cerințele de proiectare ale nivelului rețea**
- 2. Algoritmi de dirijare în rețea**

Nivelul rețea

1. Cerințele de proiectare ale nivelului rețea

1.1 Comutare de pachete de tip Memorează-și-Retransmite(Store-and-Forward)

1.2 Servicii furnizate nivelului transport

1.3 Implementarea serviciului neorientat pe conexiune

1.4 Implementarea serviciilor orientate pe conexiune

1.5 Comparatie între subrețele cu circuite virtuale și subrețele datagramă

1. Cerințele de proiectare ale nivelului rețea

1.1 Comutare de pachete de tip Memorează-și-Retransmite (Store-and-Forward)

Contextul în care operează protocoalele de la nivelul rețea:

- Componentele majore ale sistemului sunt:
 - *echipamentul companiei de telecomunicații* (routere conectate prin linii de transmisie), prezentat în interiorul ovalului umbrat
 - *echipamentul clientului*, prezentat în afara ovalului
- Gazda H1 este conectată direct la unul dintre routerele companiei de telecomunicații A, printr-o linie închiriată.
- În contrast, gazda H2 este într-o rețea LAN cu un router, F, deținut și operat de către client.
- Acest context este prezentat în fig. 1.

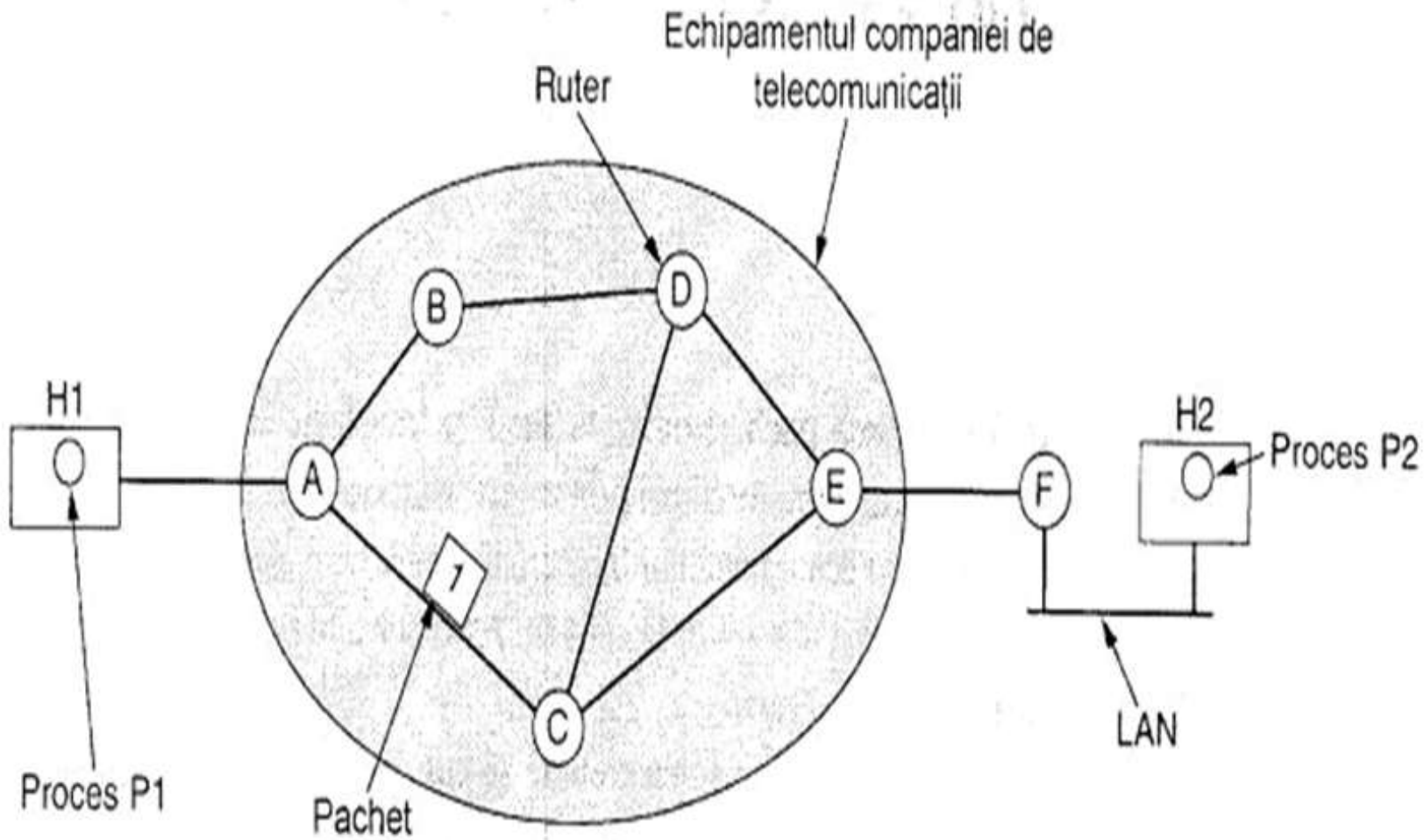


Fig. 1. Cadrul protocoalelor nivelului rețea.

1.1 Comutare de pachete de tip Memorează-și-Retransmite(Store-and-Forward)

Acest echipament este folosit după cum urmează:

- O gazdă care are de transmis un pachet îl transmite celui mai apropiat router, fie în aceeași rețea LAN, fie printr-o legătură punct la punct cu compania de telecomunicații.
- Pachetul este memorat acolo până ajunge integral, astfel încât să poată fi verificată suma de control.
- Apoi este trimis mai departe către următorul router de pe traseu, până ajunge la destinație, unde este livrat.
- Acest mecanism reprezintă comutarea de pachete de tip **memorează-și-retransmite**.

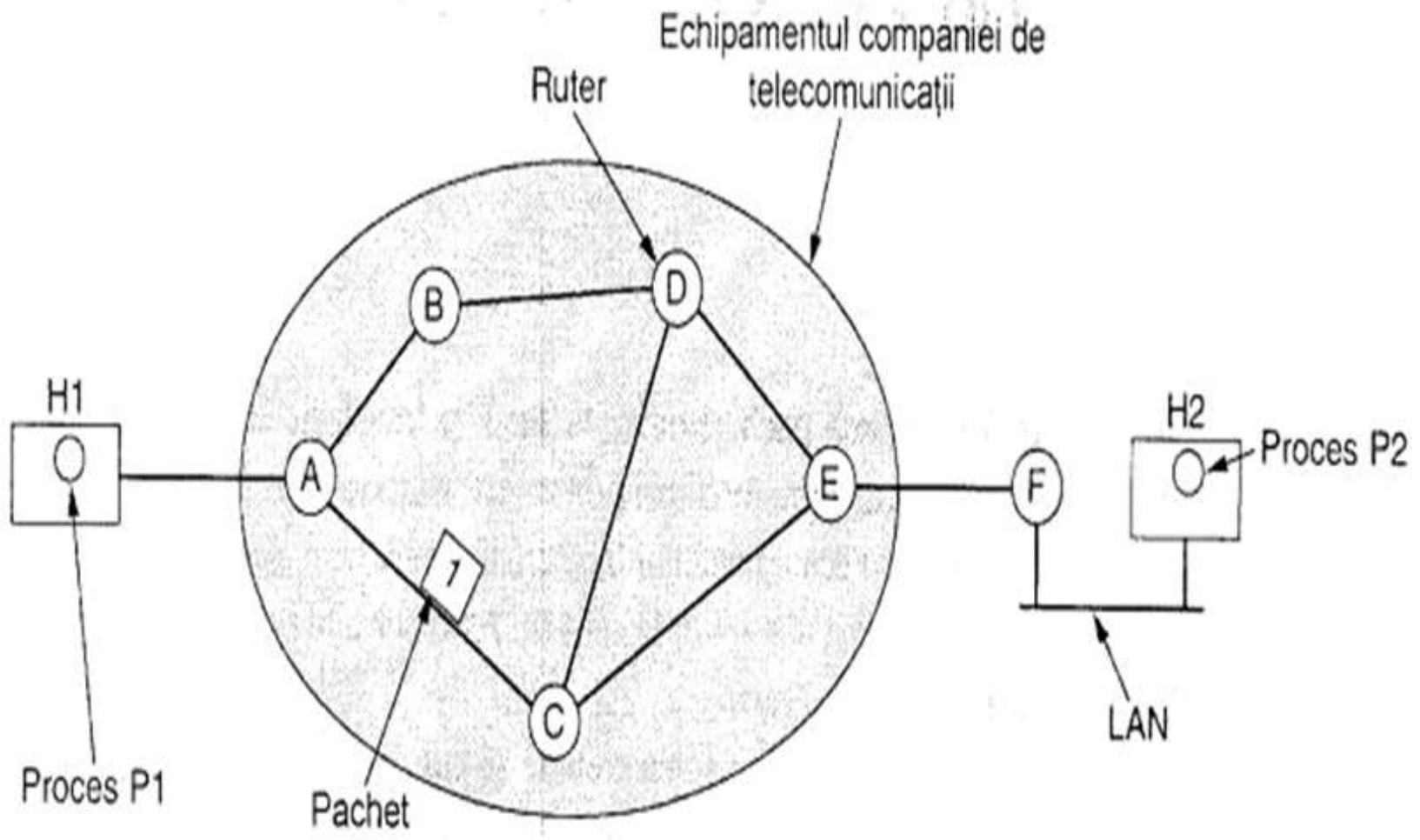


Fig. 1. Cadrul protocoalelor nivelului rețea

Nivelul rețea

1. Cerințele de proiectare ale nivelului rețea

1.1 Comutare de pachete de tip Memorează-și-Retransmite(Store-and-Forward)

1.2 Servicii furnizate nivelului transport

1.3 Implementarea serviciului neorientat pe conexiune

1.4 Implementarea serviciilor orientate pe conexiune

1.5 Comparatie între subrețele cu circuite virtuale și subrețele datagramă

1.2 Servicii furnizate nivelului transport

Serviciile nivelului rețea au fost proiectate având în vedere următoarele scopuri:

1. Serviciile trebuie să fie *independente de tehnologia routerului*.
2. *Nivelul transport* trebuie să fie *independent de numărul, tipul și topologia routerelor existente*.
3. *Adresele de rețea* disponibile la nivelul transport *trebuie să folosească o schemă de numerotare uniformă*, chiar în cadrul rețelelor **LAN** și **WAN**.

1.2 Servicii furnizate nivelului transport

- Obiectivele fiind stabilite, proiectantul nivelului rețea are o mare libertate în a scrie specificațiile detaliate ale serviciilor oferite nivelului transport.
- Această libertate degeneratează adesea pareri diferite între două variante diferite.
- Problema centrală a discuției este dacă **nivelul rețea** trebuie să furnizeze:
 1. *servicii orientate pe conexiune*
 2. *servicii neorientate pe conexiune*

1.2 Servicii furnizate nivelului transport

1. O primă variantă (reprezentată de **comunitatea Internet**) este aceea prin care se afirmă că *scopul routerului este de a transfera pachete și nimic mai mult.*
 - În aceasta variantă subrețeaua este inerent nesigură, indiferent cum ar fi proiectată.
 - De aceea *calculatoarele gazdă trebuie să accepte faptul că rețeaua este nesigură și să facă controlul erorilor (adica, detecția și corecția erorii) și controlul fluxului ele însele.*

1.2 Servicii furnizate nivelului transport

- Acest punct de vedere duce rapid la concluzia că *serviciul rețea trebuie să fie neorientat pe conexiune*, cu două primitive **SEND PACKET** și **RECEIVE PACKET** și cu foarte puțin în plus.
- În particular, nu trebuie făcută nici o operație pentru controlul ordinii sau fluxului pachetelor pentru că oricum calculatorul gazdă va face acest lucru, și, de obicei, dublarea acestor operații aduce un câștig nesemnificativ.
- În continuare, *fiecare pachet va trebui să poarte întreaga adresă de destinație*, pentru că fiecare pachet este independent de pachetele predecesoare, dacă acestea există.

