Universitatea "Constatin Brâncuşi" din Târgu-Jiu
Facultatea de Inginerie
Departamentul de Automatică, Energie şi Mediu

Rețele de calculatoare

Lect. dr. Adrian Runceanu

An universitar 2013-2014

Curs 8

1. Cerinţele de proiectare ale nivelului reţea
2. Algoritmi de dirijare

Nivelul rețea

1. Cerinţele de proiectare ale nivelului reţea

- *1.1 Comutare de pachete de tip Memorează-şi-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 Comparaţie între subreţele cu circuite virtuale şi subreţele datagramă*

2. Algoritmi de dirijare

- *2.1 Principiul optimalităţii*
- *2.2 Dirijarea pe calea cea mai scurtă*

1. Cerințele de proiectare ale nivelului rețea

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

Contextul în care operează protocoalele de la nivelul reţea:

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

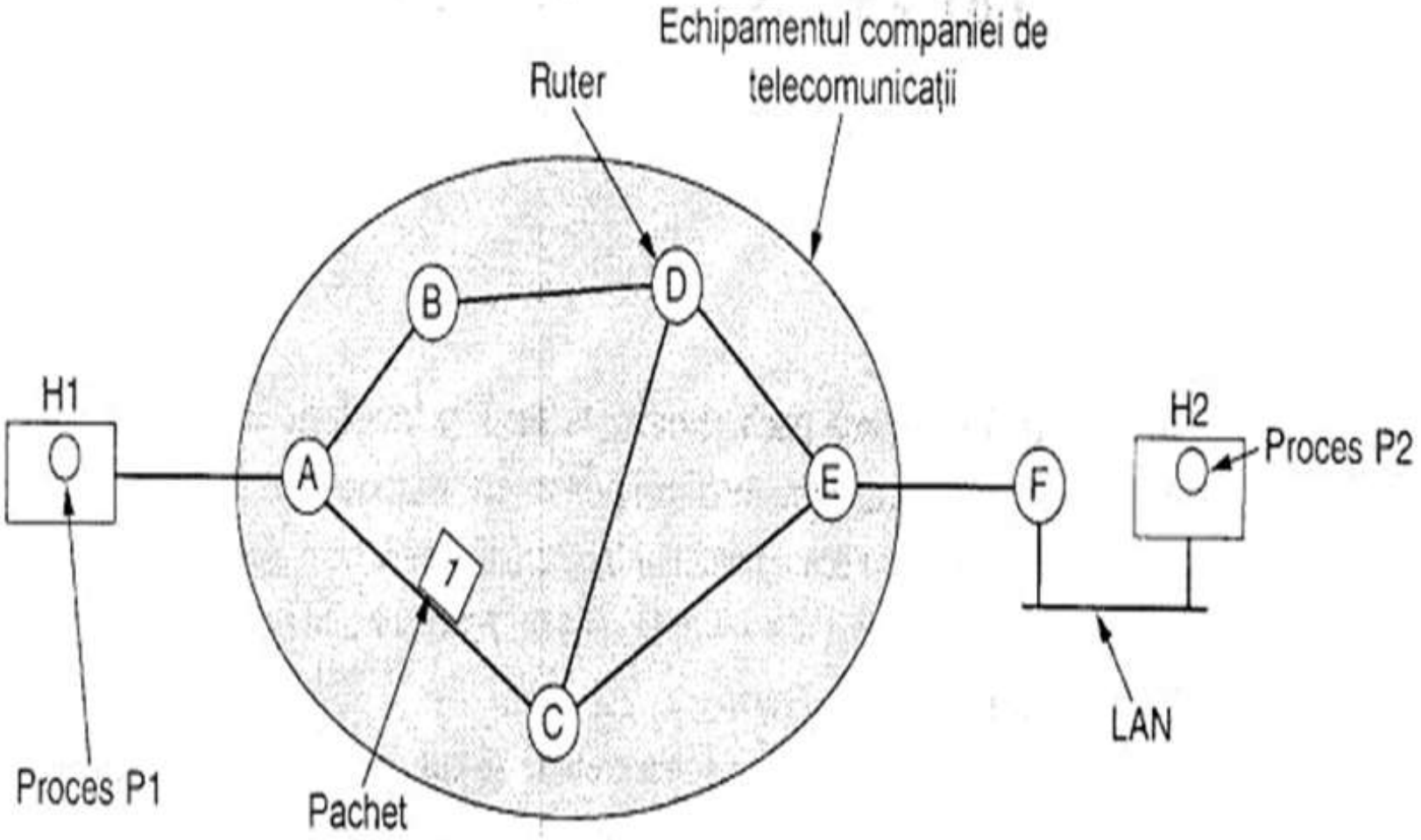

Fig. 1. Cadrul protocoalelor nivelului rețea

1.1 Comutare de pachete de tip Memorează-şi-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 aceeaşi reţea LAN, fie printr-o legătură punct la punct cu compania de telecomunicaţii.
- 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 destinaţie, unde este livrat.
- Acest mecanism reprezintă comutarea de pachete de tip memorează-şi-retransmite.

