



# Nivelul retea



# Cuprins

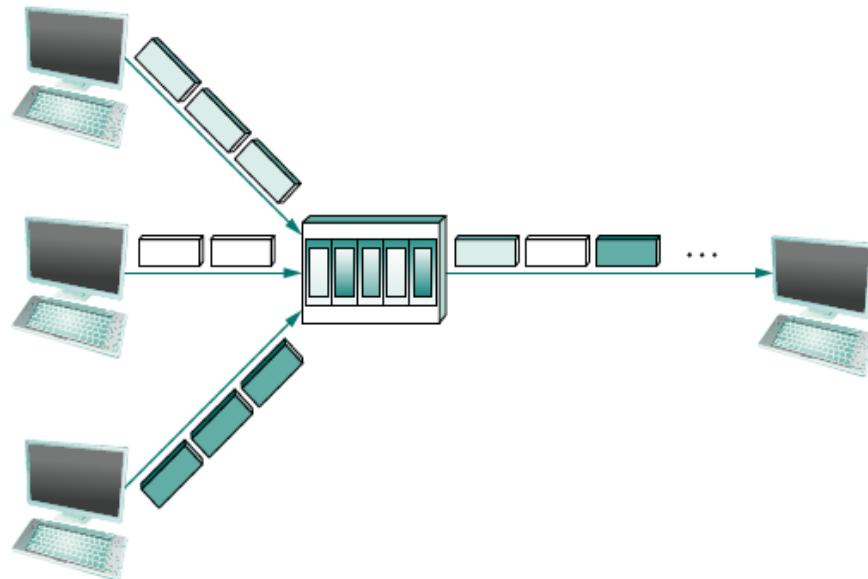
- De ce pachete ?
- Organizarea internă
- Protocolul IP – adresare și retransmiterea pachetelor
- Algoritmi de dirijare
- Protocole de control
- Structura internetului
  - protocolul OSPF – Open Shortest Path First
  - protocolul BGP – Border Gateway Protocol
- Dirijarea în rețele ad hoc
- IPv6

# De ce este nevoie de pachete?

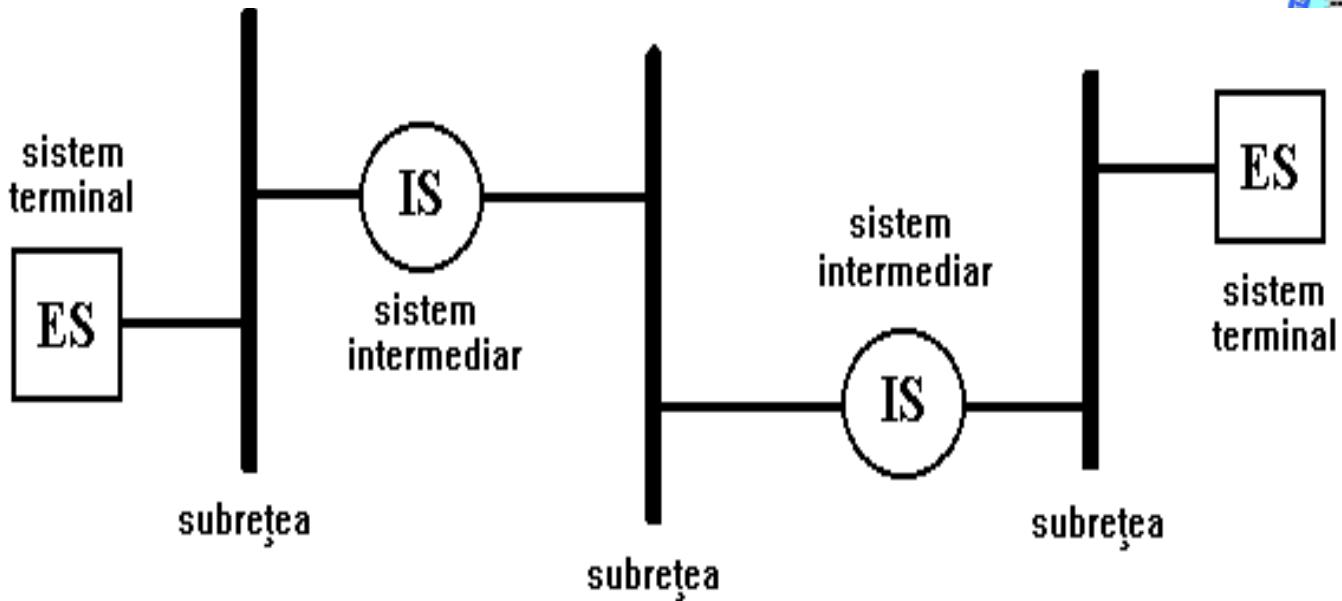
Legaturile unei rețele pot fi folosite simultan de transmisii paralele între mai multe **perechi de noduri**

**Multiplexare** – se aloca transmisiilor

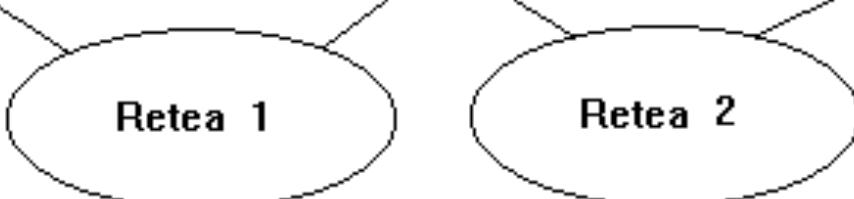
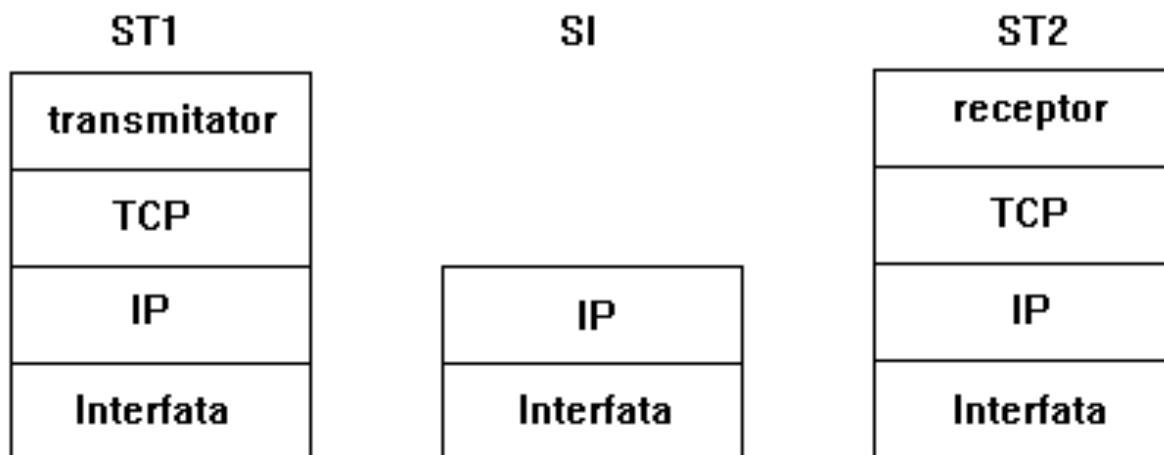
- sloturi de **timp** – STDM – Synchronous Time Division Multiplexing
- **subcanale** de frecvențe diferite – FDM – Frequency Division Multiplexing
- **multiplexare statistică** – STMD – sloturile sunt alocate **la cerere**;  
daca o singura pereche de noduri are de transmis, nu asteapta ptr. slot;
- pentru a evita **acapararea** legăturii, transmisia se face în **pachete cu dimensiune limitată**