1.2 Servicii furnizate nivelului transport

2. Cea de a doua variantă (reprezentată de **companiile de telefoane**) afirmă că *subrețeaua trebuie să asigure un serviciu orientat pe conexiune sigur.*

- Ei susțin că peste 100 de ani de experiență cu sistemul telefonic mondial reprezintă un ghid excelent.
- În această perspectivă, calitatea serviciului este elementul dominant, și într-o subrețea fără conexiuni, calitatea serviciului este dificil de obținut, în special pentru trafic în timp real cum ar fi voce și imagine.

1.2 Servicii furnizate nivelului transport

Aceste două variante sunt cel mai bine exemplificate de rețeaua Internet și rețelele ATM.

1. Rețeaua Internet oferă un *serviciu la nivelul rețea neorientat pe conexiune*
2. Rețelele ATM oferă un *serviciu la nivelul rețea orientat pe conexiune*

Totuși, este interesant de notat că, cu cât garantarea calității serviciului devine din ce în ce mai importantă, Internet-ul evoluează.

Nivelul rețea

1. Cerințele de proiectare ale nivelului rețea

1.1 Comutare de pachete de tip Memorează-și-Retransmite(Store-and-Forward)

1.2 Servicii furnizate nivelului transport

1.3 Implementarea serviciului neorientat pe conexiune

1.4 Implementarea serviciilor orientate pe conexiune

1.5 Comparatie între subrețele cu circuite virtuale și subrețele datagramă

1.3 Implementarea serviciului neorientat pe conexiune

- După ce am văzut cele două clase de servicii pe care nivelul rețea le furnizează utilizatorilor săi, este momentul să vedem funcționarea internă a acestui nivel.
- Sunt posibile două organizări diferite, în funcție de tipul serviciului oferit:
 1. serviciul neorientat pe conexiune
 2. serviciul orientat conexiune

1.3 Implementarea serviciului neorientat pe conexiune

1. Dacă este oferit un **serviciu neorientat pe conexiune**, atunci *pachetele sunt trimise în subrețea individual și dirijate independent de celelalte*.

- Nu este necesară nici o inițializare prealabilă.
- În acest context, pachetele sunt numite frecvent **datagramme** (datagrams) (prin analogie cu telegramele), iar subrețeaua este numită **subrețea datagramă** (datagram subnet).

1.3 Implementarea serviciului neorientat pe conexiune

2. Dacă este folosit **serviciul orientat conexiune**, atunci, *înainte de a trimite pachete de date, trebuie stabilită o cale de la routerul sursă la routerul destinație.*

➤ Această conexiune este numită **VC - circuit virtual (virtual circuit)**, prin analogie cu circuitele fizice care se stabilesc în sistemul telefonic, iar subrețeaua este numită **subrețea cu circuite virtuale (virtual-circuit subnet)**.

1.3 Implementarea serviciului neorientat pe conexiune

Să vedem cum funcționează o **subrețea datagramă**:

- Să presupunem că procesul P1 din fig. 2 are un mesaj lung pentru procesul P2.
- El transmite mesajul nivelului transport, cu instrucțiunile de livrare către procesul P2 aflat pe calculatorul gazdă H2.
- *Codul nivelului transport rulează pe calculatorul gazdă H1, de obicei în cadrul sistemului de operare.*
- Acesta inserează la începutul mesajului un *antet corespunzător nivelului transport* și transferă rezultatul nivelului rețea, probabil o altă procedură din cadrul sistemului de operare.

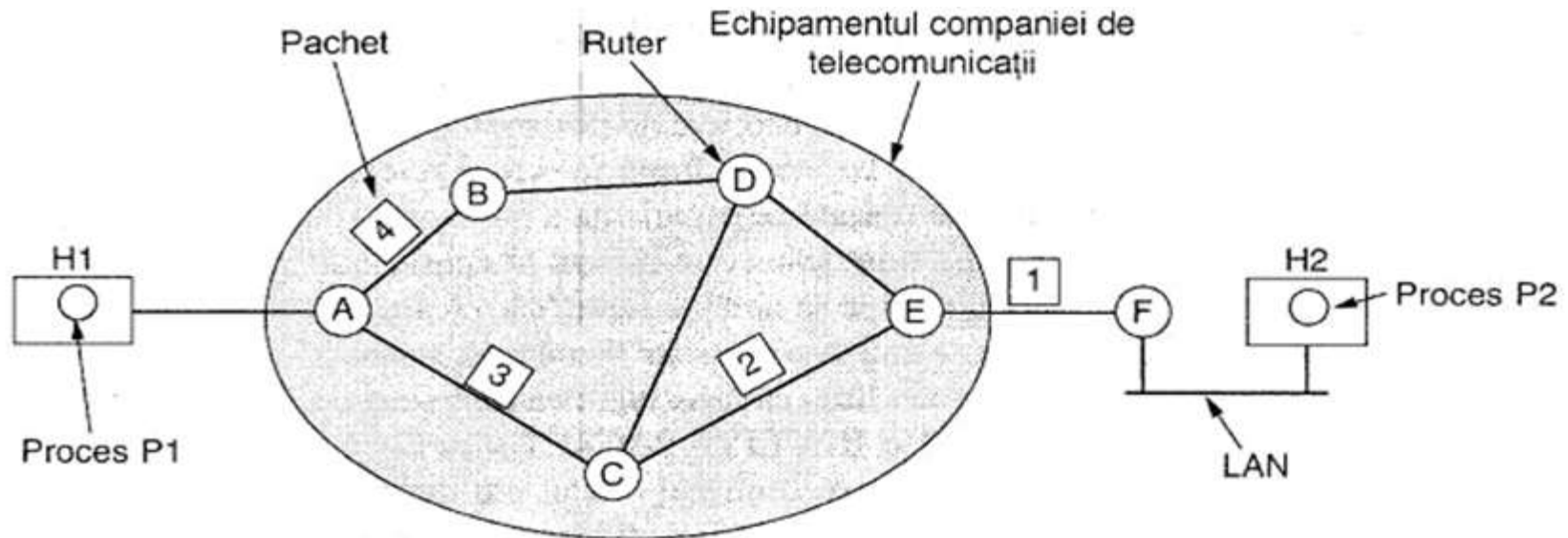


Tabela ruterului A

inițial	ulterior
A -	A -
B B	B B
C C	C C
D B	D B
E C	E B
F C	F B

Dest. Rută

Tabela ruterului C

A A
B A
C -
D D
E E
F E

Tabela ruterului E

A C
B D
C C
D D
E -
F F

Fig. 2. Dirijarea într-o subrețea datagramă

1.3 Implementarea serviciului neorientat pe conexiune

- Să presupunem că mesajul este de patru ori mai lung decât dimensiunea maximă a unui pachet, așa că nivelul rețea trebuie să îl spargă în patru pachete, 1, 2, 3, și 4 și să le trimită pe fiecare în parte routerului F, folosind un **protocol punct-la-punct**, de exemplu, **PPP (point-to-point protocol)**.
- Din acest punct controlul este preluat de compania de telecomunicații.
- Fiecare router are o tabelă internă care îi spune unde să trimită pachete pentru fiecare destinație posibilă.

1.3 Implementarea serviciului neorientat pe conexiune

- *Fiecare intrare în tabelă este o pereche compusă din destinație și linia de ieșire folosită pentru acea destinație.*
- Pot fi folosite doar linii conectate direct.
- De exemplu, în fig. 2, A are doar două linii de ieșire - către B și C - astfel că fiecare pachet ce vine trebuie trimis către unul dintre aceste routere, chiar dacă ultima destinație este alt router.
- Tabela de rutare inițială a lui A este prezentată în figură sub eticheta „inițial”.

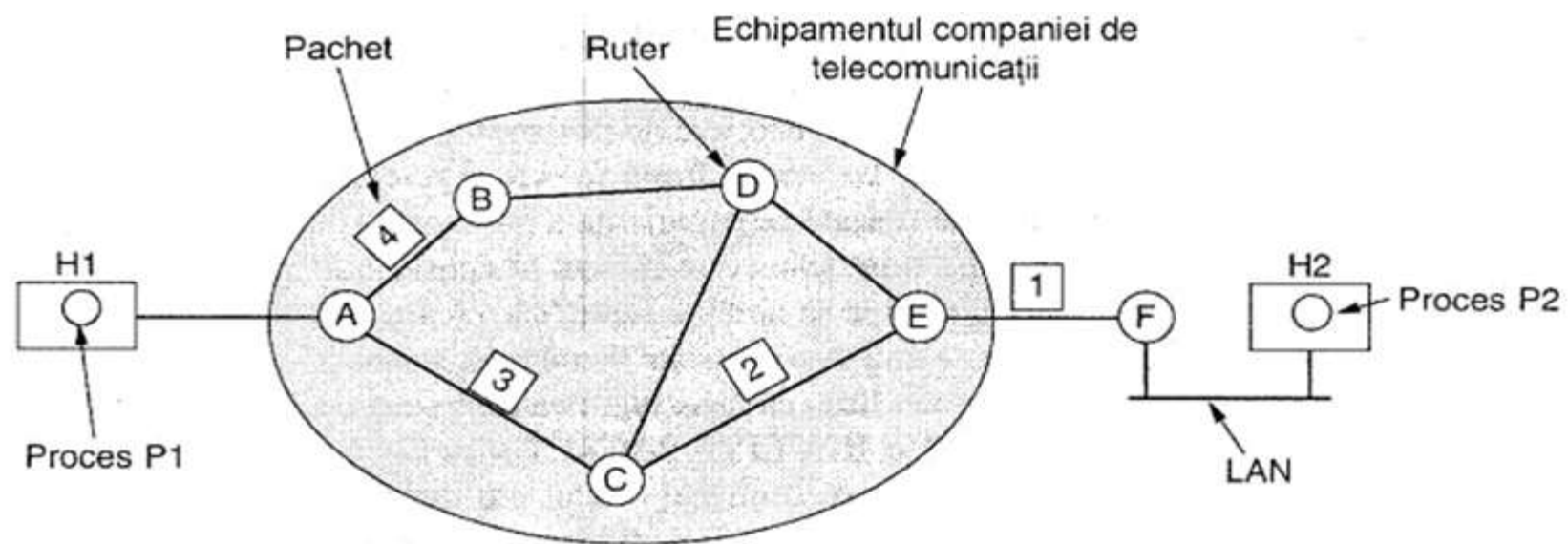


Tabela ruterului A

inițial	ulterior
A -	A -
B B	B B
C C	C C
D B	D B
E C	E B
F C	F B

Tabela ruterului C

A A
B A
C -
D D
E E
F E

Tabela ruterului E

A C
B D
C C
D D
E -
F F

Dest. Rută

Fig. 2. Dirijarea într-o subrețea datagramă

1.3 Implementarea serviciului neorientat pe conexiune

- Cum au ajuns la A, pachetele 1, 2 și 3 au fost memorate pentru scurt timp (pentru verificarea sumei de control).
- Apoi fiecare a fost trimis mai departe către C conform tabelii lui A.
- Pachetul 1 a fost apoi trimis mai departe către E și apoi către F.
- Când a ajuns la F, a fost încapsulat într-un cadru al nivelului legătură de date și trimis către calculatorul gazdă H2 prin rețeaua LAN.