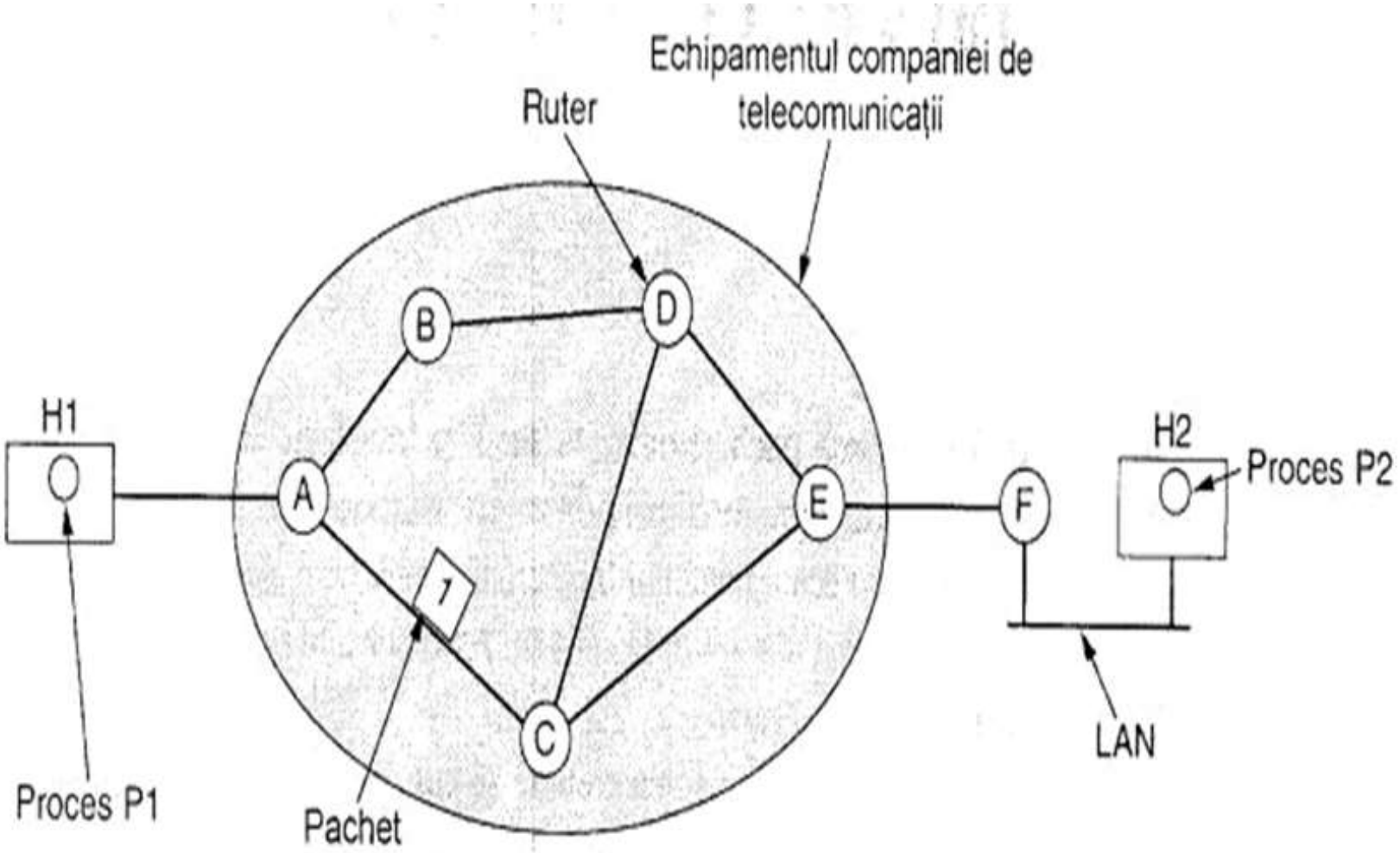

Fig. 1. Cadrul protocoalelor nivelului rețea

Nivelul rețea

1. Cerinţele de proiectare ale nivelului reţea
   - *1.1 Comutare de pachete de tip Memorează-şi-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 Comparaţie între subreţele cu circuite virtuale şi subreţele datagramă*
2. Algoritmi de dirijare
   - *2.1 Principiul optimalităţii*
   - *2.2 Dirijarea pe calea cea mai scurtă*

1.2 Servicii furnizate nivelului transport

Serviciile nivelului reţea 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 şi topologia routerelor existente*.
3. *Adresele de reţea* disponibile la nivelul transport *trebuie să folosească o schemă de numerotare uniformă*, chiar în cadrul reţelelor **LAN** şi **WAN**.

- Obiectivele fiind stabilite, proiectantul nivelului reţea are o mare libertate în a scrie specificaţiile detaliate ale serviciilor oferite nivelului transport.
- Această libertate degenerează adesea pareri diferite între două tabere opuse.
- Problema centrală a discuţiei este dacă nivelul reţea trebuie să furnizeze:
  1. *servicii orientate pe conexiune*
  2. *servicii neorientate pe conexiune*

1.2 Servicii furnizate nivelului transport

- O tabără (reprezentată de comunitatea Internet) afirmă că *scopul routerului este de a transfera pachete şi nimic mai mult*.
- În viziunea lor (bazată pe experienţa a peste 30 de ani de exploatare a unei reţele de calculatoare în funcţiune), subreţeaua este inerent nesigură, indiferent cum ar fi proiectată.
- De aceea calculatoarele gazdă trebuie să accepte faptul că reţeaua este nesigură şi să facă controlul erorilor (adica, detecţia şi corecţia erorii) şi controlul fluxului ele însele.

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

1.2 Servicii furnizate nivelului transport

- Cealaltă tabără (reprezentată de companiile de telefoane) afirmă că *subreţeaua trebuie să asigure un serviciu orientat pe conexiune sigur.*
- Ei susţin că peste 100 de ani de experienţă cu sistemul telefonic mondial reprezintă un ghid excelent.
- În această perspectivă, calitatea serviciului este elementul dominant, şi într-o subreţea fără conexiuni, calitatea serviciului este dificil de obţinut, în special pentru trafic în timp real cum ar fi voce şi imagine.

Aceste două tabere sunt cel mai bine exemplificate de reteaua Internet şi reţelele ATM.

1. Reţeaua Internet oferă un *serviciu la nivelul reţea neorientat pe conexiune*
2. Reţelele ATM oferă un *serviciu la nivelul reţea orientat pe conexiune*

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

Nivelul rețea

1. Cerinţele de proiectare ale nivelului reţea

1.1 Comutare de pachete de tip Memorează-şi-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 Comparaţie între subreţele cu circuite virtuale şi subreţele datagramă

2. Algoritmi de dirijare

2.1 Principiul optimalităţii

2.2 Dirijarea pe calea cea mai scurtă

1.3 Implementarea serviciului neorientat pe conexiune

După ce am văzut cele două clase de servicii pe care nivelul reţea le furnizează utilizatorilor săi, este momentul să vedem funcţionarea internă a acestui nivel.

Sunt posibile două organizări diferite, în funcţie de tipul serviciului oferit:

1. serviciu 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 subreţea individual şi dirijate independent de celelalte*.

- Nu este necesară nici o iniţializare prealabilă.
- În acest context, pachetele sunt numite frecvent **datagrame** (datagrams) (prin analogie cu telegramele), iar subreţeaua este numită **subreţea 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 destinaţie*.

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

1.3 Implementarea serviciului neorientat pe conexiune

Să vedem cum funcţionează o **subreţea datagramă**:

- Să presupunem că procesul P1 din fig. 2 are un mesaj lung pentru procesul P2.
- El transmite mesajul nivelului transport, cu instrucţiunile 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 şi transferă rezultatul nivelului reţea, probabil o altă procedură din cadrul sistemului de operare.