## Modelul Internetului



Nivelele străbătute de pachete





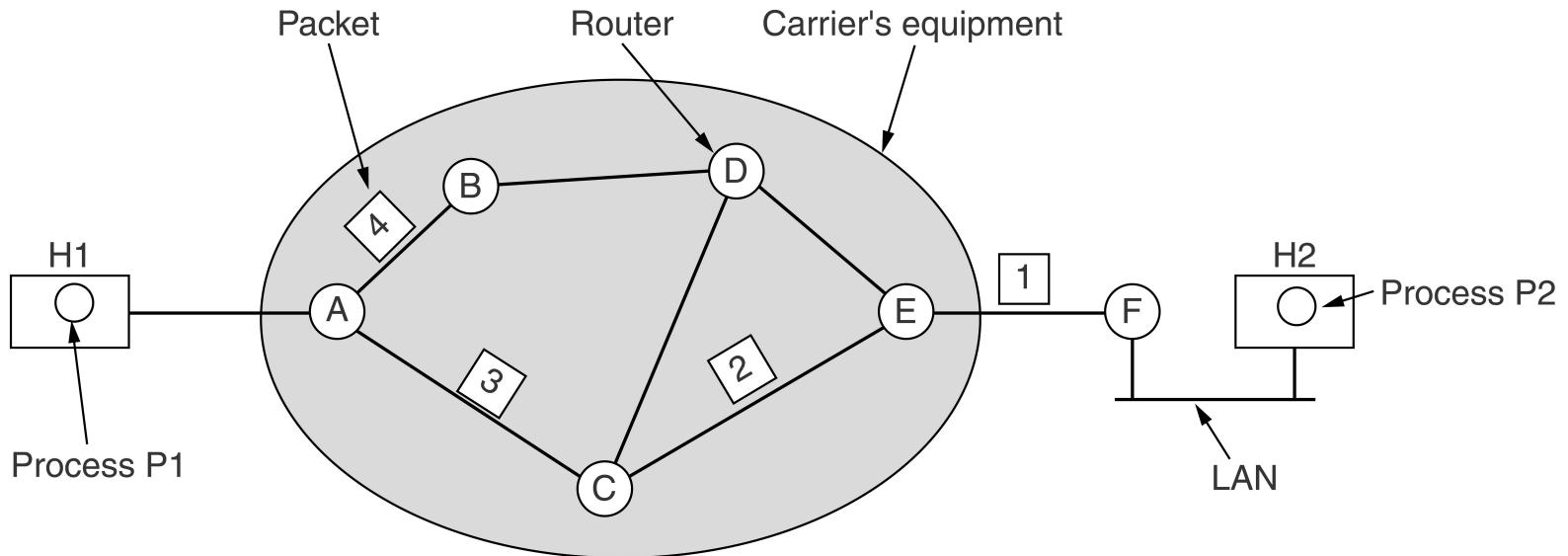
# Funcțiile nivelului rețea

- dirijarea pachetelor
- adresarea

## Aspecte principale

- servicii
  - ne-orientate pe conexiune
  - orientate pe conexiune
- organizarea internă corespunde tipurilor de servicii
  - datagrame
  - circuite virtuale

# Organizarea internă - datagrame



Folosita de  
pachetele  
1, 2 si 3

A's table

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

C's table

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

E's table

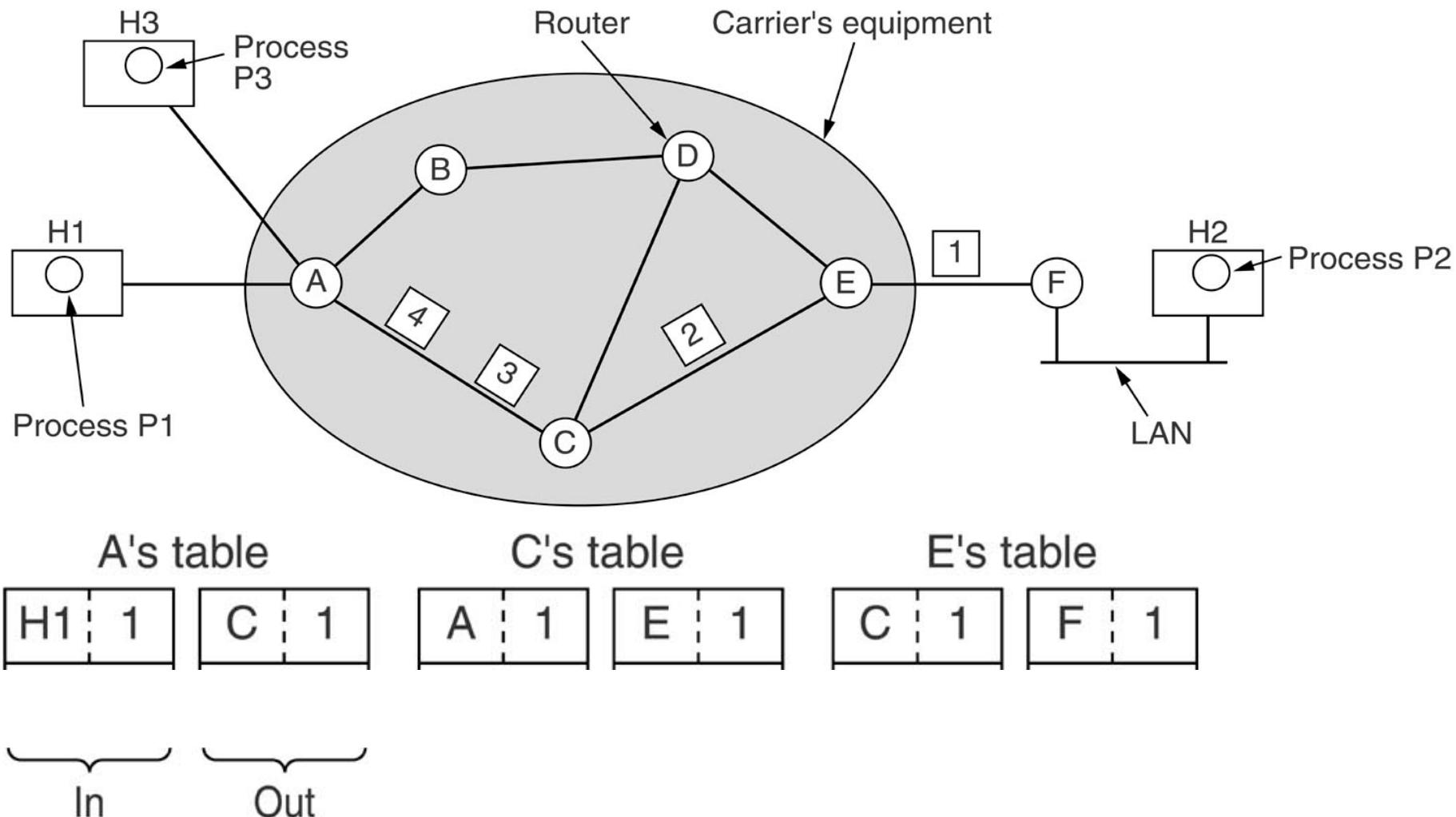
Folosita de  
pachetul 4



# Caracteristici datagrame

- Pachetul contine **toate** informațiile necesare ruterelor pentru “pasarea” lui către destinație (inclusiv adresele sursă și destinație)
- Organizare **fără conexiune**
  - Un calculator gazda (*host*) poate trimite pachetul **oricand și oriunde** în rețea
- Nu asigură **corectitudinea**
  - Transmisorul nu poate să știe dacă pachetul este livrat sau dacă destinatarul mai este conectat
- Nu pastrează **ordinea** pachetelor
  - Pachetele sunt dirijate independent unele de altele
- **Robusta**
  - La defectarea unei legături se gasesc rute alternative
- Utilizare largă în Internet