1.3 Implementarea serviciului neorientat pe conexiune

- Totuși, ceva diferit s-a întâmplat cu pachetul 4.
- Când a ajuns la A, a fost trimis către routerul B, chiar dacă și el este destinat tot lui F.
- Dintr-un motiv oarecare, A a decis să trimită pachetul 4 pe o rută diferită de cea urmată de primele trei.
- Poate că a aflat despre o congestie undeva pe calea ACE și și-a actualizat tabela de rutare, așa cum apare sub eticheta „ulterior”.
- *Algoritmul ce administrează tabelele și ia deciziile de rutare* se numește **algoritm de rutare** (routing algorithm).

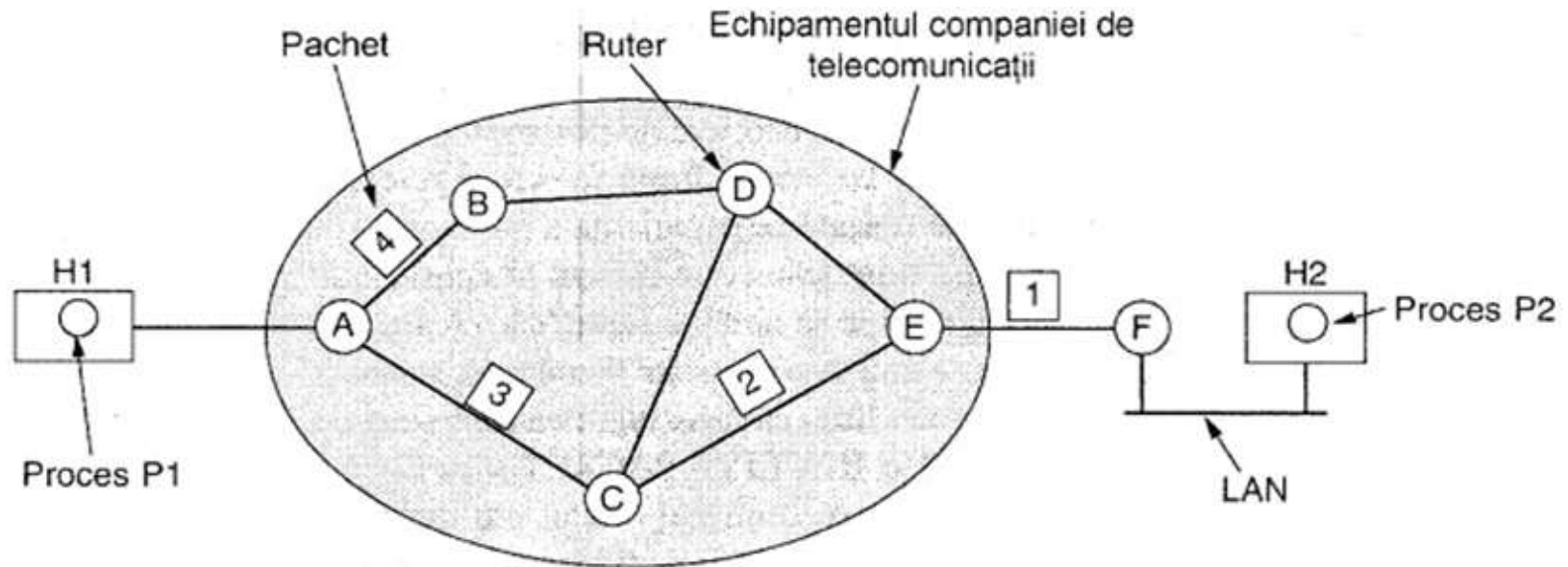


Tabela ruterului A

inițial	ulterior
A -	A -
B B	B B
C C	C C
D B	D B
E C	E B
F C	F B

Tabela ruterului C

A A
B A
C -
D D
E E
F E

Tabela ruterului E

A C
B D
C C
D D
E -
F F

Dest. Rută

Fig. 2. Dirijarea într-o subrețea datagramă

Nivelul rețea

1. Cerințele de proiectare ale nivelului rețea

1.1 Comutare de pachete de tip Memorează-și-Retransmite(Store-and-Forward)

1.2 Servicii furnizate nivelului transport

1.3 Implementarea serviciului neorientat pe conexiune

1.4 Implementarea serviciilor orientate pe conexiune

1.5 Comparatie între subrețele cu circuite virtuale și subrețele datagramă

1.4 Implementarea serviciilor orientate pe conexiune

Pentru serviciile orientate conexiune, avem nevoie de o subrețea cu circuite virtuale

- Ideea care se stă la baza circuitelor virtuale este evitarea alegerii unei noi căi (rute) pentru fiecare pachet trimis, ca în fig. 2.
- În schimb, atunci **când se stabilește o conexiune**, se alege **o cale între mașina sursă și mașina destinație**, ca parte componentă a inițializării conexiunii și **aceasta este memorată în tabelele routerelor**.
- Acea cale este folosită pentru tot traficul de pe conexiune, exact în același mod în care funcționează sistemul telefonic.
- Atunci când conexiunea este eliberată, este închis și circuitul virtual.
- În cazul serviciilor orientate conexiune, *fiecare pachet poartă un identificador care spune cărui circuit virtual îi aparține.*

1.4 Implementarea serviciilor orientate pe conexiune

- De exemplu, să considerăm situația din fig. 3.
- Aici calculatorul gazdă H1 a stabilit conexiunea 1 cu calculatorul gazdă H2.
- Aceasta este memorată ca prima intrare în fiecare tabelă de rutare.
- Prima linie a tabelului A spune că dacă un pachet purtând identificatorul de conexiune 1 vine de la H1, atunci trebuie trimis către routerul C, dându-i-se identificatorul de conexiune 1.
- Similar, prima intrare a lui C dirijează pachetul către E, tot cu identificatorul de conexiune 1.

- Acum să vedem ce se întâmplă dacă H3 vrea, de asemenea, să stabilească o conexiune cu H2.
- Alege identificatorul de conexiune 1 (deoarece inițializează conexiunea și aceasta este singura conexiune) și indică subrețelei să stabilească circuitul virtual.
- Aceasta conduce la a doua linie din tabele.
- Observați că apare un conflict deoarece deși A poate distinge ușor pachetele conexiunii 1 de la H1 de pachetele conexiunii 1 de la H3, C nu poate face asta.
- Din acest motiv, A asociază un identificator de conexiune diferit pentru traficul de ieșire al celei de a doua conexiuni.
- Pentru *evitarea conflictelor de acest gen routerele trebuie să poată înlocui identificatorii de conexiune în pachetele care pleacă*.
- În unele contexte, aceasta se numește **comutarea etichetelor (label switching)**.

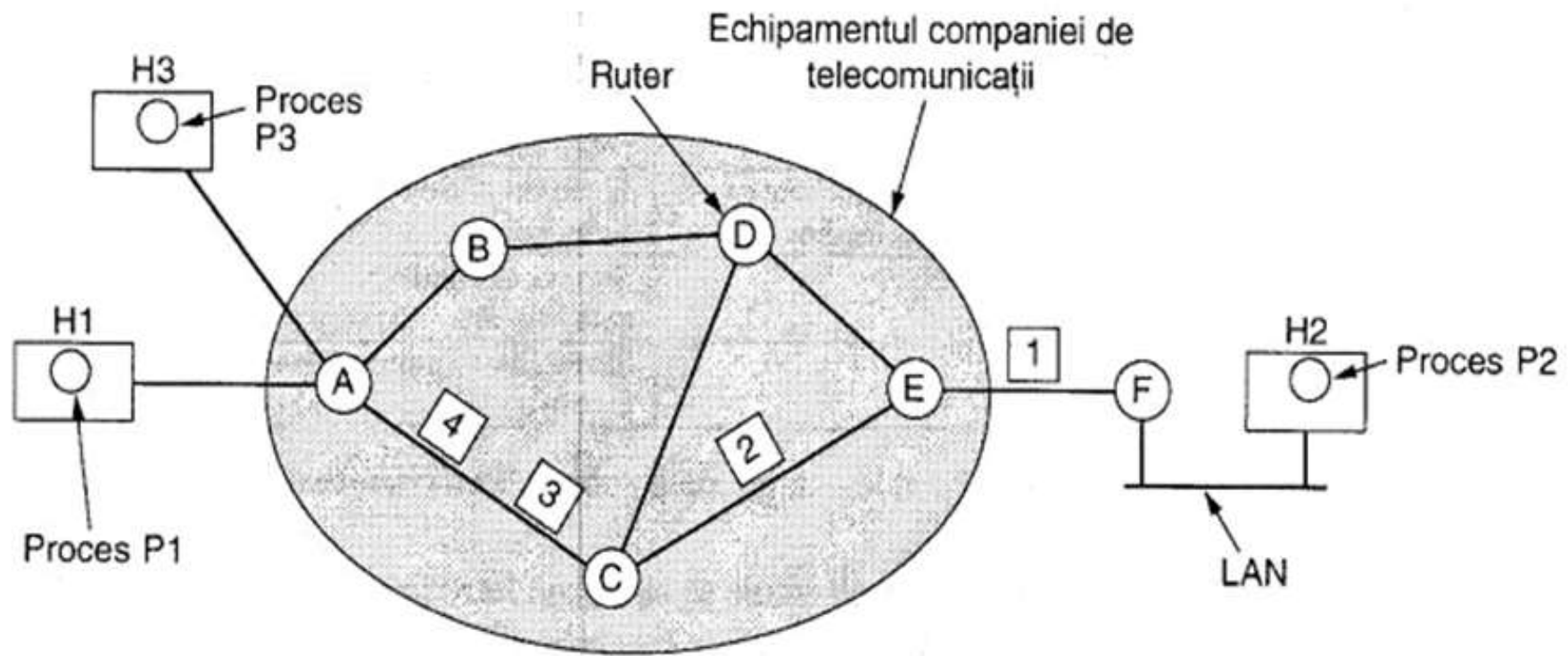


Tabela ruterului A

H1	1	C	1
H3	1	C	2

In Out

Tabela ruterului C

A	1	E	1
A	2	E	2

Tabela ruterului E

C	1	F	1
C	2	F	2

Fig. 3. Dirijare în cadrul unei subrețele cu circuite virtuale

Nivelul rețea

1. Cerințele de proiectare ale nivelului rețea

1.1 Comutare de pachete de tip Memorează-și-Retransmite(Store-and-Forward)

1.2 Servicii furnizate nivelului transport

1.3 Implementarea serviciului neorientat pe conexiune

1.4 Implementarea serviciilor orientate pe conexiune

1.5 Comparație între subrețele cu circuite virtuale și subrețele datagramă

1.5 Comparație între subrețele cu circuite virtuale și subrețele datagramă

- Atât circuitele virtuale cât și datagramele au suporteri și oponenți.
- Vom încerca acum să rezumăm argumentele ambelor variante.
- Principalele aspecte sunt prezentate în figura următoare:

Problemă	Subrețea datagramă	Subrețea cu circuite virtuale (CV)
Stabilirea circuitului	Nu este necesară	Obligatorie
Adresare	Fiecare pachet conține adresa completă pentru sursă și destinație	Fiecare pachet conține un număr mic de CV
Informații de stare	Routerele nu păstrează informații despre conexiuni	Fiecare CV necesită spațiu pentru tabela routerului per conexiune
Dirijare	Fiecare pachet este dirijat independent	Calea este stabilită la inițierea CV; toate pachetele o urmează
Efectul defectării routerului	Nici unul, cu excepția pachetelor pierdute în timpul defectării	Toate circuitele virtuale care trec prin routerul defect sunt terminate
Calitatea serviciului	Dificil	Simplu, dacă pentru fiecare CV pot fi alocate în avans suficiente resurse
Controlul congestiei	Dificil	Simplu, dacă pentru fiecare CV pot fi alocate în avans suficiente resurse

1.5 Comparație între subrețele cu circuite virtuale și subrețele datagramă

- În interiorul subrețelei există situații în care trebuie să se aleagă între facilități antagoniste specifice fie *circuitelor virtuale*, fie *datagramelor*.