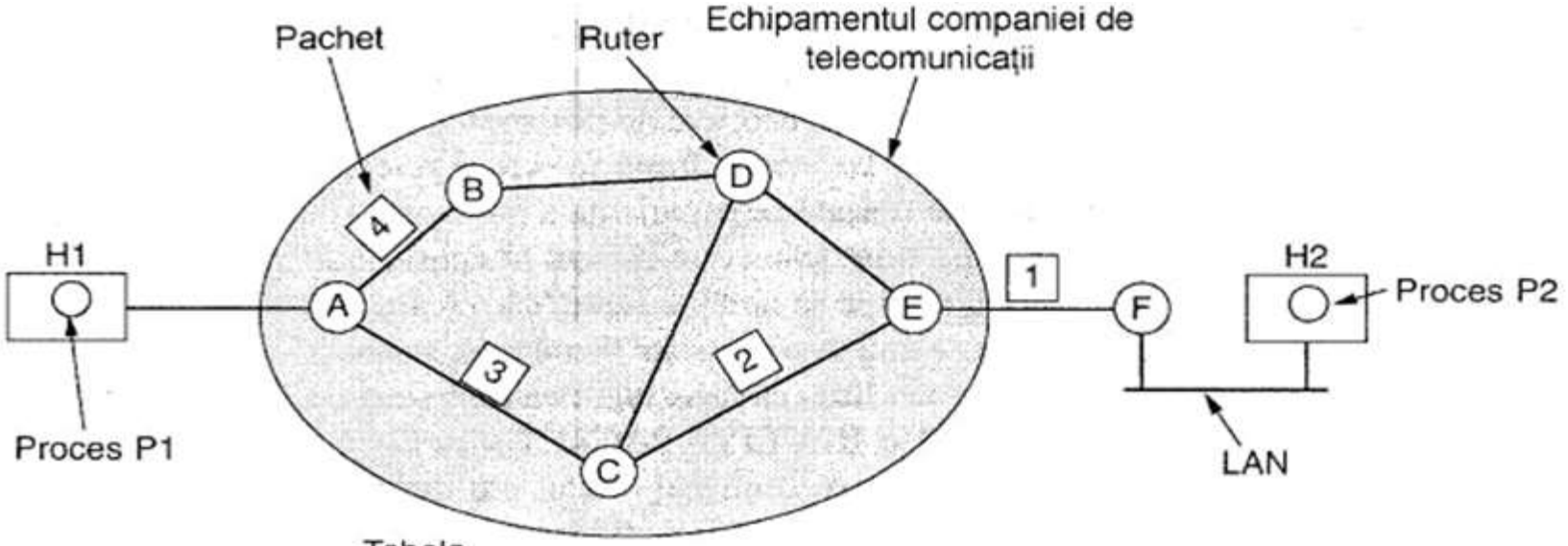

Tabela ruterului A

iniţial		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 subreţea datagramă

1.3 Implementarea serviciului neorientat pe conexiune

- Să presupunem că mesajul este de patru ori mai lung decât dimensiunea maximă a unui pachet, aşa că nivelul reţea trebuie să îl spargă în patru pachete, 1, 2, 3, şi 4 şi 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 telecomunicaţii.
- Fiecare router are o tabelă internă care îi spune unde să trimită pachete pentru fiecare destinaţie posibilă.

1.3 Implementarea serviciului neorientat pe conexiune

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

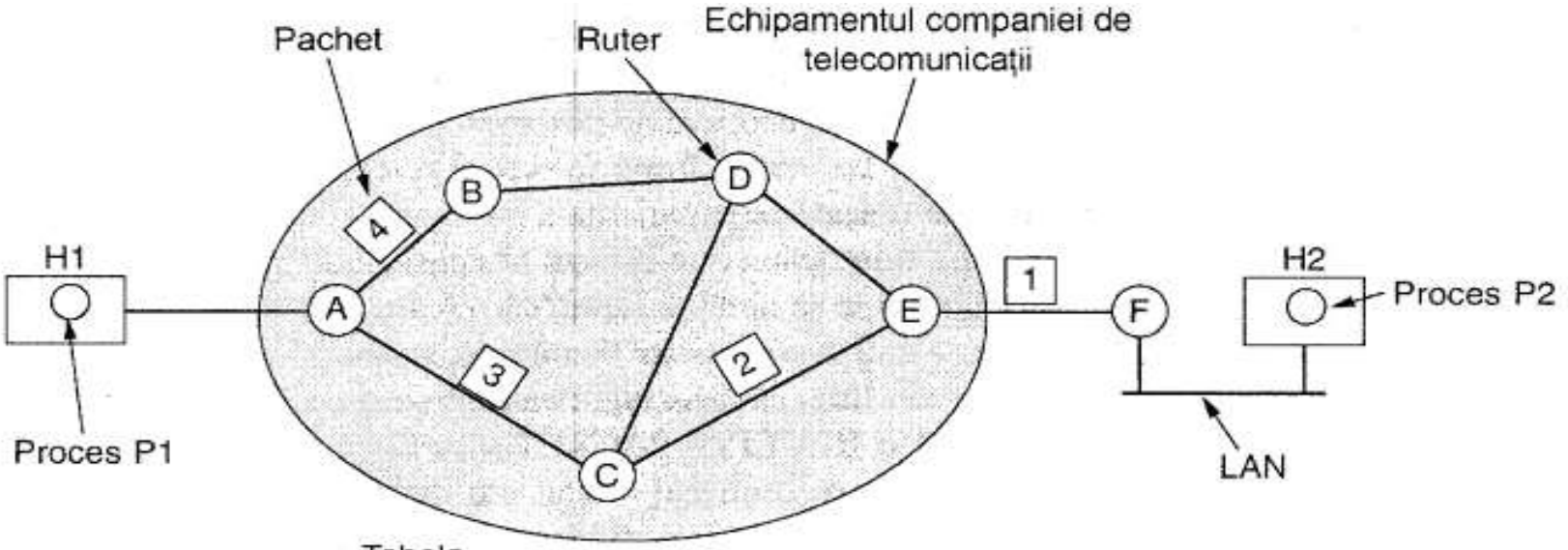

Tabela ruterului A

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. 5-2. Dirijarea într-o subrețea datagramă.

1.3 Implementarea serviciului neorientat pe conexiune

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

1.3 Implementarea serviciului neorientat pe conexiune

- Totuşi, ceva diferit s-a întâmplat cu pachetul 4.
- Când a ajuns la A, a fost trimis către routerul B, chiar dacă şi 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 şi şi-a actualizat tabela de rutare, aşa cum apare sub eticheta „ulterior".
- *Algoritmul ce administrează tabelele şi ia deciziile de rutare* se numeşte **algoritm de rutare** (routing algorithm).