# Organizarea internă – circuit virtual



Nodul A re-numeroteaza circuitul virtual



# Caracteristici circuit virtual

- Organizare **bazata pe conexiune**
  - Transferul incepe după stabilirea conexiunii
  - Pachetul contine id-ul conexiunii
  - Se pot **aloca resurse** la stabilirea conexiunii (memorie tampon pentru pachete)
- Transmitatorul stie
  - ca există o conexiune
  - ca receptorul este **pregatit** să primească pachete
- Se pastrează **ordinea** pachetelor
- Se poate controla **fluxul**
  
- Folosit în
  - MPLS (MultiProtocol Label Switching)
  - retele virtuale private (VPN – Virtual Private Network)



# Protocolul IP

- Are
  - o schema de adresare care permite identificarea oricărui calculator din Internet
  - un model - **datagrama** - pentru transmiterea datelor de la un nod gazda la altul
    - **datagrama = pachet**
- Modelul de serviciu – **best effort**
  - rețeaua “face toate eforturile” să livreze pachetele la destinație
  - nu face nici o încercare să corecteze erorile

# Protocol IPv4 – formatul pachetului

SERVICE TYPE = precedence (3), delay, throughput, reliability, cost

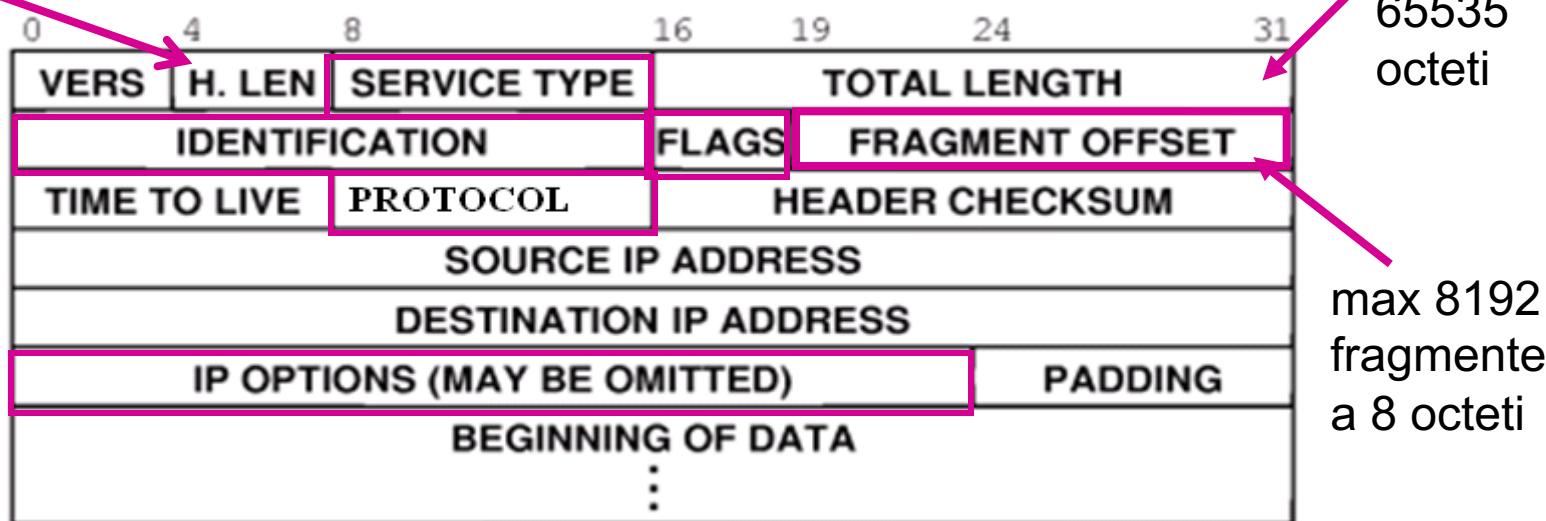
PROTOCOL = (TCP, UDP, etc.)

IDENTIFICATION Id datagrama de care aparține fragmentul

FRAGMENT OFFSET pozitia fragmentului in pachet

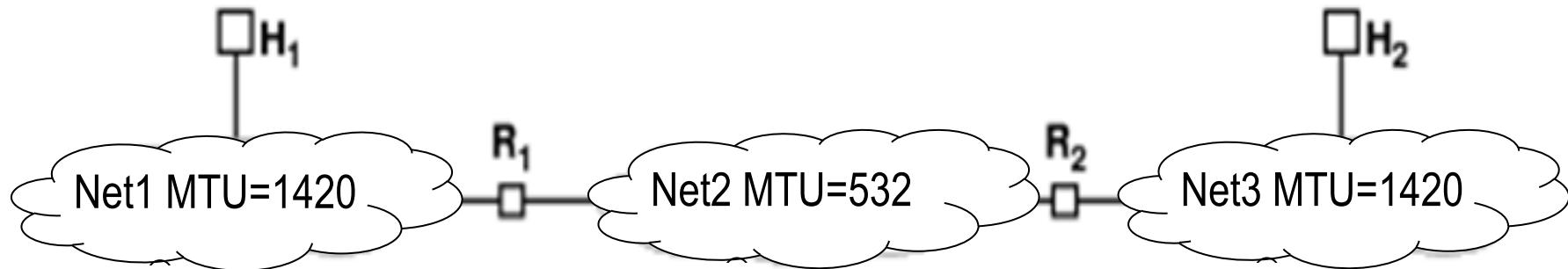
FLAGS **DF** = Don't Fragment / **MF** = More Fragments