1. Un astfel de compromis este acela între spațiul de memorie al routerului și lățimea de bandă.

- *Circuitele virtuale permit pachetelor să conțină numere de circuite în locul unor adrese complete.*
- Dacă pachetul tinde să fie foarte mic, atunci existența unei adrese complete în fiecare pachet poate reprezenta o supraîncărcare (overhead) importantă și deci o irosire a lățimii de bandă.
- Prețul plătit pentru folosirea internă a circuitelor virtuale este spațiul necesar păstrării tabelului în router.
- Soluția mai ieftină este determinată de *raportul între costul circuitelor de comunicație și cel al memoriei routerului.*

1.5 Comparație între subrețele cu circuite virtuale și subrețele datagramă

2. Alt compromis este cel între *timpul necesar stabilirii circuitului și timpul de analiză a adresei*.

- Folosirea circuitelor virtuale presupune *existența unei faze inițiale de stabilire a căii*, care cere timp și consumă resurse.
- Oricum, este ușor să ne imaginăm ce se întâmplă cu un pachet de date într-o *subrețea bazată pe circuite virtuale*:
 - *routerul folosește numărul circuitului ca un index într-o tabelă pentru a afla unde merge pachetul*
- Într-o rețea bazată pe *datagrame*, pentru a găsi intrarea corespunzătoare destinației *se folosește o procedură de căutare mult mai complicată*.

3. O altă problemă este cea a ***dimensiunii spațiului necesar pentru tabela din memoria router-ului.***

- O subrețea datagramă necesită o intrare pentru fiecare destinație posibilă, în timp ce o subrețea cu circuite virtuale necesită o intrare pentru fiecare circuit virtual.
- Totuși, acest avantaj este relativ iluzoriu deoarece și pachetele de inițializare a conexiunii trebuie rutate, iar ele folosesc adresele destinație, la fel ca și datagramele.

1.5 Comparație între subrețele cu circuite virtuale și subrețele datagramă

- Circuitele virtuale au unele **avantaje** în *garantarea calității serviciului și evitarea congestiunii subrețelei*, deoarece resursele (de exemplu zone tampon, lățime de bandă și cicluri CPU) pot fi rezervate în avans, atunci când se stabilește conexiunea.
- La sosirea pachetelor, *lățimea de bandă necesară și capacitatea routerului vor fi deja pregătite*.
- Pentru o **subrețea bazată pe datagrame**, **evitarea congestiunii este mult mai dificilă**.

- Pentru **sistemele de prelucrare a tranzacțiilor** (de exemplu *apelurile magazinelor pentru a verifica cumpărături realizate cu cărți de credit*) supraincarcarea datorata stabilirii și eliberării unui circuit virtual poate reduce cu ușurință utilitatea circuitului.
- *Dacă majoritatea traficului este de acest tip, folosirea internă a circuitelor virtuale în cadrul subrețelei nu prea are sens.*
- Pe de altă parte, ar putea fi de folos circuite virtuale permanente, stabilite manual și care să dureze luni sau chiar ani.

1.5 Comparație între subrețele cu circuite virtuale și subrețele datagramă

Circuitele virtuale au o problemă de vulnerabilitate:

- Dacă un router se defectează și își pierde conținutul memoriei, *atunci toate circuitele virtuale care treceau prin el sunt suprimate*, chiar dacă acesta își revine după o secundă.
- Prin contrast, *dacă se defectează un router bazat pe datagrame* vor fi afectați doar acei utilizatori care aveau pachete memorate temporar în cozile de așteptare ale routerului și este posibil ca numărul lor să fie și mai mic, în funcție de câte pachete au fost deja confirmate.

1.5 Comparație între subrețele cu circuite virtuale și subrețele datagramă

- *Pierderea liniei de comunicație este fatală pentru circuitele virtuale care o folosesc, însă poate fi ușor compensată dacă se folosesc datagrame.*
- De asemenea, datagramele permit routerului să echilibreze traficul prin subrețea, deoarece căile pot fi modificate parțial în cursul unei secvențe lungi de pachete transmise.

Curs 8 (partea II)

2. Algoritmi de dirijare în rețea

Curs 8

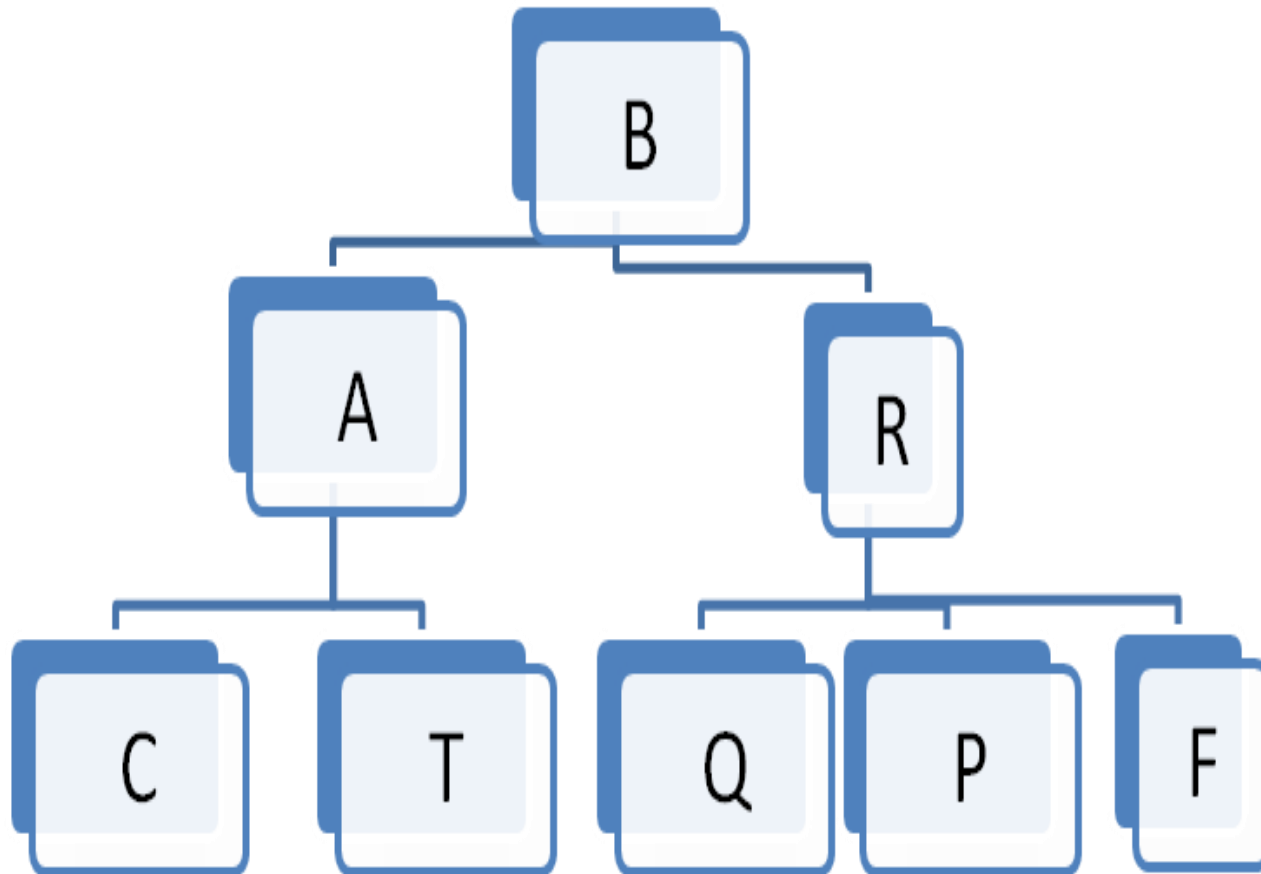
Algoritmi de dirijare

1. Algoritmul Dijkstra

2. Algoritmul Belmann-Ford

Principiul optimalității

- *Principiul optimalității presupune că dacă o rută optimă între nodul A și nodul B trece prin nodul C atunci ruta optimă între B și C este inclusă în ruta A-B.*
- O consecință a principiului optimalității este formarea **arborelui de scufundare** care semnifică *mulțimea rutelor optime către un nod destinație*, reprezentat de rădăcina arborelui.
- Scopul **algoritmilor de dirijare** constă în *descoperirea arborilor de scufundare pentru fiecare rută.*



Arbore de scufundare pentru nodul B

Algoritmi de dirijare

- *Algoritmul de dirijare este acea parte a software-lui nivelului rețea care răspunde de alegerea liniei de ieșire pe care un pachet recepționat trebuie trimis mai departe.*
- În cazul **datagramelor** această decizie trebuie luată din nou pentru fiecare pachet recepționat, deoarece e posibil ca cea mai bună rută să se fi modificat între timp.
- În cazul **circuitelor virtuale**, decizia de dirijare se ia doar la stabilirea unui nou circuit virtual.
- După aceea pachetele vor urma doar calea stabilită anterior.
- Acest ultim caz este numit uneori **sesiune de dirijare** deoarece calea rămâne în funcțiune pentru o întreagă sesiune utilizator (ex. transfer de fișiere, login prin terminal).