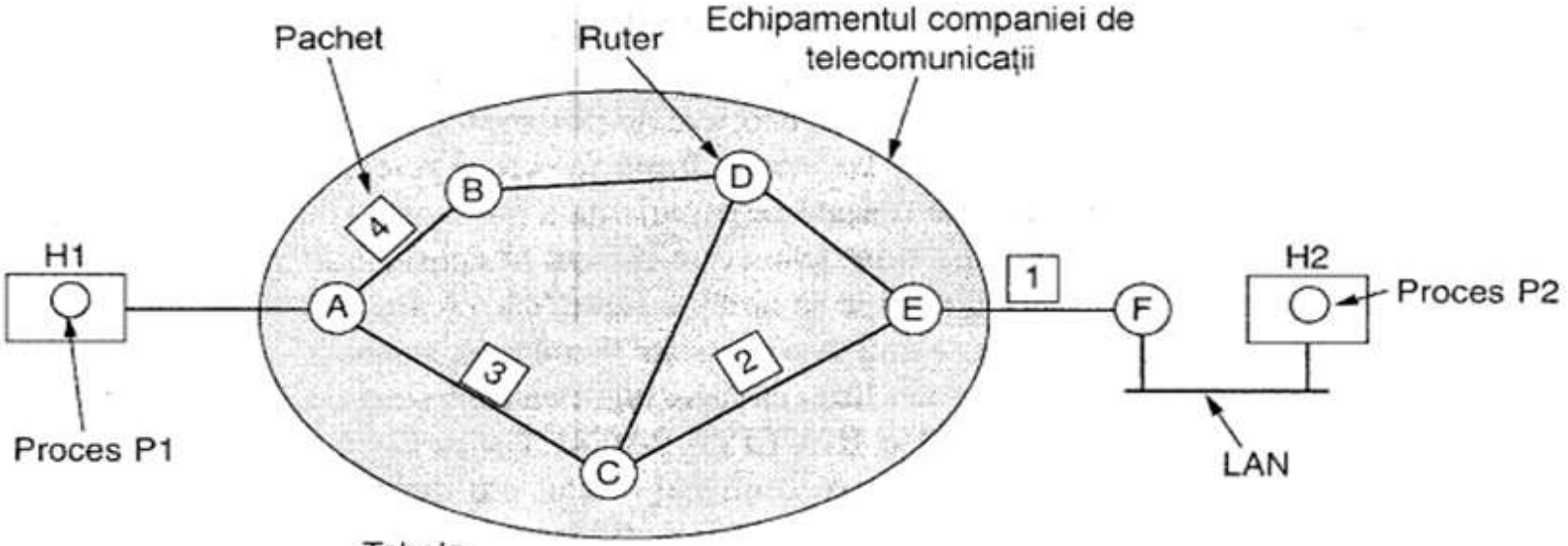

Tabela ruterului A

Dest.	inițial Rută	ulterior Rută
A	–	–
B	B	B
C	C	C
D	B	B
E	C	B
F	C	B

Tabela ruterului C

Dest.	Rută
A	A
B	A
C	–
D	D
E	E
F	E

Tabela ruterului E

Dest.	Rută
A	C
B	D
C	C
D	D
E	–
F	F

Fig. 2. Dirijarea într-o subrețea datagramă

Nivelul rețea

1. Cerinţele de proiectare ale nivelului reţea

1.1 Comutare de pachete de tip Memorează-şi-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 Comparaţie între subreţele cu circuite virtuale şi subreţele datagramă

2. Algoritmi de dirijare

2.1 Principiul optimalităţii

2.2 Dirijarea pe calea cea mai scurtă

1.4 Implementarea serviciilor orientate pe conexiune

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

Să vedem cum funcţionează:

- 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 stabileşte o conexiune, se alege o cale între maşina sursă şi maşina destinaţie, ca parte componentă a iniţializării conexiunii şi aceasta este memorată în tabelele routerelor.
- Acea cale este folosită pentru tot traficul de pe conexiune, exact în acelaşi mod în care funcţionează sistemul telefonic.
- Atunci când conexiunea este eliberată, este închis şi circuitul virtual.
- În cazul serviciilor orientate conexiune, *fiecare pachet poartă un identificator care spune cărui circuit virtual îi aparţine*.

1.4 Implementarea serviciilor orientate pe conexiune

- De exemplu, să considerăm situaţia 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 tabelei lui 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 iniţializează conexiunea şi aceasta este singura conexiune) şi indică subreţelei să stabilească circuitul virtual.
- Aceasta conduce la a doua linie din tabele.
- Observaţi că apare un conflict deoarece deşi A poate distinge uşor 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 ieşire 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 numeşte **comutarea etichetelor** (label switching).