numar  
cuvinte  
de 32  
biti

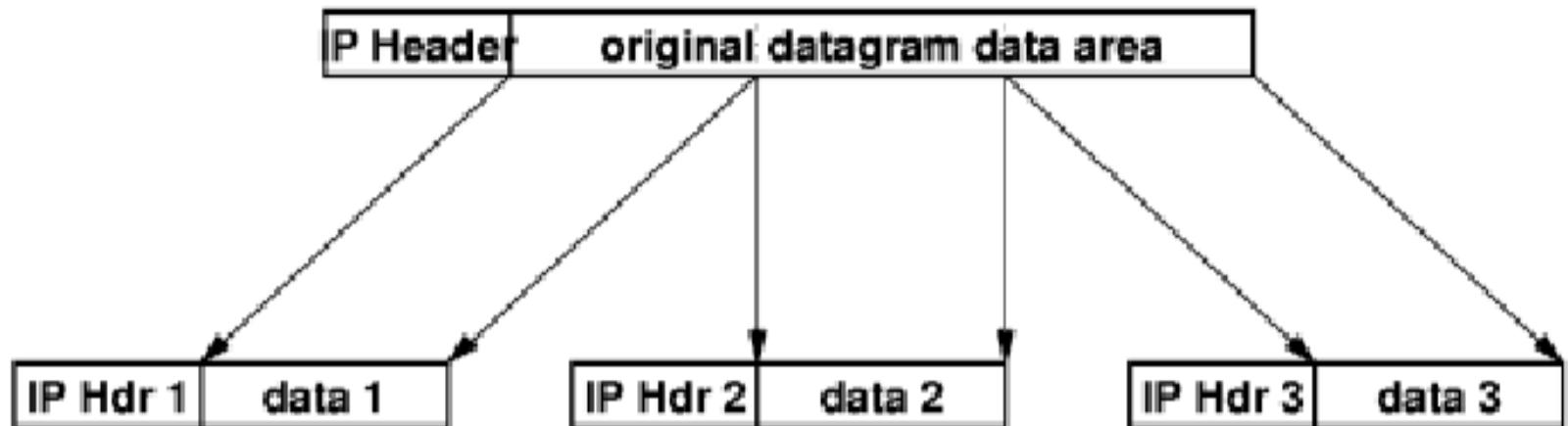


OPTIONS:  
Security  
Strict source routing  
Loose source routing  
Record route  
Timestamp

# MTU – Maximum Transmission Unit



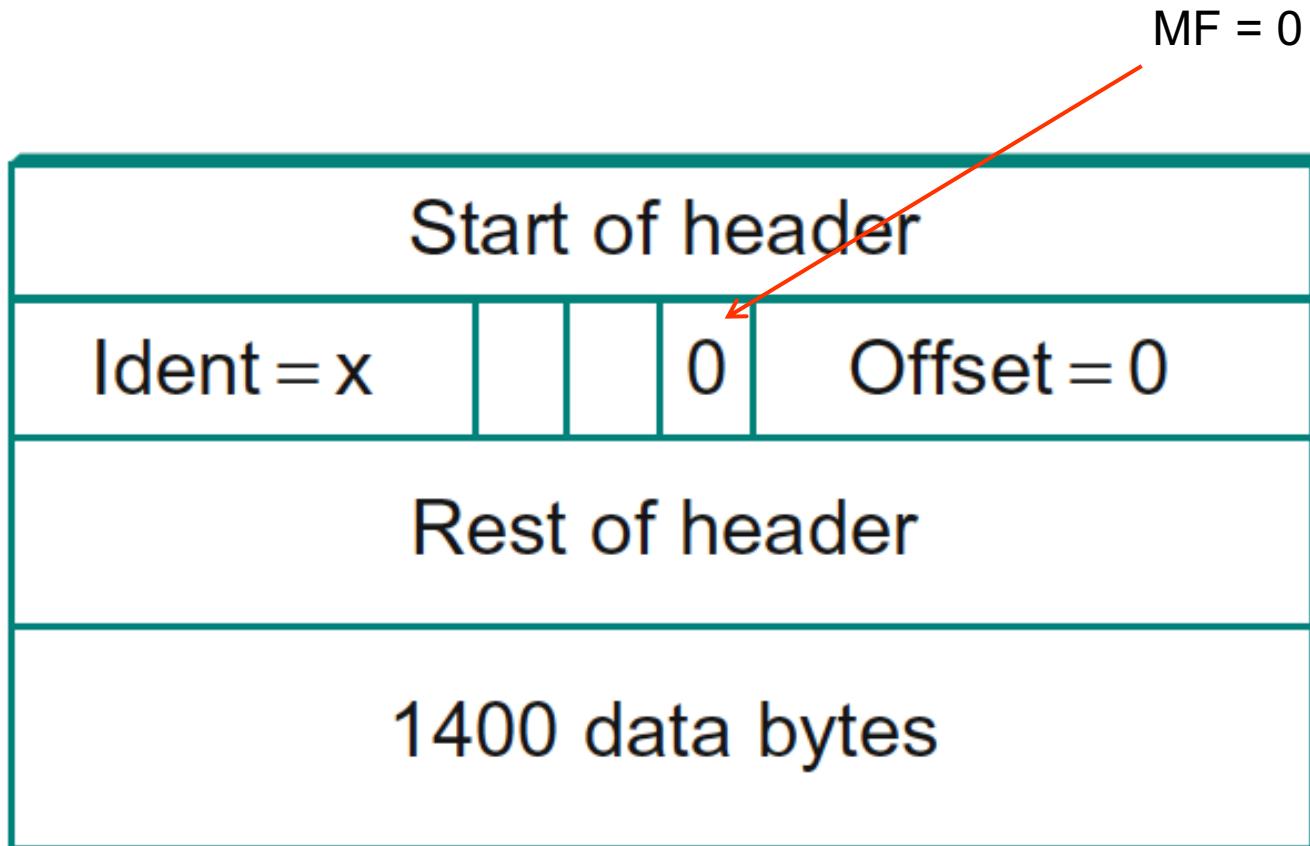
Fragmentarea se face la  $R_1$



Reasamblarea se face la  $H_2$  – de ce?

# Campurile de antet pentru fragmentare (1)

(a) pachet transmis ne-fragmentat prin Net1 cu MTU = 1420  
are: 1400 octeti de date  
20 octeti de antet IP



# Campurile de antet pentru fragmentare (2)

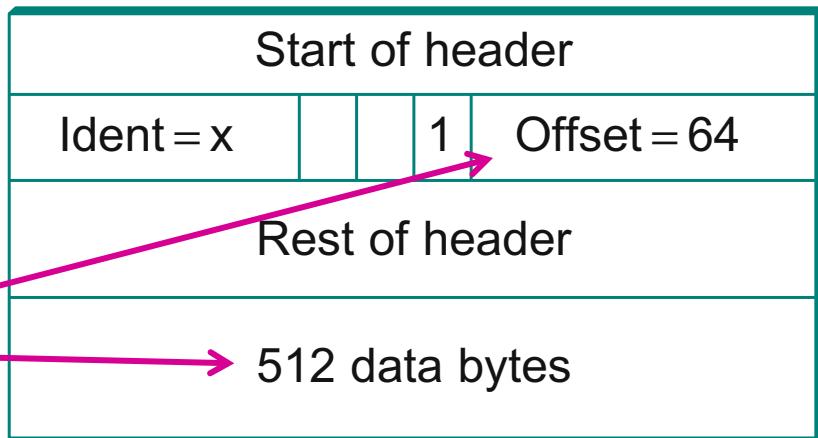
Net2 are MTU = 532

Ruterul R1 segmenteaza pachetul in 3 fragmente (fig b1-b3)

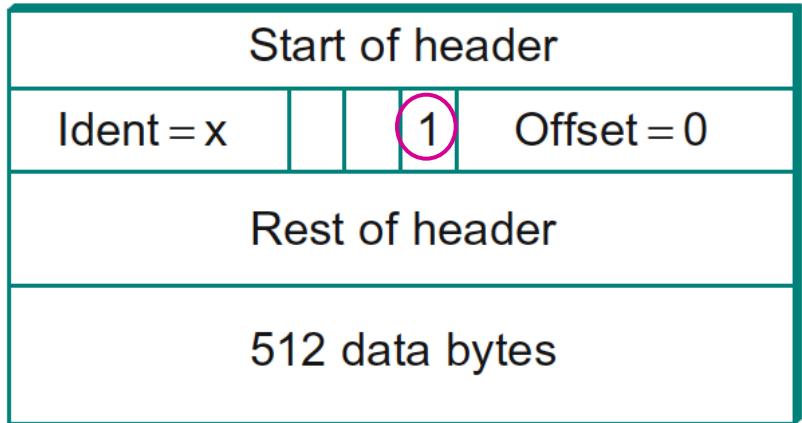
- lungimea fiecarui segment este multiplu de 8 octeti
- offset-ul numara grupuri de cate 8 octeti
- Offset = 64 = 512 / 8

unde e restul pana la 532 octeti?