Clasificare

Algoritmii pot fi:

1. neadaptivi - *alegerea căii se calculează în avans (of line) și parvine routerului la inițializarea rețelei*

2. adaptivi - *își modifică deciziile de dirijare pentru a reflecta modificările de topologie și pe cele de trafic*

Algoritmii adaptivi diferă prin:

1. *locul de unde își iau informația:*

- a) local
- b) de la un vecin
- c) de la toate routerele

2. *momentul în care schimbă rutele:*

- a) când se schimbă încărcarea
- b) când se schimbă topologia
- c) la ΔT secunde

3. *metrica folosită pentru optimizare:*

- a) distanța
- b) timpul de tranzit
- c) numărul de salturi

1. Algoritmi statici de dirijare

1.1. Dirijarea pe calea cea mai scurtă (shortest path)

- *Calea cea mai scurtă este dictată de metrica de măsurare a distanței.*
- Aceasta poate fi:
 - distanța în km
 - numărul de salturi
 - rata de transfer
 - traficul mediu
 - costul comunicației
 - lungimea minimă a cozilor de așteptare
 - întârzieri măsurate

1.2. Algoritmul de inundare - generează foarte multe pachete.

Pentru a limita numărul acestora se procedează astfel:

- 1. contor de salturi în antetul fiecărui pachet decrementat la fiecare salt și care face ca pachetul să fie distrus atunci când contorul atinge valoarea 0.*
- 2. routerul sursă să plaseze un număr de secvență în fiecare pachet pe care îl primește de la calculatorul gazdă asociat.*

1.2. Algoritmul de inundare

- *Fiecare router necesită menținerea unei liste pentru fiecare router sursă, cu numere de secvență inițiate de acel router sursă și care au fost deja trimis mai departe.*
- Dacă sosește un pachet care deja se află în listă el nu se mai trimite.
- Se poate însoți lista de un contor k care semnifică faptul că toate numerele de secvență până la k au fost deja tratate.
- Inundarea se utilizează:
 - în aplicații militare
 - în aplicații cu baze de date distribuite în care este necesar să fie actualizate concurent

1.3. Dirijarea bazată pe flux

- În condițiile în care traficul mediu de la i la j este cunoscut în avans și într-o aproximare rezonabilă, constant în timp, este posibilă analiza matematică a fluxurilor pentru a optimiza dirijarea.
- Ideea care stă la baza analizei este aceea că pentru o anumită linie dacă se cunosc capacitatea și fluxul mediu, este posibil să se calculeze **întârzierea medie a unui pachet pe linia respectivă**, folosind teoria cozilor.

- Pe baza întârzierilor medii ale tuturor liniilor se poate calcula imediat, ca medie ponderată după flux, întârzierea medie a unui pachet pentru întreaga subrețea.
- *Problema dirijării se reduce apoi la găsirea algoritmului de dirijare care produce întârzierea medie minimă pentru subrețea.*
- Folosirea acestei tehnici presupune:
 - cunoașterea topologiei subrețelei
 - matricea traficului
 - matricea capacităților liniilor, C_i (măsurată în Kbps)

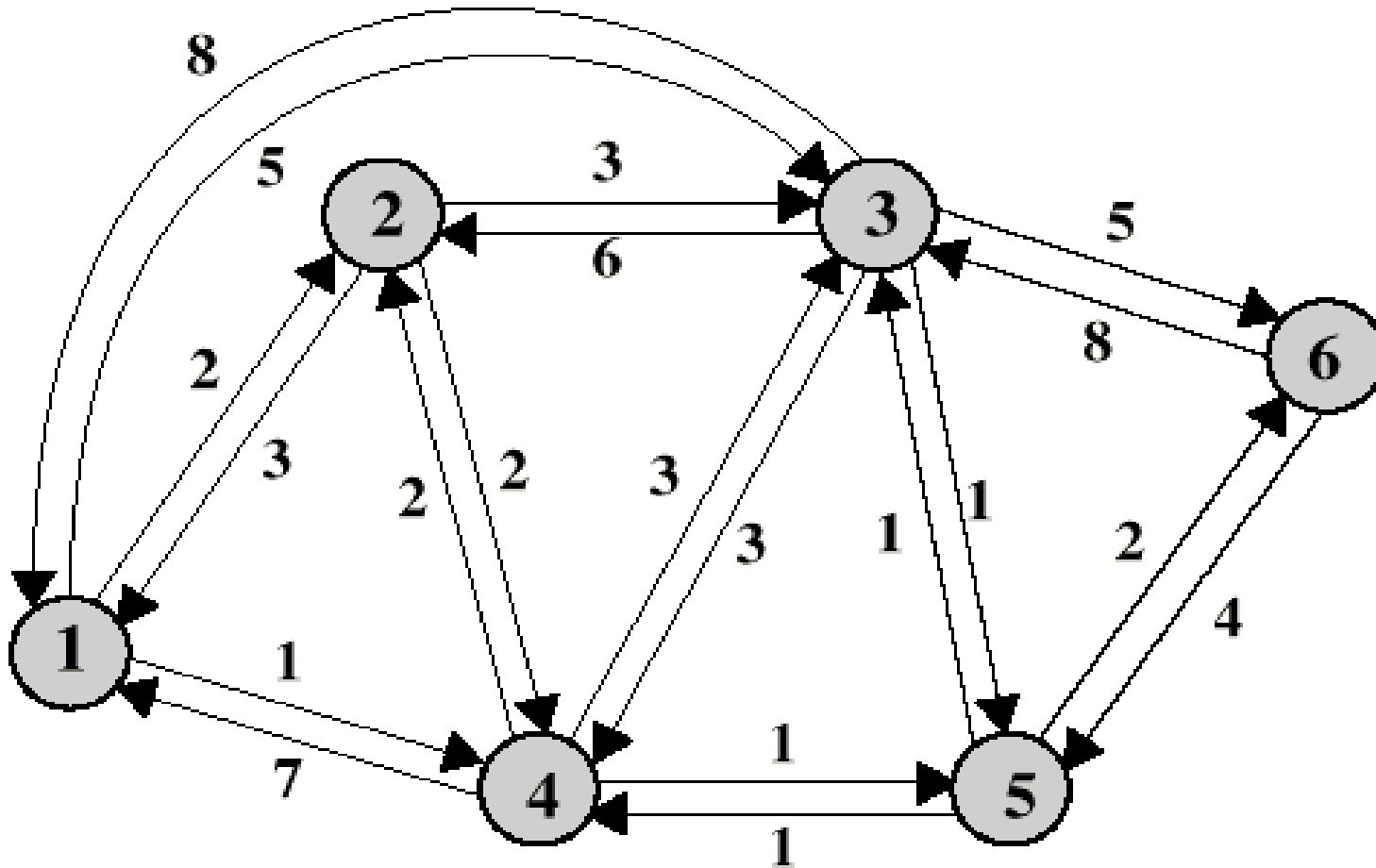
2. Algoritmi dinamici de dirijare

Cei mai folosiți sunt:

1. Algoritmul de dirijare cu vectori distanță
2. Algoritmul de dirijare bazat pe starea legăturilor

Algoritmul de dirijare cu vectori distanță (Bellman - Ford - Fulkerson)

Costurile rutelor



Strategii de Rutare

1. Rutare fixă
2. Rutare bazată pe inundare
3. Rutare aleatoare
4. Rutare adaptivă

1. Rutare Fixă

- O singură cale pentru fiecare pereche **sursă-destinație**
- Rutele sunt determinate printr-un *algorithm de cost minim*
- Rutele rămân fixe, până la schimbarea topologiei rețelei

Tabele de Rutare Fixe

CENTRAL ROUTING DIRECTORY

From Node

To Node	1	2	3	4	5	6
1	—	1	5	2	4	5
2	2	—	5	2	4	5
3	4	3	—	5	3	5
4	4	4	5	—	4	5
5	4	4	5	5	—	5
6	4	4	5	5	6	—

Node 1 Directory

Destination	Next Node
2	2
3	4
4	4
5	4
6	4

Node 2 Directory

Destination	Next Node
1	1
3	3
4	4
5	4
6	4

Node 3 Directory

Destination	Next Node
1	5
2	5
4	5
5	5
6	5

Node 4 Directory

Destination	Next Node
1	2
2	2
3	5
5	5
6	5

Node 5 Directory

Destination	Next Node
1	4
2	4
3	3
4	4
6	6

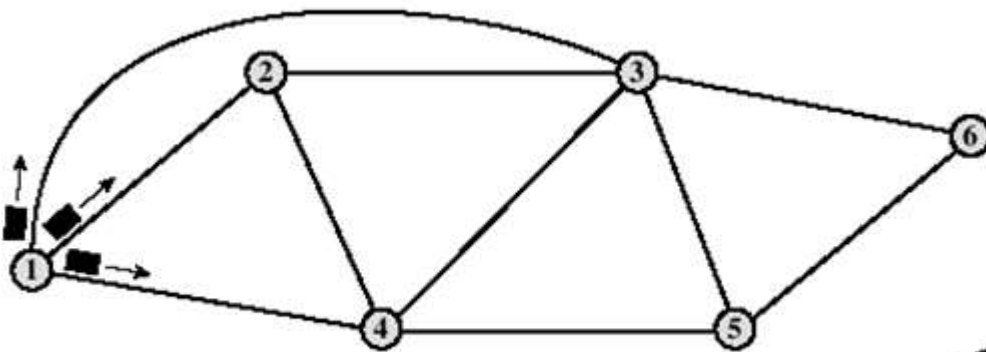
Node 6 Directory

Destination	Next Node
1	5
2	5
3	5
4	5
5	5

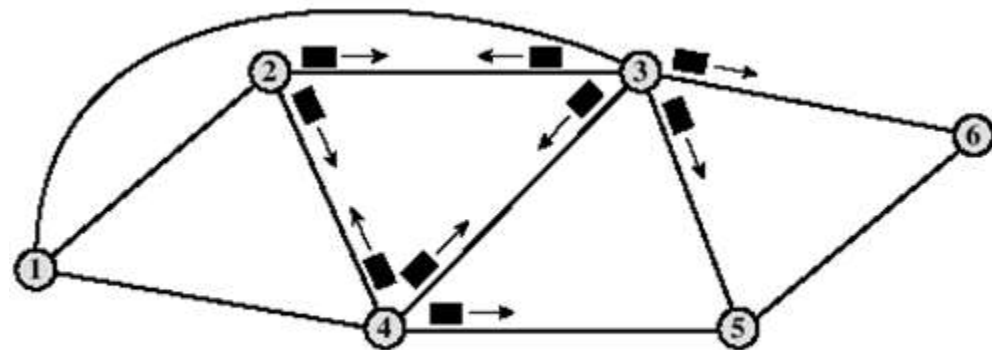
2. Rutare bazata pe inundare

- Nu sunt necesare informatii despre rețea
- *Pachetul este trimis la toți vecinii*
- *Sau la toți în afară de unde a venit*
- Un număr de copii ajung după un timp la destinație
- Fiecare pachet are un număr unic, duplicatele se ignoră
- Nodurile pot reține identitatea pachetelor pentru a nu le ruta din nou
- Se poate defini un timp de viață a pachetelor

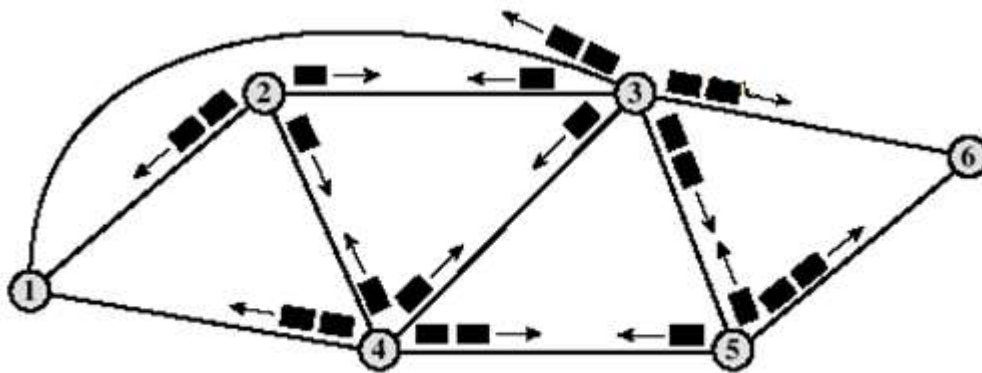
Inundare Exemplu



(a) Primul salt



(b) Al doilea salt



(c) Al treilea salt

Proprietăți ale Inundării

- TOATE rutele posibile sunt încercate
 - foarte robust
- Cel puțin un pachet va ajunge pe calea de cost minim
 - Se poate folosi pt. stabilirea unui circuit virtual
- Toate nodurile sunt atinse
 - Util pentru distribuirea de informații (ex. rutare)