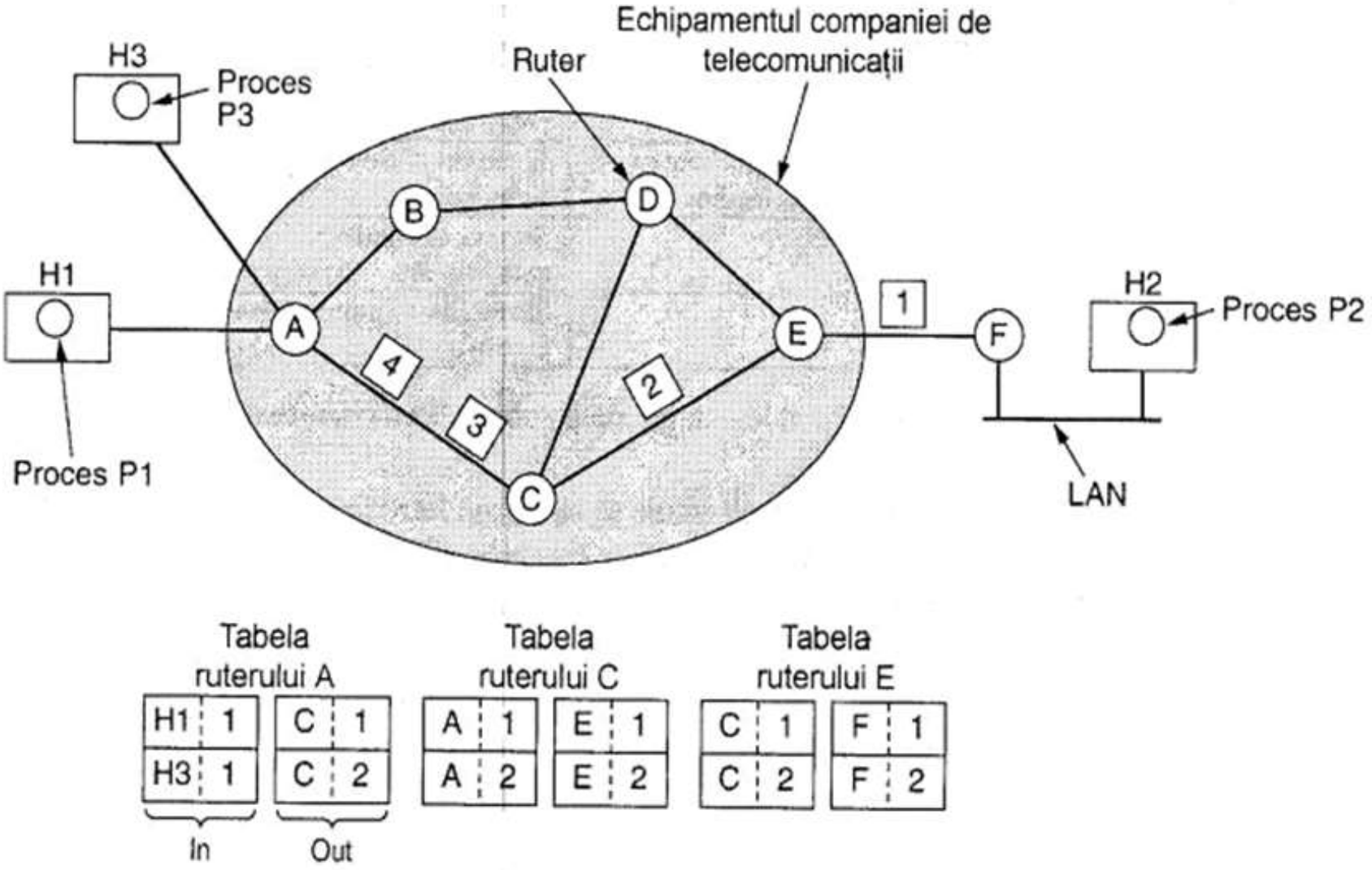

Fig. 3. Dirijare în cadrul unei subrețele cu circuite virtuale

Nivelul rețea

1. Cerinţele de proiectare ale nivelului reţea

1.1 Comutare de pachete de tip Memorează-şi-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 Comparaţie între subreţele cu circuite virtuale şi subreţele datagramă

2. Algoritmi de dirijare

2.1 Principiul optimalităţii

2.2 Dirijarea pe calea cea mai scurtă

1.5 Comparație între subrețele cu circuite virtuale şi subrețele datagramă

- Atât circuitele virtuale cât şi datagramele au suporteri şi oponenţi.
- Vom încerca acum să rezumăm argumentele ambelor tabere.
- Principalele aspecte sunt prezentate în fig.4, deşi cei extrem de riguroşi ar putea probabil găsi un contraexemplu pentru toate cele descrise în această figură.

Problemă	Subreţea datagramă	Subreţea cu circuite virtuale (CV)
Stabilirea circuitului	Nu este necesară	Obligatorie
Adresare	Fiecare pachet conţine adresa completă pentru sursă şi destinaţie	Fiecare pachet conţine un număr mic de CV
Informaţii de stare	routerele nu păstrează informaţii despre conexiuni	Fiecare CV necesită spaţiu pentru tabela routerului per conexiune
Dirijare	Fiecare pachet este dirijat independent	Calea este stabilită la iniţierea CV; toate pachetele o urmează
Efectul defectării routerului	Nici unul, cu excepţia 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 in avans suficiente resurse

1.5 Comparație între subrețele cu circuite virtuale şi subrețele datagramă

- În interiorul subreţelei există situaţii în care trebuie să se aleagă între facilităţi antagoniste specifice fie *circuitelor virtuale*, fie *datagramelor*.

Un astfel de compromis este acela între spaţiul de memorie al routerului şi lăţimea de bandă.

- *Circuitele virtuale permit pachetelor să conţină numere de circuite în locul unor adrese complete*.
- Dacă pachetul tinde să fie foarte mic, atunci existenţa unei adrese complete în fiecare pachet poate reprezenta o supraîncărcare (overhead) importantă şi deci o irosire a lăţimii de bandă.
- Preţul plătit pentru folosirea internă a circuitelor virtuale este spaţiul necesar păstrării tabelei în router.
- Soluţia mai ieftină este determinată de *raportul între costul circuitelor de comunicaţie şi cel al memoriei routerului*.

1.5 Comparaţie între subreţele cu circuite virtuale şi subreţele datagramă

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

- Folosirea circuitelor virtuale presupune existenţa unei faze iniţiale de stabilire a căii, care cere timp şi consumă resurse.
- Oricum, este uşor să ne imaginăm ce se întâmplă cu un pachet de date într-o subreţea bazată pe circuite virtuale:
  - *routerul foloseşte numărul circuitului ca un index într-o tabelă pentru a afla unde merge pachetul*
- Intr-o reţea bazată pe datagrame, pentru a găsi intrarea corespunzătoare destinaţiei *se foloseşte o procedură de căutare mult mai complicată*.

- O altă problemă este cea a ***dimensiunii spaţiului necesar pentru tabela din memoria router-ului***.
- O subreţea datagramă necesită o intrare pentru fiecare destinaţie posibilă, în timp ce o subreţea cu circuite virtuale necesită o intrare pentru fiecare circuit virtual.
- Totuşi, acest avantaj este relativ iluzoriu deoarece şi pachetele de iniţializare a conexiunii trebuie rutate, iar ele folosesc adresele destinaţie, la fel ca şi datagramele.

1.5 Comparație între subrețele cu circuite virtuale şi subrețele datagramă

- Circuitele virtuale au unele **avantaje** în *garantarea calităţii serviciului şi evitarea congestionării subreţelei*, deoarece resursele (de exemplu zone tampon, lărgime de bandă şi cicluri CPU) pot fi rezervate în avans, atunci când se stabileşte conexiunea.
- La sosirea pachetelor, *lăţimea de bandă necesară şi capacitatea routerului vor fi deja pregătite*.
- Pentru o subreţea bazată pe datagrame, evitarea congestionării este mult mai dificilă.

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

1.5 Comparație între subrețele cu circuite virtuale şi subrețele datagramă

Circuitele virtuale au o problemă de vulnerabilitate:

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

- Pierderea liniei de comunicaţie este fatală pentru circuitele virtuale care o folosesc, însă poate fi uşor compensată dacă se folosesc datagrame.

- De asemenea, datagramele permit routerului să echilibreze traficul prin subreţea, deoarece căile pot fi modificate parţial în cursul unei secvenţe lungi de pachete transmise.

Nivelul rețea

1. Cerinţele de proiectare ale nivelului reţea

1.1 Comutare de pachete de tip Memorează-şi-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 Comparaţie între subreţele cu circuite virtuale şi subreţele datagramă

2. Algoritmi de dirijare

2.1 Principiul optimalităţii

2.2 Dirijarea pe calea cea mai scurtă

2. Algoritmi de dirijare

- *Principala funcţie a nivelului reţea este dirijarea pachetelor de la maşina sursă către maşina destinaţie*, astfel incat în majoritatea subreţelelor pachetele vor face salturi multiple pentru a ajunge la destinaţie.
- *Algoritmii care aleg calea şi structurile de date folosite* de aceştia reprezintă un domeniu important al proiectării nivelului reţea.
- ***Algoritmul de dirijare*** ***(routing algorithm)*** este acea parte a software-ului nivelului reţea care răspunde de alegerea liniei de ieşire pe care trebuie trimis un pachet recepţionat.