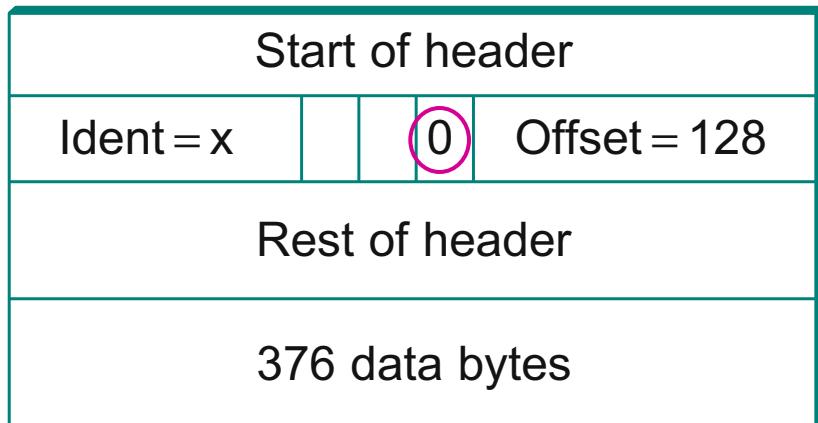
b2



b1



b3



# Clase de adrese IP

Fiecare nod (gazda/ruter) are o **adresa IP** asociată cu interfața lui de rețea

Un nod cu mai multe interfețe are mai multe adrese IP (ex. rutere)

0	7 8	hostid	31	clasa A
0	15 16	hostid	31	clasa B
10	netid	hostid	23 24	clasa C
110	netid	hostid	31	clasa D
1110	adresa multicast		31	clasa E
0			31	
11110	rezervat pentru utilizare viitoare		31	

Clasa de adrese	Biți în prefix	Număr maxim de rețele	Biți în sufix	Număr maxim de gazde per rețea
A	7	128	24	16777216
B	14	16384	16	65536
C	21	2097152	8	256



# Câteva adrese speciale

Prefix (rețea)	Suffix (gazdă)	Semnificație	Scop
toți 0	toți 0	acest calculator	Folosită cand nodul încă nu are o adresă
network	toți 0	network	Identifică rețeaua
network	toți 1	broadcast	broadcast în rețeaua specificată
toți 1	toți 1	broadcast	broadcast în rețeaua locală
127	orice	loopback	testare

## Notări pentru adrese

binară 11000010 00011000 00010001 00000100

zecimală 194.24.17.4



# Tabele de dirijare

Orice pachet conține o **adresa IP** a destinatarului, cu două părți

**<adresa\_retea, adresa\_nod>**

Un pachet este transmis de la sursă la destinație trecând prin noduri intermediare (**rutere**), fiecare legând între ele cel puțin două rețele

**Rol ruter** – primește un pachet și

- îl livrează **gazdei** de destinație (dacă este în aceeași rețea)
  - Toate nodurile care au aceeași **adresa\_retea** sunt situate în **aceeași retea fizică** și pot comunica direct prin **legătura de date** (transmit **cadre**)
- altfel, îl re-transmite (**forward**) către un alt nod **NextHop**
  - Folosește **tabela de dirijare (rutare)** care are intrări de forma

**<adresa\_retea, NextHop>**



# Algoritm de *forwarding* IP

Extrage **<adresa\_retea, adresa\_gazda>** destinație din datagrama

Caută o intrare cu **adresa\_retea** în tabela de dirijare

**if** **adresa\_retea** apare în tabela de dirijare

**if** **adresa\_retea** indică o retea direct conectată **then**

        transmite datagrama direct la **adresa\_gazda**

**else** transmite datagrama urmatorului ruter (**Next Hop**)

**else** transmite datagrama unui **ruter implicit**

Adresarea **ierarhica** **<adresa\_retea, adresa\_gazda>** reduce numărul de intrări în tabela de dirijare (o intrare pentru o **adresa\_retea**)

In practica, tabelele de rutare sunt separate pe clase de adrese

- Căutare prin: indexare (A și B) sau hashing (C)