3. Rutare Aleatoare

- *Nodul selectează o cale de ieșire pentru transmiterea unui pachet primit*
- Selecția poate fi aleatoare sau round-robin (fiecare nod este analizat)
- Se pot utiliza și probabilități
- *Nu sunt necesare informații despre rețea*
- Ruta nu este în general optimă
- Trafic inutil mai mic ca la inundare

4. Rutare Adaptivă - caracteristici

- **Rutarea adaptiva** este cel mai des utilizată
- Decizia de rutare se adaptează condițiilor din rețea:
 - Defecte de linie sau noduri
 - Congestie
- *Necesită informații despre rețea*
- Decizia este mai complexă
- Compromis între calitatea rețelei și overhead (supraîncărcare)
- Reacție prea rapidă produce oscilații
- Prea încet pentru a fi relevant

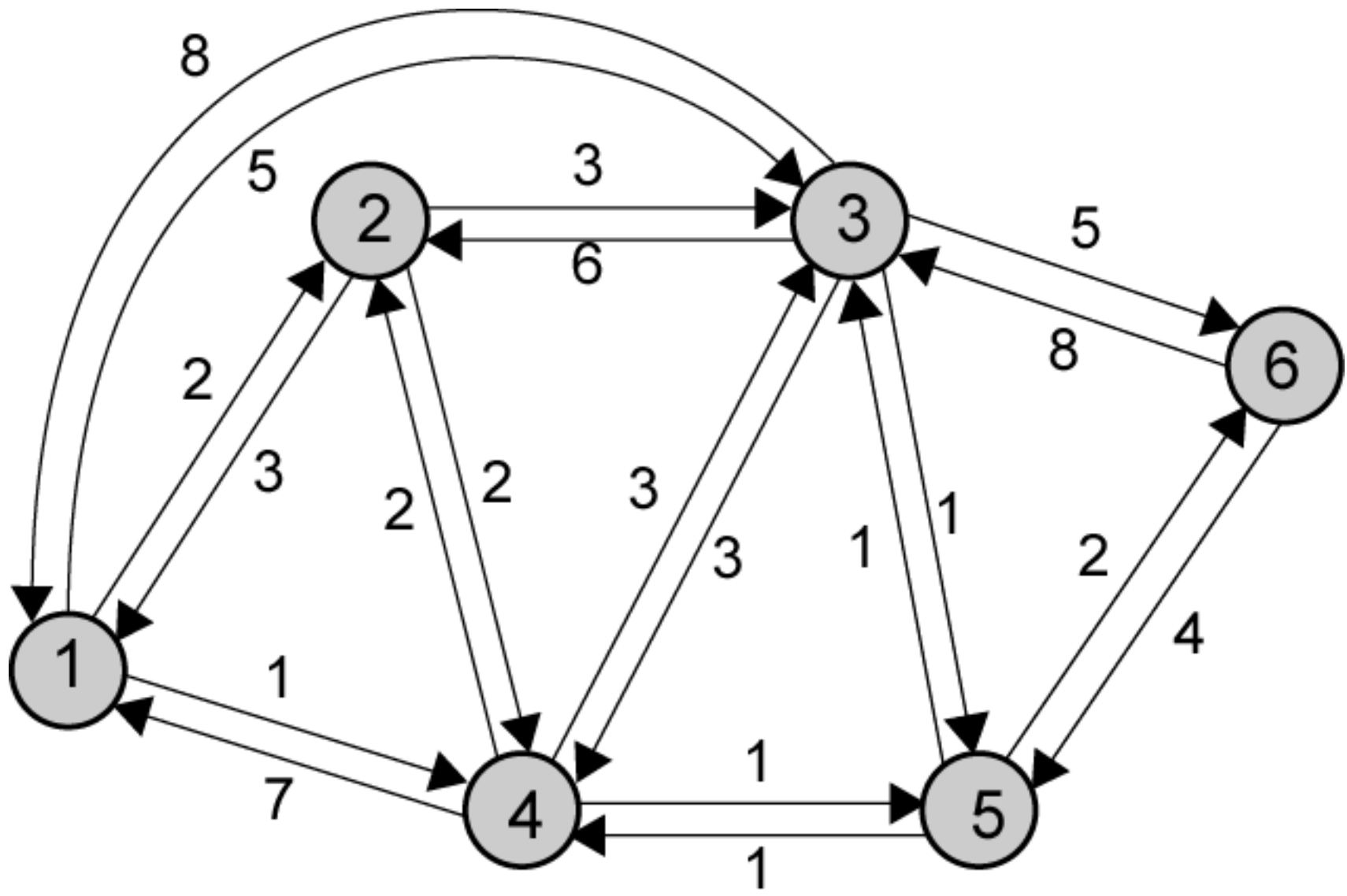
Rutare Adaptivă – Avantaje

1. Creșterea performanței
2. Ajută la controlul congestiei
3. Sistem Complex
 - Poate să nu ajungă la beneficiile teoretice

Criteria de selectie a rutelor

- Minimum hop count (numărul de salturi)
- Minimum cost

Retea



Algoritm de cost minim

- Decizia de rutare este data de:
 - Numar de hopuri(salturi)
 - Cost link invers proportional cu capacitatea
- Retea cu link-uri bidirectionale
- Cost asociat in ambele directii
- Costul caii de rutare = suma costurilor pe link
- Se cauta *calea de cost minim pentru toate perechile sursa si destinatie*
- Costurile pot fi diferite pe cele doua sensuri

Curs 8

Algoritmi de dirijare

1. Algoritmul Dijkstra

2. Algoritmul Belmann-Ford

Algoritm Dijkstra - definiții

➤ *Caută calea cea mai scurtă de la un nod sursă la toate nodurile prin dezvoltarea de căi de lungime din ce în ce mai mare*

Notatii:

- N = multimea de noduri în rețea
- s = nod sursă
- T = multimea de noduri încorporate în algoritm
- $w(i, j)$ = **costul legaturii de la nod i la nod j**
 - $w(i, i) = 0$
 - $w(i, j) = \infty$ dacă nu sunt direct conectate
 - $w(i, j) \geq 0$ dacă sunt direct conectate
- $L(n)$ = costul căii cunoscute cea mai ieftină de la nod s la nodul n
 - la terminare, **$L(n)$ este costul căii celei mai ieftine de la s la n**

Algoritm Dijkstra - metodă

Pas 1 [Inițializare]

- $T = \{s\}$ Nod sursă
- $L(n) = w(s, n)$ pentru $n \neq s$
- Initial doar costuri link

Pas 2 [Următorul Nod]

- Caută nod vecin necuprins în T cu cost minim față de s
- Adaugare nod în T

Pas 3 [Actualizare cale cea mai ieftină]

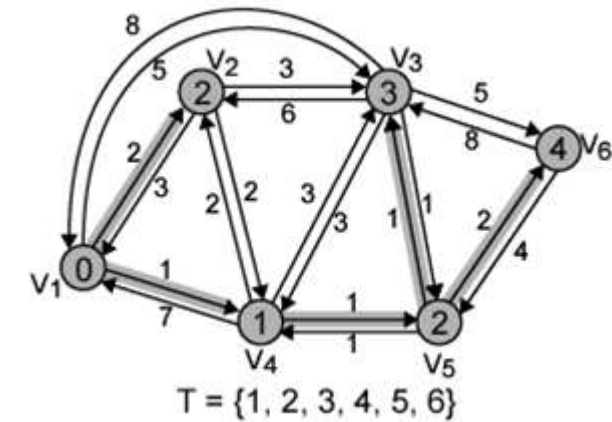
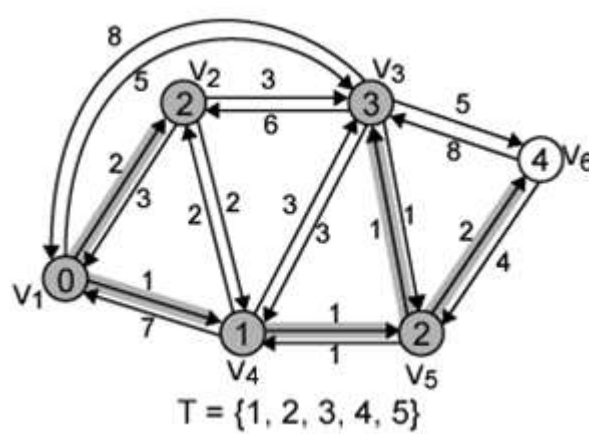
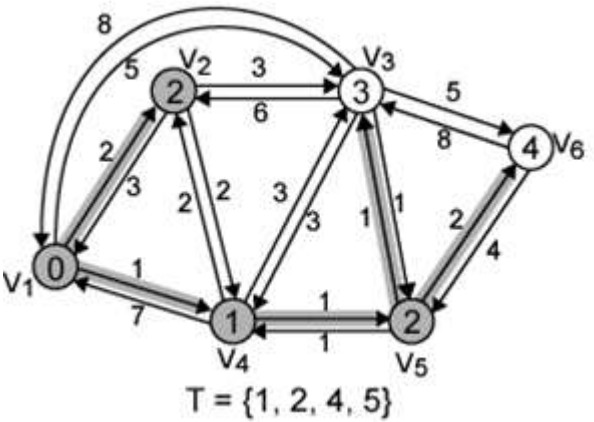
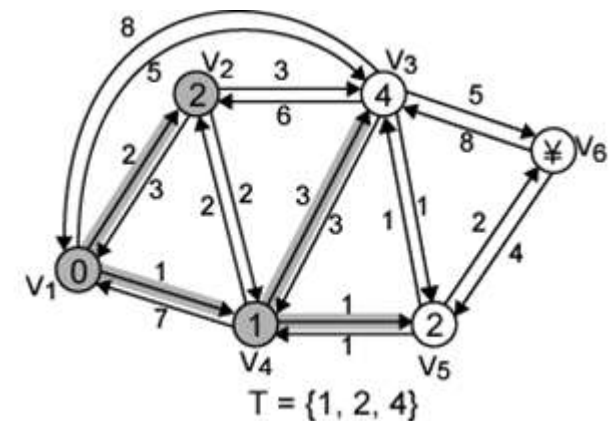
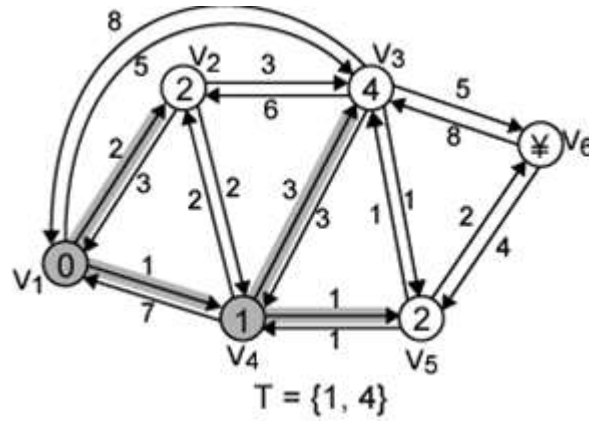
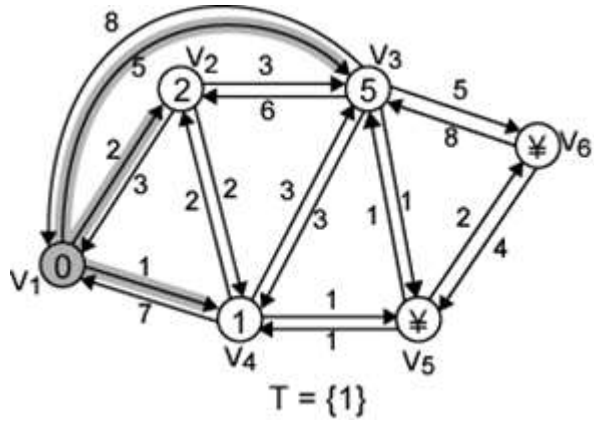
- $L(n) = \min[L(n), L(x) + w(x, n)]$ pentru toți $n \notin T$
- Dacă al doilea termen este minimul, actualizare cale prin concatenare