2. Algoritmi de dirijare

- Dacă subreţeaua foloseşte intern datagrame, această decizie trebuie luată din nou pentru fiecare pachet recepţionat, deoarece este posibil ca cea mai bună rută să se fi modificat între timp.
- Dacă subreţeaua foloseşte circuite virtuale, deciziile de dirijare sunt luate doar la iniţializarea unui nou circuit virtual.
- După aceea pachetele de date vor urma doar calea stabilită anterior.
- Acest ultim caz este numit uneori **dirijare de sesiune** (session routing), deoarece calea rămâne în funcţiune pentru o întreagă sesiune utilizator (de exemplu o sesiune de conectare de la un terminal – login - sau un transfer de fişiere).

2. Algoritmi de dirijare

- Uneori este util să se facă distincţia între **dirijare**, care înseamnă *alegerea căii care va fi folosită*, şi **retransmitere**, care *se referă la ceea ce se întâmplă atunci când soseşte un pachet*.
- Se poate spune despre un router că rulează intern două procese:
  1. *Unul dintre ele preia fiecare pachet care soseşte, căutând în tabela de dirijare linia de ieşire folosită pentru el.* Acesta este ***procesul de retransmitere*** (forwarding)
  2. *Celălalt proces se ocupă de completarea şi actualizarea tabelei de rutare.* Aici intervine ***algoritmul de dirijare***.
- Indiferent dacă ruta se alege independent pentru fiecare pachet sau doar la stabilirea unei noi conexiuni, un algoritm de dirijare trebuie să aibă anumite proprietăţi: corectitudine, simplitate, robusteţe, stabilitate, echitate, optimalitate.

2. Algoritmi de dirijare

Algoritmii de dirijare pot fi grupaţi în două mari clase:

1. **Algoritmi neadaptivi**
2. **Algoritmi adaptivi**

2. Algoritmi de dirijare

1. Algoritmii neadaptivi (nonadaptive algorithms) *nu îşi bazează deciziile de dirijare pe măsurători sau estimări ale traficului şi topologiei curente*.

- Astfel, alegerea căii folosite pentru a ajunge de la nodul I la nodul J (oricare ar fi I şi J) se calculează în avans, off-line şi parvine routerului la iniţializarea reţelei
- Această procedură se mai numeşte şi **dirijare statică** (static routing)

2. Algoritmii adaptivi (adaptive algorithms), prin contrast, *îşi modifică deciziile de dirijare pentru a reflecta modificările de topologie şi de multe ori şi pe cele de trafic*.

Algoritmii adaptivi diferă:

- prin locul de unde îşi iau informaţia (de exemplu local, de la un router vecin sau de la toate routerele)
- prin momentul la care schimbă rutele (de exemplu la fiecare ΔT secunde, când se schimbă încărcarea sau când se schimbă topologia)
- şi prin metrica folosită pentru optimizare (de exemplu distanţa, numărul de salturi sau timpul estimat pentru tranzit)

Nivelul rețea

1. Cerinţele de proiectare ale nivelului reţea

1.1 Comutare de pachete de tip Memorează-şi-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 Comparaţie între subreţele cu circuite virtuale şi subreţele datagramă

2. Algoritmi de dirijare

2.1 Principiul optimalităţii

2.2 Dirijarea pe calea cea mai scurtă

2.1 Principiul optimalității

Principiul optimalităţii (optimaly principle) stabileşte că *dacă routerul J este pe calea optimă de la routerul I către routerul K, atunci calea optimă de la J la K este pe aceeaşi rută*.

- Pentru a vedea aceasta, să notăm cu r1, partea din cale de la I la J, iar cu r2 restul rutei.
- Dacă ar exista o rută mai bună decât r2 de la J la K, ea ar putea fi concatenată cu r1 şi ar îmbunătăţi ruta de la I la K, ceea ce ar contrazice presupunerea că r1r2 este optimală.

Nivelul rețea

1. Cerinţele de proiectare ale nivelului reţea

1.1 Comutare de pachete de tip Memorează-şi-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 Comparaţie între subreţele cu circuite virtuale şi subreţele datagramă

2. Algoritmi de dirijare

2.1 Principiul optimalităţii

2.2 Dirijarea pe calea cea mai scurtă

2.2. Dirijarea pe calea cea mai scurtă

Ideea este de a construi un graf al subreţelei:

- fiecare nod al grafului fiind un router
- iar fiecare arc al grafului fiind o linie de comunicaţie (numită adesea legătură)

Pentru a alege o cale între o pereche dată de routere, *algoritmul trebuie să găsească în graf calea cea mai scurtă dintre ele.*

Algoritmul lui Dijkstra

Algoritmul de determinare a celei mai scurte cai (shortest path routing) este propus de Dijkstra.

- Fiecare nod este etichetat (în paranteze) cu distanţa de la nodul sursă până la el, de-a lungul celei mai bune căi cunoscute.
- Iniţial nu se cunoaşte nici o cale, aşa că toate nodurile vor fi etichetate cu infinit.
- Pe măsură ce se execută algoritmul şi se găsesc noi căi, etichetele se pot schimba, reflectând căi mai bune.
- O etichetă poate fi fie temporară, fie permanentă.
- Iniţial toate etichetele sunt temporare.
- Atunci când se descoperă că o etichetă reprezintă cea mai scurtă cale posibilă de la sursă către acel nod, ea devine permanentă şi nu se mai schimbă ulterior.

Algoritmul lui Dijkstra

- Pentru a ilustra cum funcţionează algoritmul de etichetare, să ne uităm la graful neorientat, etichetat din fig. 5(a), unde etichetele reprezintă, de exemplu, distanţa.
- Dorim să aflăm cea mai scurtă cale de la *A* la *D.*
- Începem prin a marca nodul *A* ca permanent, indicând aceasta printr-un cerc colorat.
- Apoi vom examina fiecare nod adiacent cu *A* (care este acum nodul curent), reetichetând fiecare nod cu distanţa până la nodul A.
- De fiecare dată când un nod este reetichetat, îl vom eticheta şi cu nodul de la care s-a făcut încercarea, pentru a putea reface calea ulterior.
- După ce am examinat toate nodurile adiacente ale lui *A,* vom examina toate nodurile cu etichetă temporară din întregul graf şi îl facem permanent pe cel cu eticheta minimă, aşa cum se observă din fig. 5(b).
- Acest nod devine noul nod curent.