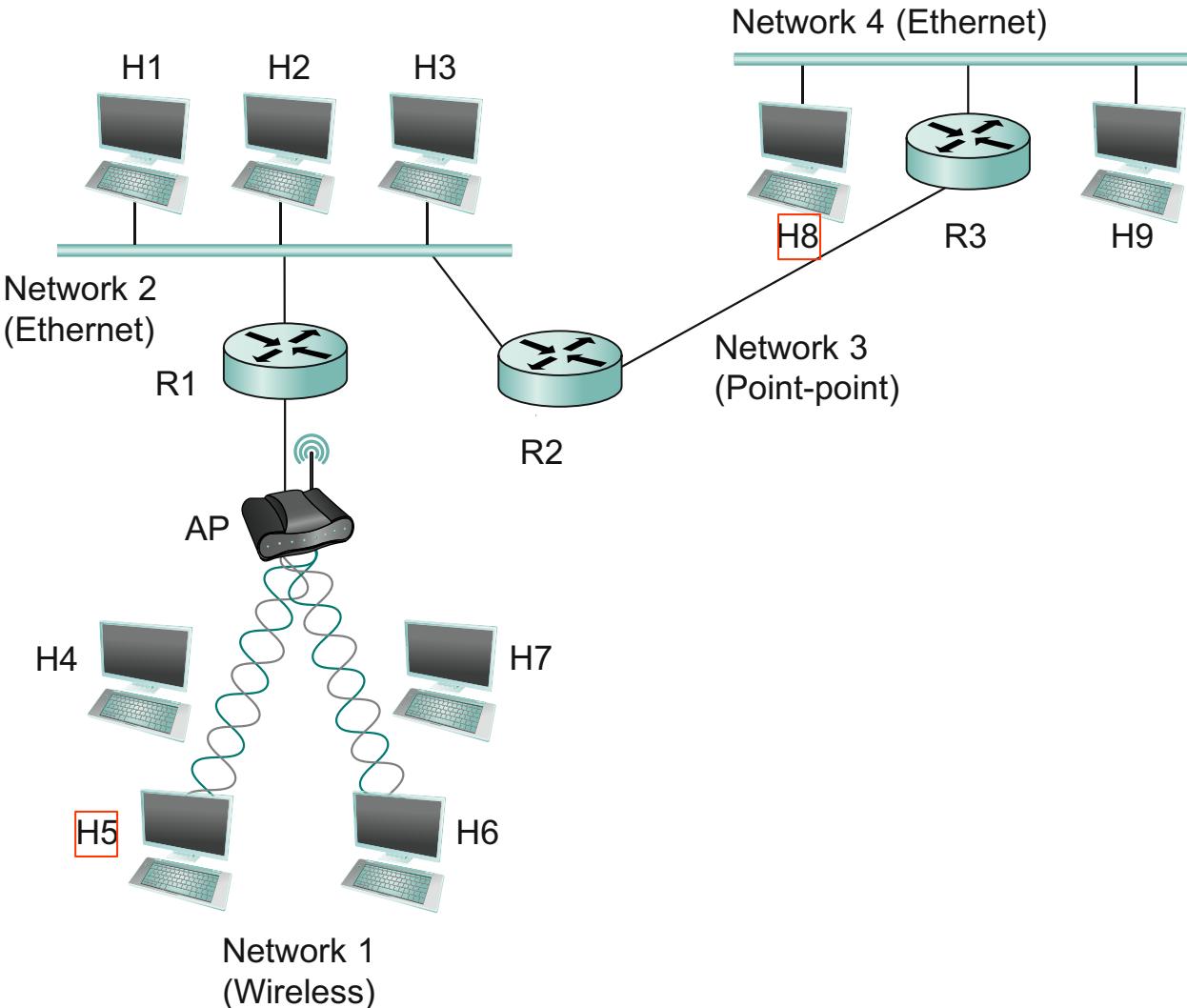
# Exemplu

Transferul unei datagrame între **H5** și **H8**

**H5** are legătura (printr-un punct de acces AP) la ruterul **R1**

Pachetul trece prin rutele **R1**, **R2** și **R3**

**R3** este în aceeași rețea cu **H8** și îl livrează direct datagrama





# Conversie adresa IP – adresa fizică

La livrarea directă (destinatar în aceeași rețea) se folosește **adresa fizică** a receptorului (ptr care se cunoaște adresa IP)

**Nu există** o legătură biunivocă între adresa fizică și adresa IP

De ex.

- adresa IP are 32 biti
- adresa Ethernet are 48 biti

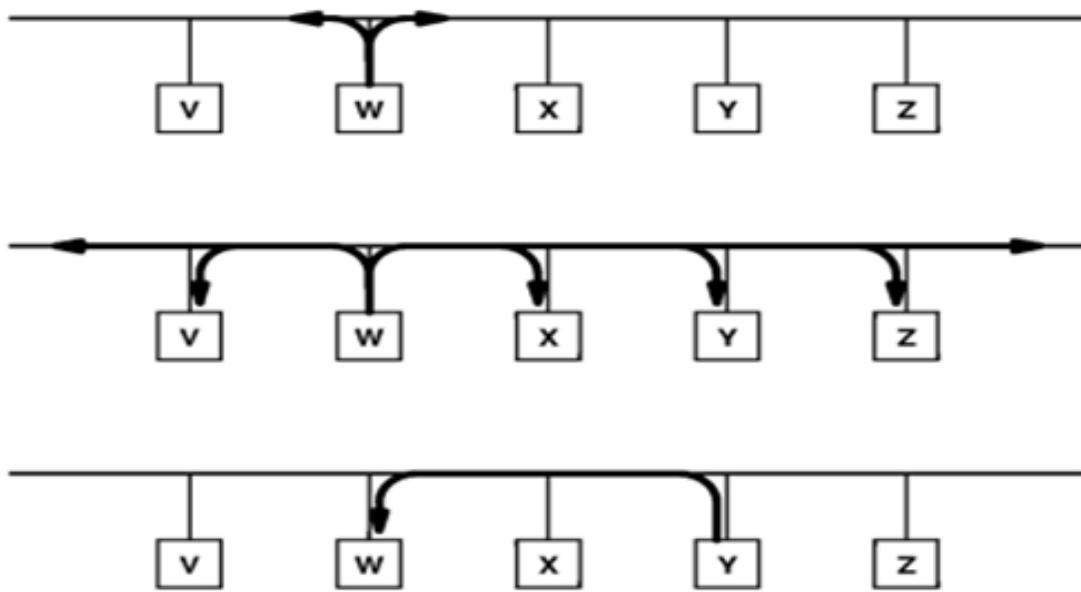
Pentru mapare se pot folosi

- tabele de corespondență
- formule de calcul
- schimb de mesaje
  - ex. ARP - Address Resolution Protocol
    - Face maparea între adresa de protocol și adresa hardware

# ARP - Address Resolution Protocol

Secvența de mesaje pentru a afla adresa fizică

- w difuzează o cerere ARP cu adresa IP cunoscută
- cererea este primită de toate gazdele din reteaua locală
- y – care are adresa IP respectivă – trimite ca răspuns adresa fizică (ex. Ethernet)



# Conversie adresa IP – adresa fizică

0	8	15	16	31		
Hardware Type		Protocol Type				
HLEN	PLEN	Operation				
Sender HA (octets 0-3)						
Sender HA (octets 4-5)		Sender IP (octets 0-1)				
Sender IP (octets 2-3)		Target HA (octets 0-1)				
Target HA (octets 2-5)						
Target IP (octets 0-3)						

Format mesaje ARP – conceput pentru diverse categorii de adrese

- **Hardware type:** 1 pentru Ethernet
- **Protocol type:** 0x0800 pentru IP (0000.1000.0000.0000)
- **HLEN** - Hardware len: 6 octeti pentru Ethernet
- **PLEN** - Protocol len: 4 octeti pentru IP
- **Operation:** 1=cerere, 2=raspuns



# Subretele

- Regula “o adresă pentru fiecare rețea fizică separată”
  - foloseste ineficient spațiul de adrese
    - o rețea de clasa C cu 2 noduri consumă doar 2 din totalul de 255 de adrese de nod
    - o rețea de clasa B cu peste 255 noduri ocupă peste 64000 de adrese indiferent dacă le folosește pe toate sau nu
- soluția 1 – subretele
  - de ex. o rețea clasa B este împărțită în mai multe subretele apropiate geografic
- soluția 2 - adrese fara clase



# CIDR – Classless InterDomain Routing

**Ideea:** alocă spațiul de adrese IP în blocuri de lungimi diferite

Notația specială pentru **adresa de rețea** CIDR

194.24.0.0/21 → din cei 32 de biți ai adresei IP

**adresa\_retea** ocupă **21** biți

**adresa\_gazda** ocupă **11** biti

University	First address	Last address	How many	Written as
Cambridge	194.24.0.0	194.24.7. <b>255</b>	2048	194.24.0.0/21
Edinburgh	194.24.8.0	194.24.11.255	1024	194.24.8.0/22
(Available)	194.24.12.0	194.24.15.255	1024	194.24.12/22
Oxford	194.24.16.0	194.24.31.255	4096	194.24.16.0/20



# CIDR – Exemplu

Pentru a afla **adresa\_retea** se folosesc măști

**Cambridge:** Adresă 11000010 00011000 00000000 00000000  
Mască 11111111 11111111 11111000 00000000

**Edinburgh:** Adresă 11000010 00011000 00001000 00000000  
Mască 11111111 11111111 11111100 00000000

**Oxford:** Adresă 11000010 00011000 00010000 00000000  
Mască 11111111 11111111 11110000 00000000

University	First address	Last address	How many	Written as
Cambridge	194.24.0.0	194.24.7. 255	2048	194.24.0.0/21
Edinburgh	194.24.8.0	194.24.11.255	1024	194.24.8.0/22
(Available)	194.24.12.0	194.24.15.255	1024	194.24.12/22
Oxford	194.24.16.0	194.24.31.255	4096	194.24.16.0/20