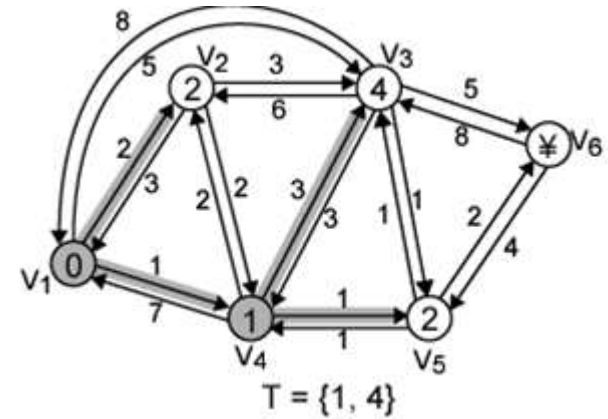
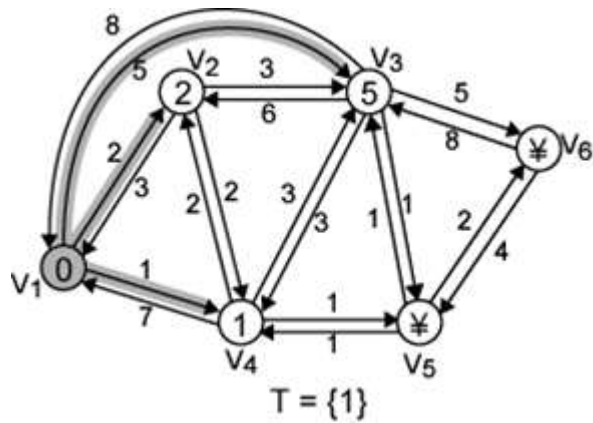
[Terminare] Algoritmul se termină după adaugarea tuturor nodurilor

Algoritm Dijkstra - observații

- La terminare, *valoarea $L(x)$ asociată fiecărui nod x este costul (lungimea) căii de cost minim de la s la x .*
- *Multimea T definește calea cea mai ieftină de la s la fiecare alt nod*
- O iterație a pașilor 2 și 3 adaugă un nou nod în T

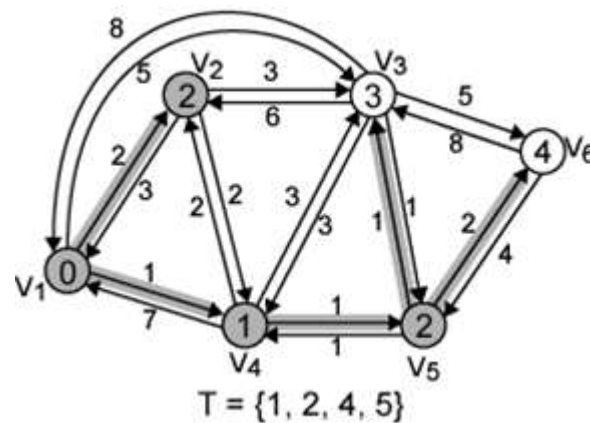
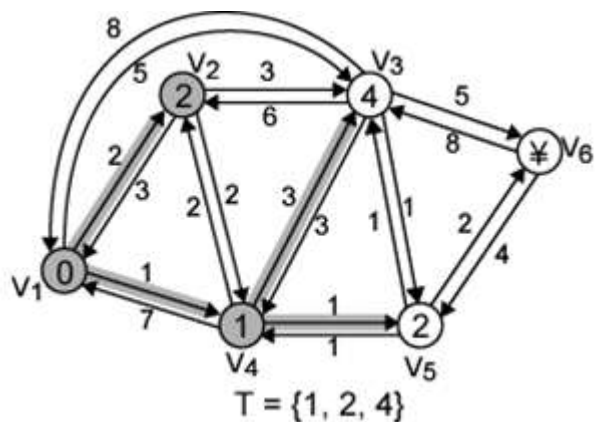
Algoritm Dijkstra - exemplu





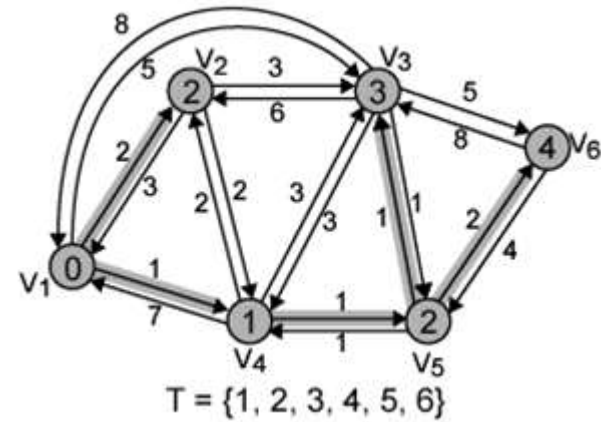
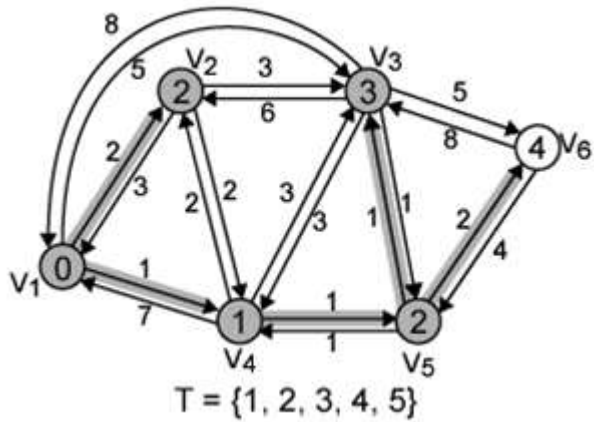
Iteratii	T	L(2)	Cale	L(3)	Cale	L(4)	Cale	L(5)	Cale	L(6)	Cale
1	{1}	2	1-2	5	1-3	1	1-4	∞	-	∞	-

Iteratii	T	L(2)	Cale	L(3)	Cale	L(4)	Cale	L(5)	Cale	L(6)	Cale
2	{1,4}	2	1-2	4	1-4-3	1	1-4	2	1-4-5	∞	-



Iteratii	T	L(2)	Cale	L(3)	Cale	L(4)	Cale	L(5)	Cale	L(6)	Cale
3	{1, 2, 4}	2	1-2	4	1-4-3	1	1-4	2	1-4-5	∞	-

Iteratii	T	L(2)	Cale	L(3)	Cale	L(4)	Cale	L(5)	Cale	L(6)	Cale
4	{1, 2, 4, 5}	2	1-2	3	1-4-5-3	1	1-4	2	1-4-5	4	1-4-5-6



Iteratii	T	L(2)	Cale	L(3)	Cale	L(4)	Cale	L(5)	Cale	L(6)	Cale
5	{1, 2, 3, 4, 5}	2	1-2	3	1-4-5-3	1	1-4	2	1-4-5	4	1-4-5-6

Iteratii	T	L(2)	Cale	L(3)	Cale	L(4)	Cale	L(5)	Cale	L(6)	Cale
6	{1, 2, 3, 4, 5, 6}	2	1-2	3	1-4-5-3	1	1-4	2	1-4-5	4	1-4-5-6

Rezultate la Exemplu

Algoritm Dijkstra

Iteratii	T	L(2)	Cale	L(3)	Cale	L(4)	Cale	L(5)	Cale	L(6)	Cale
1	{1}	2	1-2	5	1-3	1	1-4	∞	-	∞	-
2	{1,4}	2	1-2	4	1-4-3	1	1-4	2	1-4-5	∞	-
3	{1, 2, 4}	2	1-2	4	1-4-3	1	1-4	2	1-4-5	∞	-
4	{1, 2, 4, 5}	2	1-2	3	1-4-5-3	1	1-4	2	1-4-5	4	1-4-5-6
5	{1, 2, 3, 4, 5}	2	1-2	3	1-4-5-3	1	1-4	2	1-4-5	4	1-4-5-6
6	{1, 2, 3, 4, 5, 6}	2	1-2	3	1-4-5-3	1	1-4	2	1-4-5	4	1-4-5-6

Curs 8

Algoritmi de dirijare

1. Algoritmul Dijkstra

2. Algoritmul Belmann-Ford

Algoritm Bellman-Ford. Definiții

- Caută calea cea mai ieftină formată din cel mult un link
- Caută calea cea mai ieftină formată din cel mult două link-uri
- ș.a.m.d.