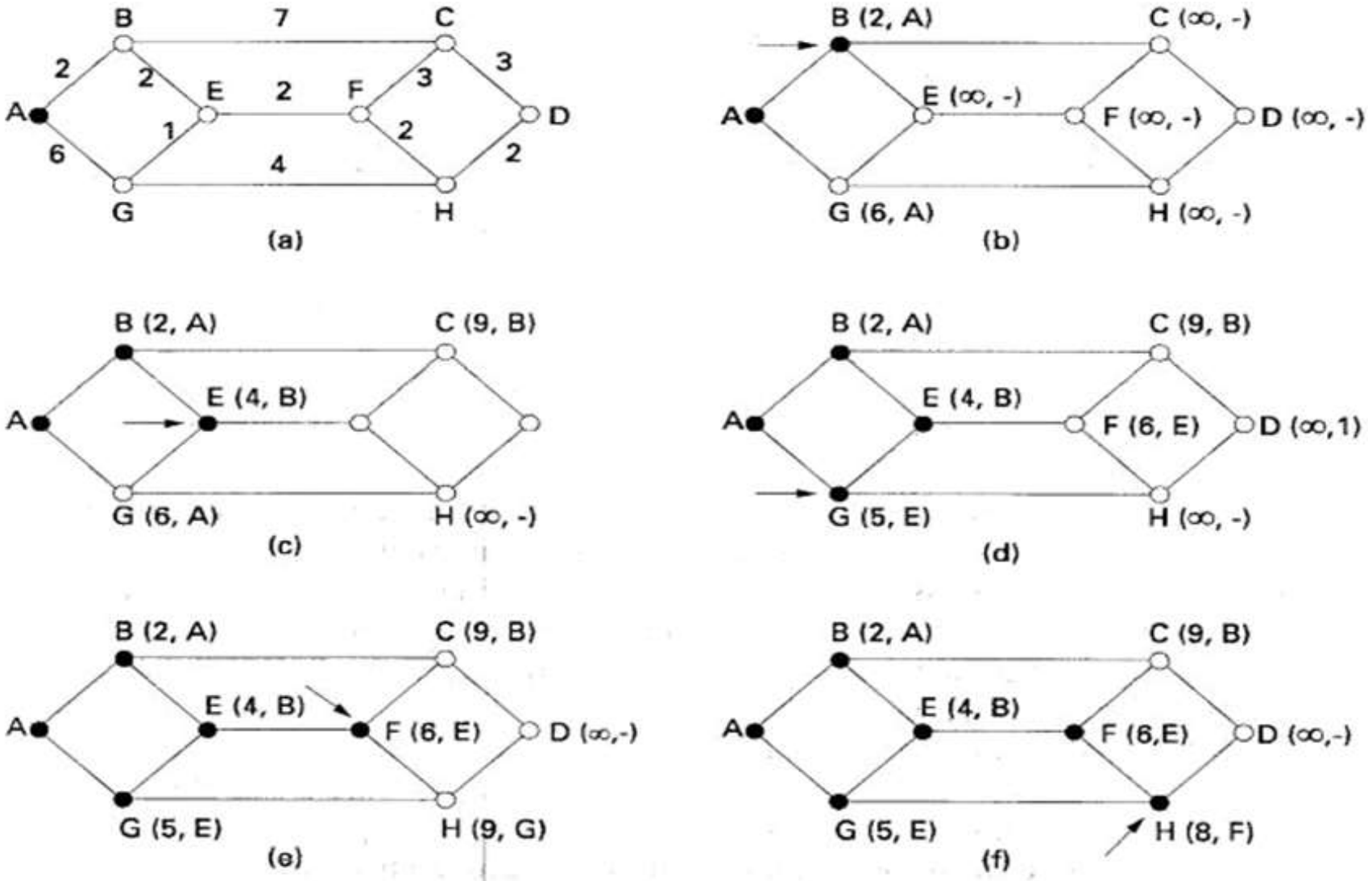

Fig. 5 Primii cinci pași folosiți în calcularea celei mai scurte căi de la A la D. Săgețile indică nodul curent.

Algoritmul lui Dijkstra

- Acum începem din B şi examinăm toate nodurile sale adiacente.
- *Dacă suma între eticheta lui B şi distanţa de la B la nodul considerat este mai mică decât eticheta acelui nod*, înseamnă că am găsit o cale mai scurtă şi va trebui făcută reetichetarea nodului.
- După ce toate nodurile adiacente nodului curent au fost inspectate şi au fost schimbate toate etichetele temporare posibile, se reia căutarea în întregul graf pentru a identifica nodul cu eticheta temporară minimă.
- Acest nod este făcut permanent şi devine nodul curent al etapei următoare.
- Fig. 5 prezintă primii cinci paşi ai algoritmului.