# CIDR – reguli de alocare a adreselor

University	First address	Last address	How many	Written as
Cambridge	194.24.0.0	194.24.7. 255	2048	194.24.0.0/21
Edinburgh	194.24.8.0	194.24.11.255	1024	194.24.8.0/22
(Available)	194.24.12.0	194.24.15.255	1024	194.24.12/22
Oxford	194.24.16.0	194.24.31.255	4096	194.24.16.0/20

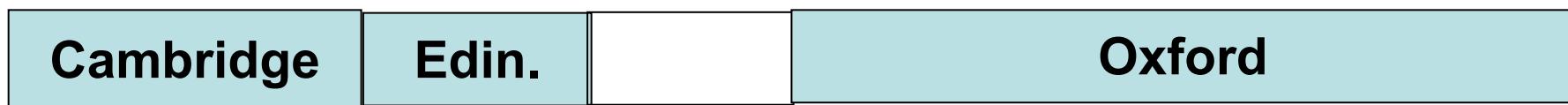
Reguli pentru a avea o mască pentru un bloc de adrese

→ lungimea blocului trebuie să fie o putere a lui 2

→ toate adresele din bloc au aceeași adresa\_retea → adresa de început a blocului de adrese trebuie să fie multiplu de dimensiunea acestuia

Ex.: zona de adrese pentru Oxford începe la o frontieră de 4096 octeți

0                    2048                    3072                    4096





# Algoritm *forwarding*

Intrare în tabela de rutare - ([adresa\\_retea](#), Masca, NextHop)

Algoritmul alege intrarea pentru care

$$(\text{Adresa\_IP AND Masca}) = \text{adresa\_reteea}$$

Ex. Sosește pachet cu adresa\_IP = [194.24.17.4](#)

compara cu [Cambridge /21](#) – adresa\_retea = 194.24.0.0

adresa\_retea: 11000010 00011000 00000000 00000000

Masca: 11111111 11111111 11111000 00000000

adresa\_IP: 11000010 00011000 00010001 00000100

Adresa\_IP AND Masca:

11000010 00011000 00010000 00000000

→ = 194.24.16.0 ≠ 194.24.0.0 → **nepotrivire**



## Algoritm *forwarding* (2)

Ex. Sosește pachet cu adresa\_IP = 194.24.17.4

Cambridge /21 – adresa\_rețea = 194.24.0.0

(Adresa\_IP AND Masca) = 194.24.16.0 → nepotrivire

Edinburgh /22 - adresa\_rețea = 194.24.8.0

(Adresa\_IP AND Masca) = 194.24.16.0 → nepotrivire

Oxford /20 - adresa\_rețea = 194.24.16.0

(Adresa\_IP AND Masca) = 194.24.16.0 → potrivire

Dacă nu sunt alte potriviri -> folosește intrarea pentru Oxford



# Potriviri multiple

Prefixe de lungimi diferite

→ unele **adrese IP** se pot potrivi cu mai multe **adrese\_retea** din tabela de dirijare

Ex.

adresa\_IP                    171.69.10.5 se potrivește cu

adresele de rețea        171.69.0.0/16

                                171.69.10.0/24

Regula: se alege potrivirea “mai lungă”



# Reducere dimensiune tabelă rutare

Soluție - agregarea unor adrese

Consideram adresele de retea:

C - Cambridge: 194.24.0.0/21

E - Edinburgh: 194.24.8.0/22

O - Oxford: 194.24.16.0/20

C: adresa\_retea 11000010 00011000 00000000 00000000

E: adresa\_retea 11000010 00011000 00001000 00000000

O: adresa\_retea 11000010 00011000 00010000 00000000

Presupunem: pentru rutare la C, E, O, nodul **NewYork** are în tabela de dirijare **același** NextHop

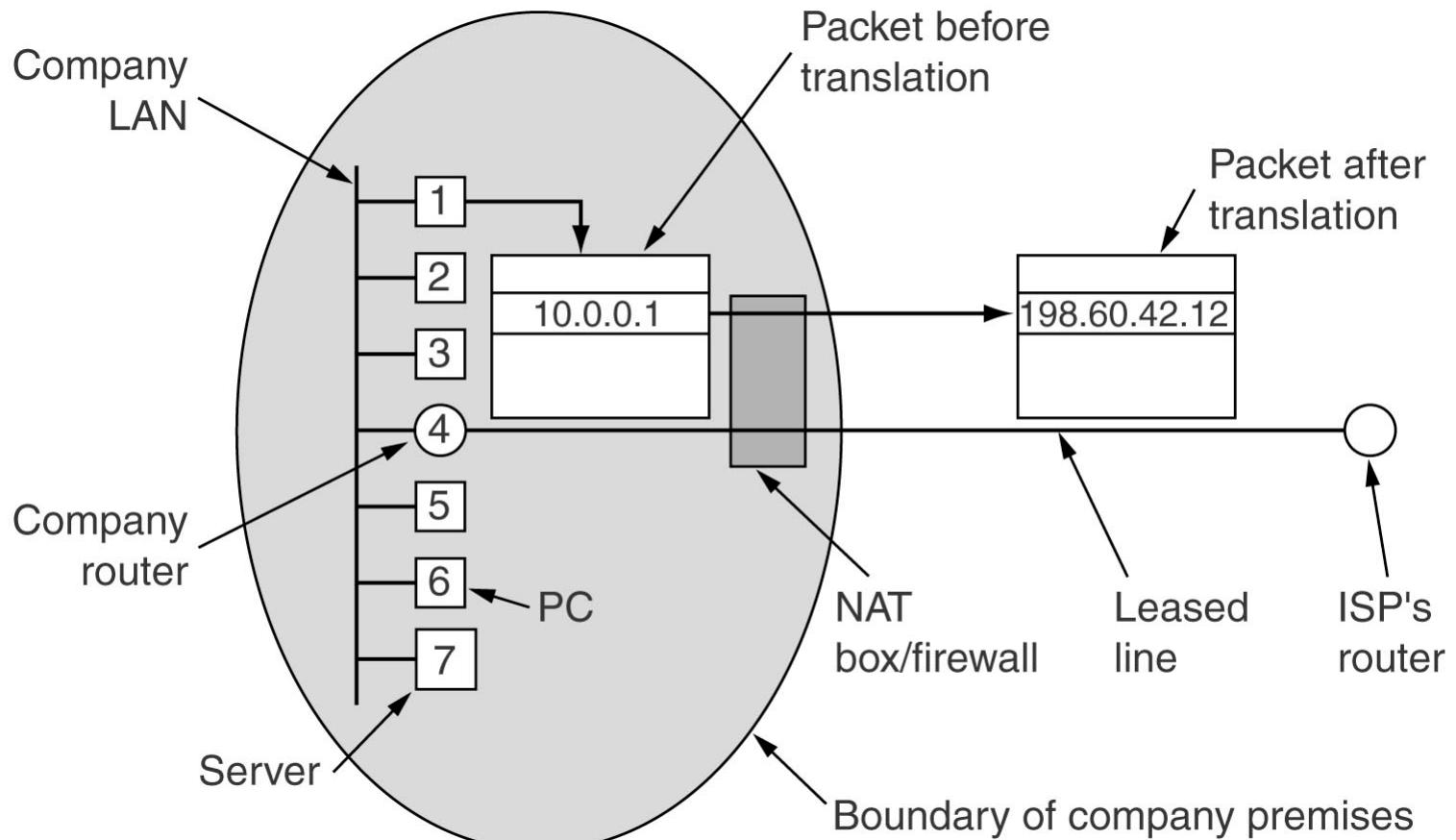
Tabela de dirijare poate include o singură intrare comună pentru adresa\_retea 11000010 00011000 00000000 00000000  
adică 194.24.0.0/19