Notatii:

- s = nod sursă
- $w(i, j)$ = cost link de la nod i la nod j
 - $w(i, i) = 0$
 - $w(i, j) = \infty$ dacă cele două noduri nu sunt conectate direct
 - $w(i, j) \geq 0$ dacă cele două noduri sunt conectate direct
- h = număr maxim link-uri în cale în pasul curent al algoritmului
- $L_h(n)$ = costul căii cea mai ieftine de la s la n cu mai mult de h link-uri

Algoritm Bellman-Ford. Metodă

Pas 1 [Inițializare]

- $L_0(n) = \infty$, pentru toți $n \neq s$
- $L_h(s) = 0$, pentru toți h

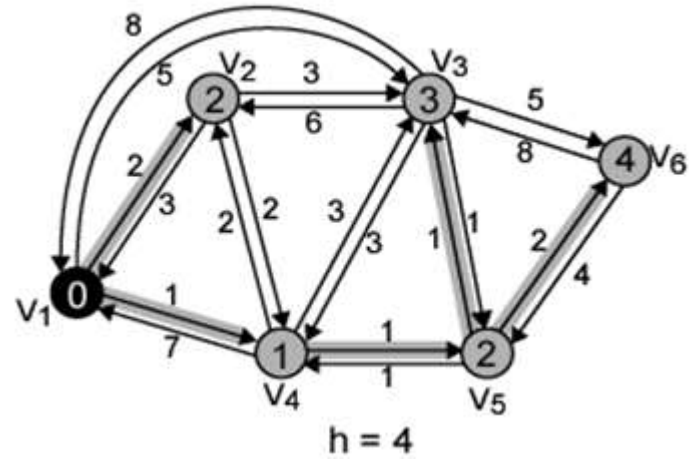
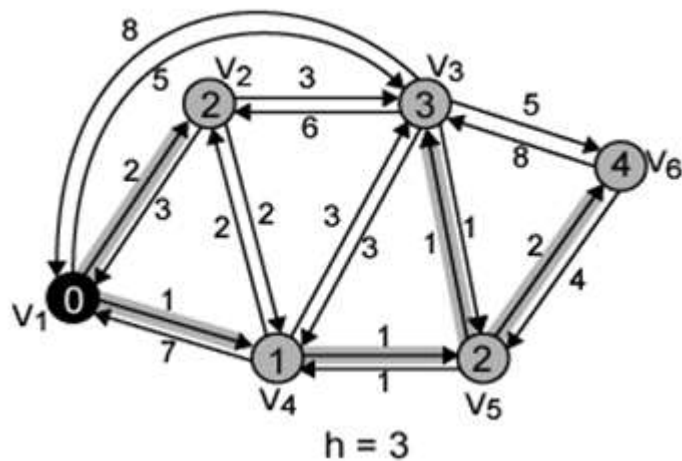
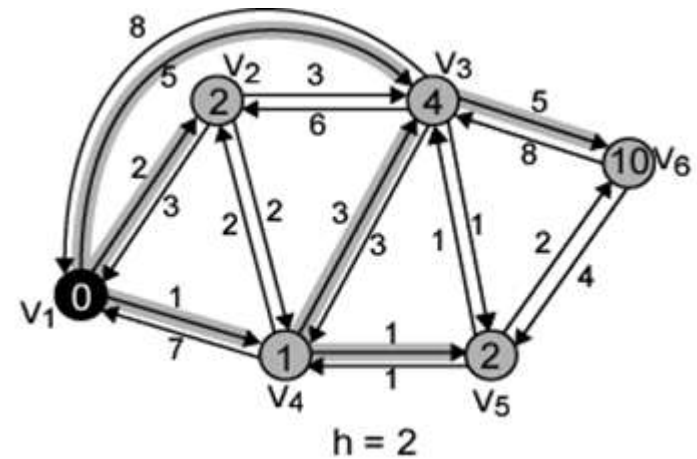
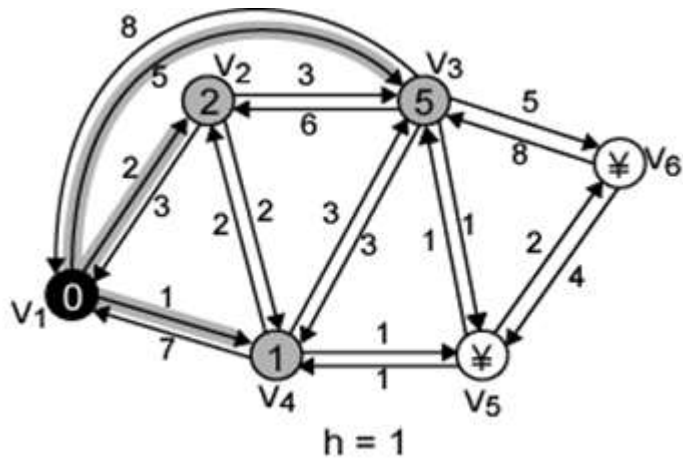
Pas 2 [Actualizare]

- Pentru fiecare $h \geq 0$ succesiv
- Pentru fiecare nod $n \neq s$, calculează
 - $L_{h+1}(n) = \min_j [L_h(j) + w(j, n)]$
- Conectează n cu nod predecesor j care atinge minimum
- *Elimină orice conexiune a unui nod n cu un nod predecesor diferit format în iterație anterioară*
- Calea de la s la n se termină cu link de la j la n

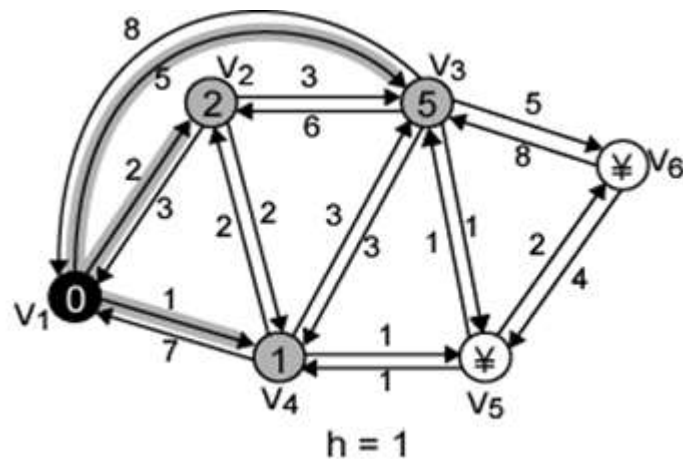
Observații Algoritm Bellman-Ford

- *La fiecare iterație a pasului 2 pentru $h=K$ și fiecare nod destinație n , algoritmul compară căile de la s la n de lungime $K=1$ cu calea de la iterația anterioară*
- În funcție de cost se menține calea anterioară sau se actualizează

Exemplu Algoritm Bellman-Ford

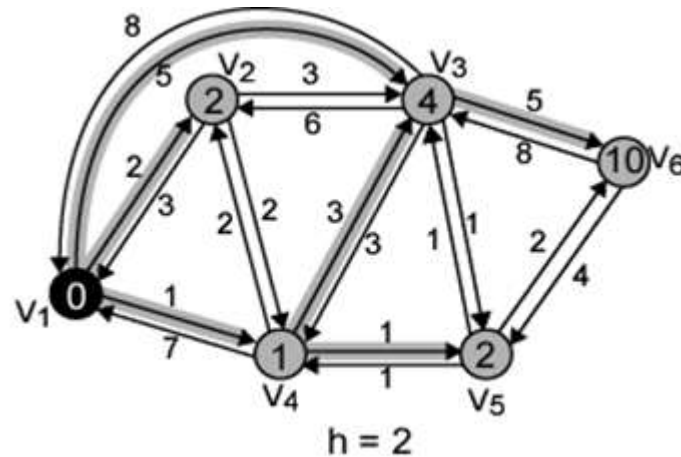


Exemplu Algoritm Bellman-Ford



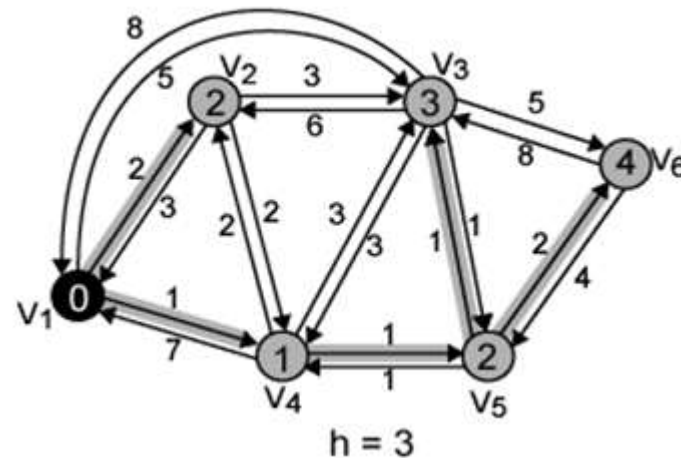
h	$L_h(2)$	Cale	$L_h(3)$	Cale	$L_h(4)$	Cale	$L_h(5)$	Cale	$L_h(6)$	Cale
0	∞	-	∞	-	∞	-	∞	-	∞	-

Exemplu Algoritm Bellman-Ford



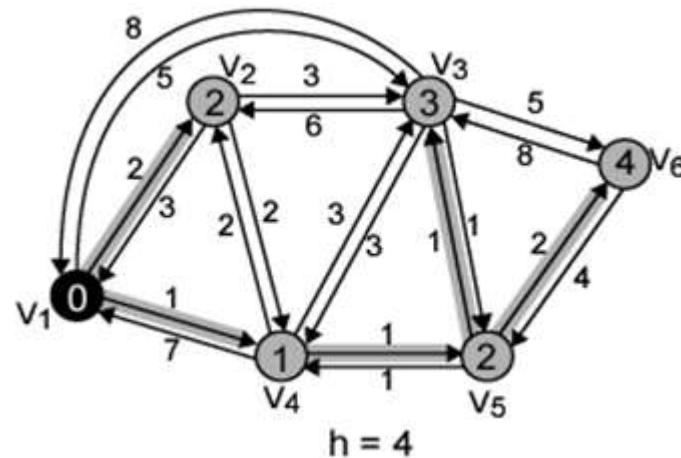
h	$L_h(2)$	Cale	$L_h(3)$	Cale	$L_h(4)$	Cale	$L_h(5)$	Cale	$L_h(6)$	Cale
1	2	1-2	5	1-3	1	1-4	∞	-	∞	-

Exemplu Algoritm Bellman-Ford



h	$L_h(2)$	Cale	$L_h(3)$	Cale	$L_h(4)$	Cale	$L_h(5)$	Cale	$L_h(6)$	Cale
2	2	1-2	4	1-4-3	1	1-4	2	1-4-5	10	1-3-6

Exemplu Algoritm Bellman-Ford



h	$L_h(2)$	Cale	$L_h(3)$	Cale	$L_h(4)$	Cale	$L_h(5)$	Cale	$L_h(6)$	Cale
3	2	1-2	3	1-4-5-3	1	1-4	2	1-4-5	4	1-4-5-6

Rezultate Exemplu Bellman-Ford

h	$L_h(2)$	Cale	$L_h(3)$	Cale	$L_h(4)$	Cale	$L_h(5)$	Cale	$L_h(6)$	Cale
0	∞	-	∞	-	∞	-	∞	-	∞	-
1	2	1-2	5	1-3	1	1-4	∞	-	∞	-
2	2	1-2	4	1-4-3	1	1-4	2	1-4-5	10	1-3-6
3	2	1-2	3	1-4-5-3	1	1-4	2	1-4-5	4	1-4-5-6
4	2	1-2	3	1-4-5-3	1	1-4	2	1-4-5	4	1-4-5-6

Comparație Bellman-Ford cu Dijkstra

➤ Rezultatele celor doi algoritmi coincid

➤ Informații colectate:

– Bellman-Ford

- Calculul în nodul n necesită cunoașterea costului link-ului la toți vecinii plus totalul costului de la fiecare vecin la s
- Fiecare nod poate menține un set de costuri și căi pentru fiecare alt nod
- Face schimb de informații cu vecinii direcți
- Poate actualiza costurile și căile pe baza informațiilor de la vecini și costurile linkurilor locale

– Dijkstra

- Fiecare nod are nevoie de topologia completă
- Trebuie să cunoască costul tuturor linkurilor din rețea
- Schimbă informații cu toate celelalte noduri

Bibliografie

1. Rețele de calculatoare, ANDREW S. TANENBAUM, ediția a treia, 1998, Editura Computer Press Agora
2. Rețele de calculatoare, ANDREW S. TANENBAUM, ediția a patra, 2000, Editura Computer Press Agora – varianta electronică
3. Rețele de calculatoare, Valentin Cristea, Nicolae Tapus, Trandafir Moisa, Valeriu Damian, 1992, Editura Teora.

Întrebări?