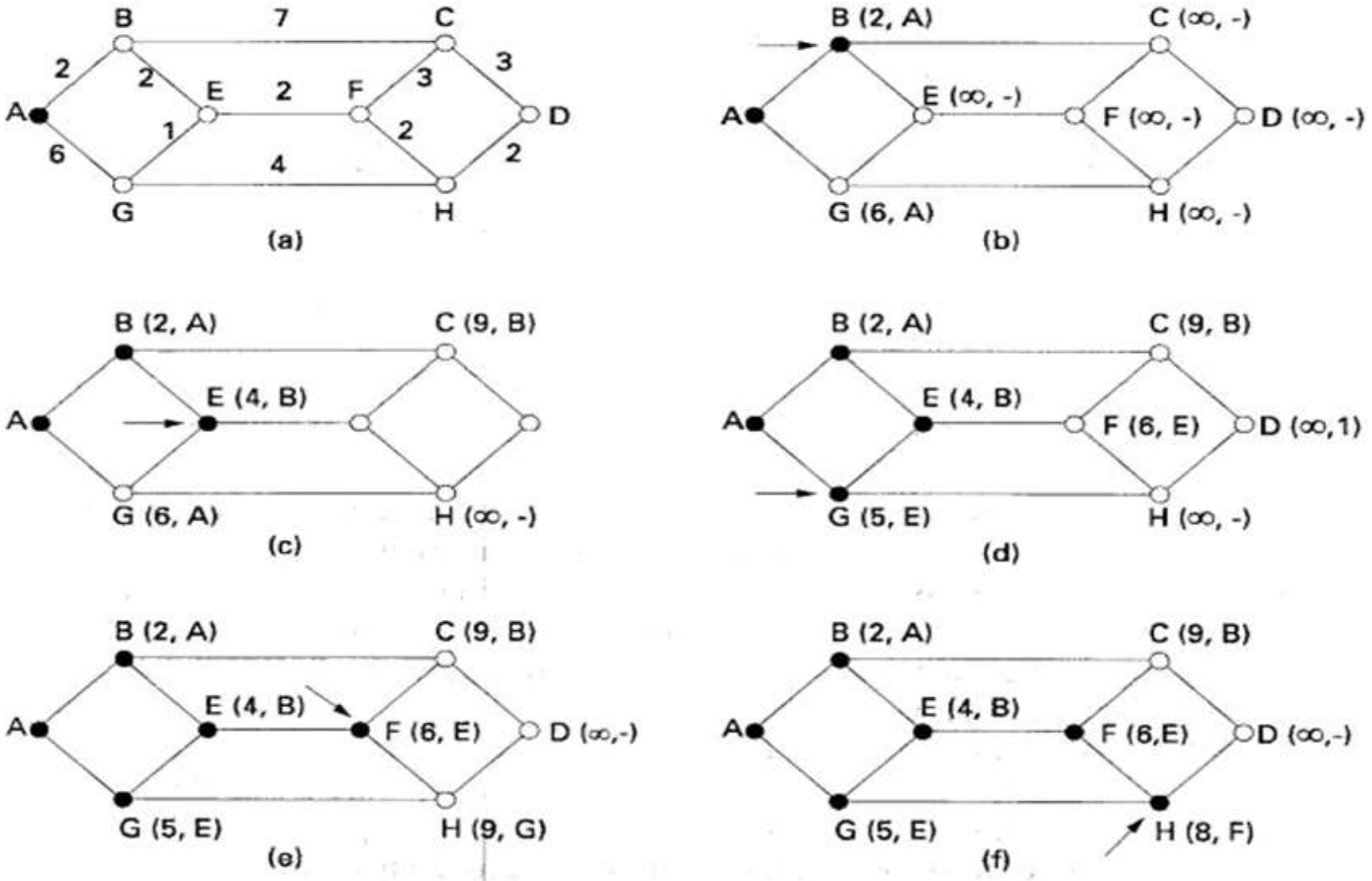

Fig. 5 Primii cinci pași folosiți în calcularea celei mai scurte căi de la A la D. Săgețile indică nodul curent.

- Pentru a vedea de ce merge algoritmul, să privim fig. 5(c).
- La momentul respectiv de abia am făcut permanent nodul E.
- Să presupunem că ar exista o cale mai scurtă decât ABE de exemplu AXYZE.
- Există două posibilităţi:
  - fie nodul 2 a fost deja făcut permanent
  - fie încă nu a fost

  Dacă a fost, atunci E a fost deja examinat (la pasul imediat următor celui la care Z a fost făcut permanent), astfel încât calea, AXYZE nu a fost ignorată şi deci nu poate fi cea mai scurtă cale.

- Să considerăm acum cazul în care Z este încă etichetă temporară.
- Atunci fie eticheta lui Z este mai mare sau egală cu cea a lui E, caz în care ABE nu poate fi o cale mai scurtă decât AXYZE, fie este mai mică decât cea a lui E, caz în care Z şi nu E va deveni permanent mai întâi, permiţând lui E să fie examinat din Z.

Algoritmul lui Dijkstra

Algoritmul este prezentat mai jos:

- Variabilele globale n şi dist sunt iniţializate înainte să fie apelată **shortest_path**.
- Singura diferenţă între program şi algoritmul descris mai sus este aceea că, acum *calculăm calea cea mai scurtă pornind de la nodul terminal, t, în locul nodului sursă, s*.

- Deoarece *calea cea mai scurtă de la t la s într-un graf neorientat este exact aceeaşi cu calea cea mai scurtă de la s la t*, nu contează la care capăt începem (decât dacă există mai multe căi scurte, caz în care, inversând calea, am putea găsi alt drum).
- Motivul pentru care începem căutarea de la nodul destinaţie este acela că nodurile sunt etichetate cu predecesorul şi nu cu succesorul.
- Atunci când calea finală este copiată în variabila de ieşire, **path**, calea este inversată.
- Prin inversarea căutării, cele două efecte se anulează, astfel încât rezultatul se obţine în ordinea corectă.

Algoritmul lui Dijkstra (I)

```
#define max_nodes 1024        /* numarul maxim de noduri */
#define infinity 1000000000   /* un numar mai mare decat orice
    cale */
int n, dist[max_nodes][max_nodes]      /* dist[i][j] e
    distanta de la i la j */
void shortest_path(int s,int t, int path[])
{
     struct state{                /*calea cu care se lucreaza */
        int predecessor;     /* nodul anterior */
        int length;          /* lungimea de la sursa la acest nod */
enum{permanent, tentative} label;     /* eticheta stare */
} state [max_nodes];
```

Algoritmul lui Dijkstra (II)

```
int i, k, min;
struct state *p;
for( p=&state[0]; p<&state[n]; p++)
{                              /* initializari */
        p->predecessor = -1;
        p->length = infinity;
        p->label = tentative;
}
```

Algoritmul lui Dijkstra (III)

```
state[t].length=0;
state[t].label=permanent;
k = t;                                  /* k este nodul initial de lucru */
do{                                     /* exista vreo cale mai buna de la k? */
   for(i=0; i<n; i++)                                  /* graful are noduri */
        if(dist[k][i] != 0 && state[i].label = = tentative)
         {
           if(state[k].length + dist[k][i] < state[i].length)
            {
                state[i].predecessor = k;
                state[i].length = state[k].length + dist[k][i];
            }
        }
```

Algoritmul lui Dijkstra (IV)

```
/* gaseste nodul etichetat temporar cu cea mai mica eticheta */
    k = 0;
    min = infinity;
    for(i=0; i<n; i++)
      if(state[i].label == tentative && state[i].length < min)
            {
                  min = state[i].length;
                  k = i;
             }
    state[k].label = permanent;
 } while(k!=s);
```

Algoritmul lui Dijkstra (V)

```
/* copiaza calea in vectorul de iesire */
    i=0;
    k=s;
    do{
        path[i++] = k;
        k = state[k].predecessor;
    }while (k >= 0);
}
```

2.3. Dirijare cu vectori distanță

Reţelele moderne de calculatoare folosesc de obicei algoritmi dinamici de dirijare în locul celor statici, descrişi anterior, deoarece algoritmii statici nu ţin seama de încărcarea curentă a reţelei:

- **Algoritmul de dirijare cu vectori distanţă** (*distance vector routing*) presupune că fiecare router menţine o tabelă (de exemplu un vector) care păstrează cea mai bună distanţă cunoscută spre fiecare destinaţie şi linia care trebuie urmată pentru a ajunge acolo.

Aceste tabele sunt actualizate prin schimbul de informaţii între nodurile vecine.

- **Algoritmul de dirijare cu vectori distanţă** este cunoscut şi sub alte nume, cel mai des **algoritmul distribuit de dirijare Bellman-Ford** şi **algoritmul Ford-Fulkerson.**

Întrebări?