# NAT – Network Address Translation

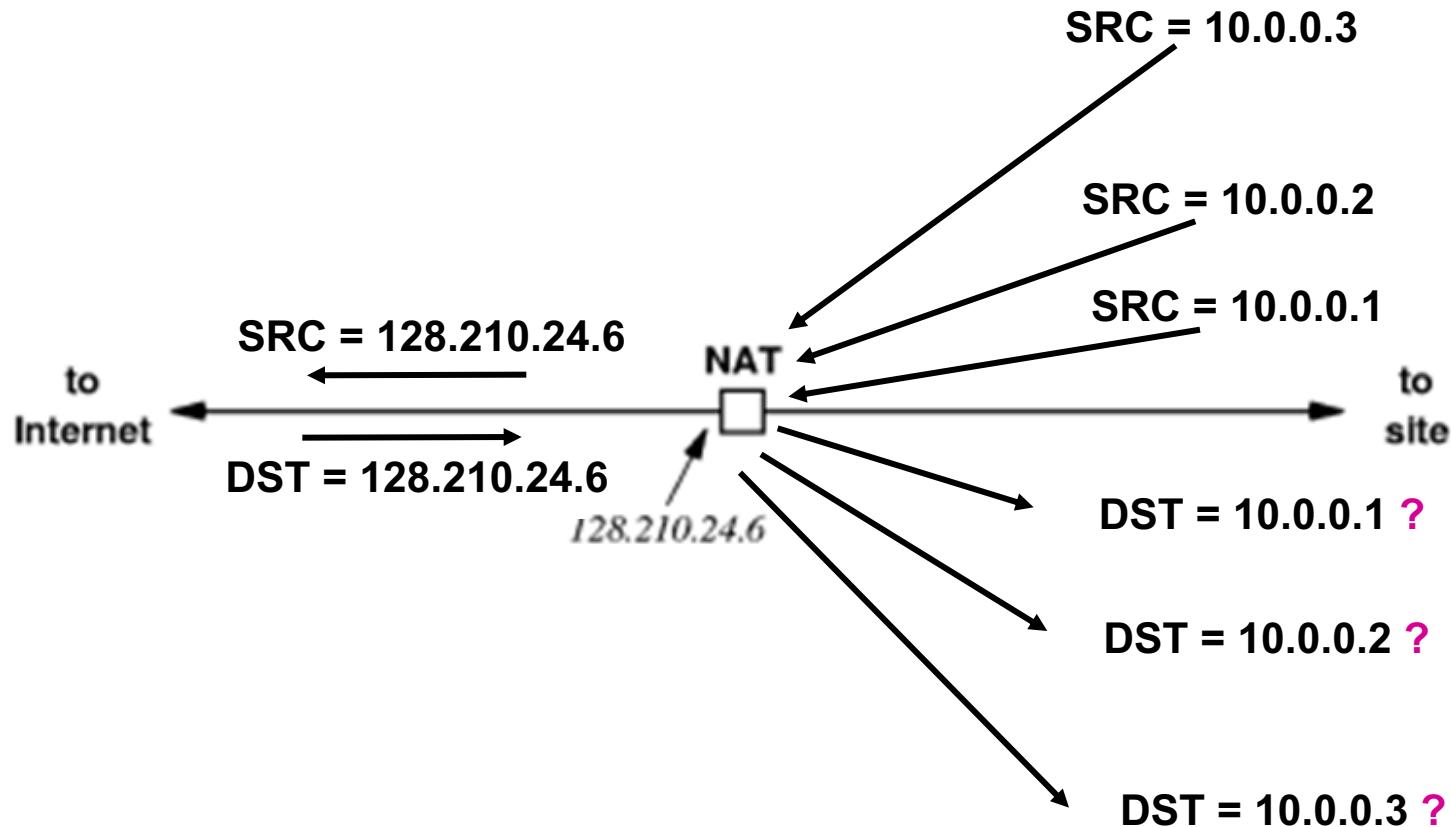
O adresă pentru mai multe calculatoare

Folosește adrese locale (private sau non-rutabile)

NAT translatează între adresa privată și o adresă globală



# Translatarea adresa globală → adresa privată



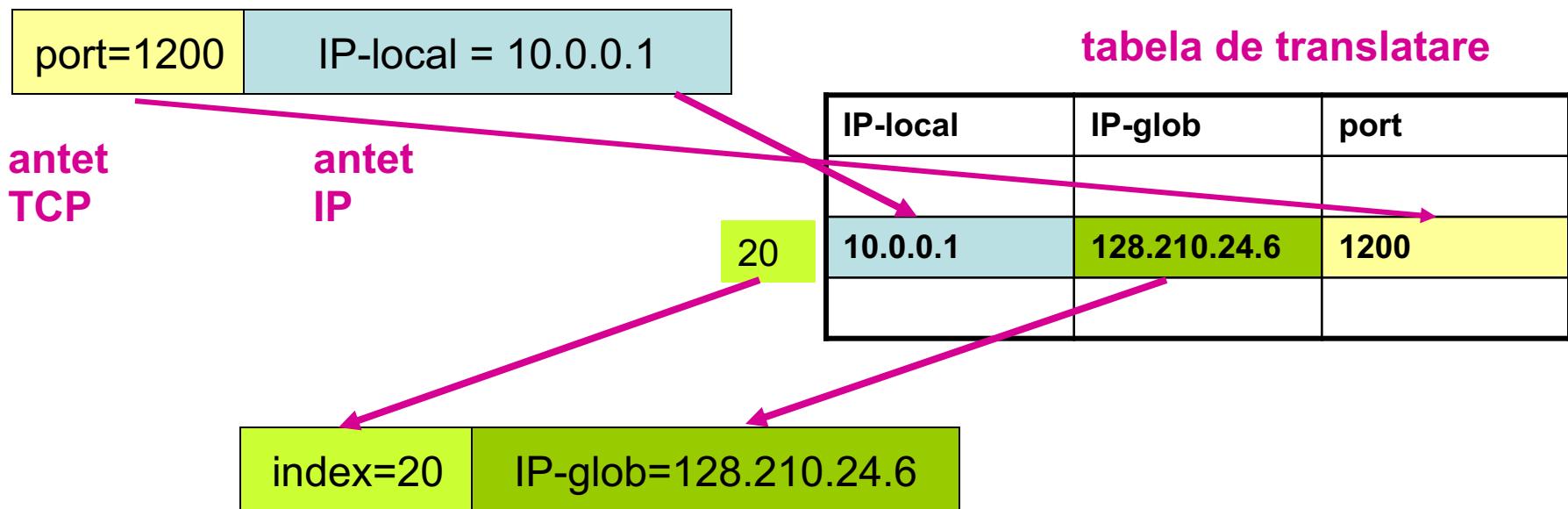
# Principiul NAT

Foloseşte

adresa IP + număr port transmititor  
tabela de translatare

## Transmisie

înlocuieşte adresa IP locală cu o adresă IP globală  
memorează (in tabela de translatare) corespondența și număr port  
înlocuieşte număr port cu index în tabela translatare  
re-compune sumele de control IP și TCP





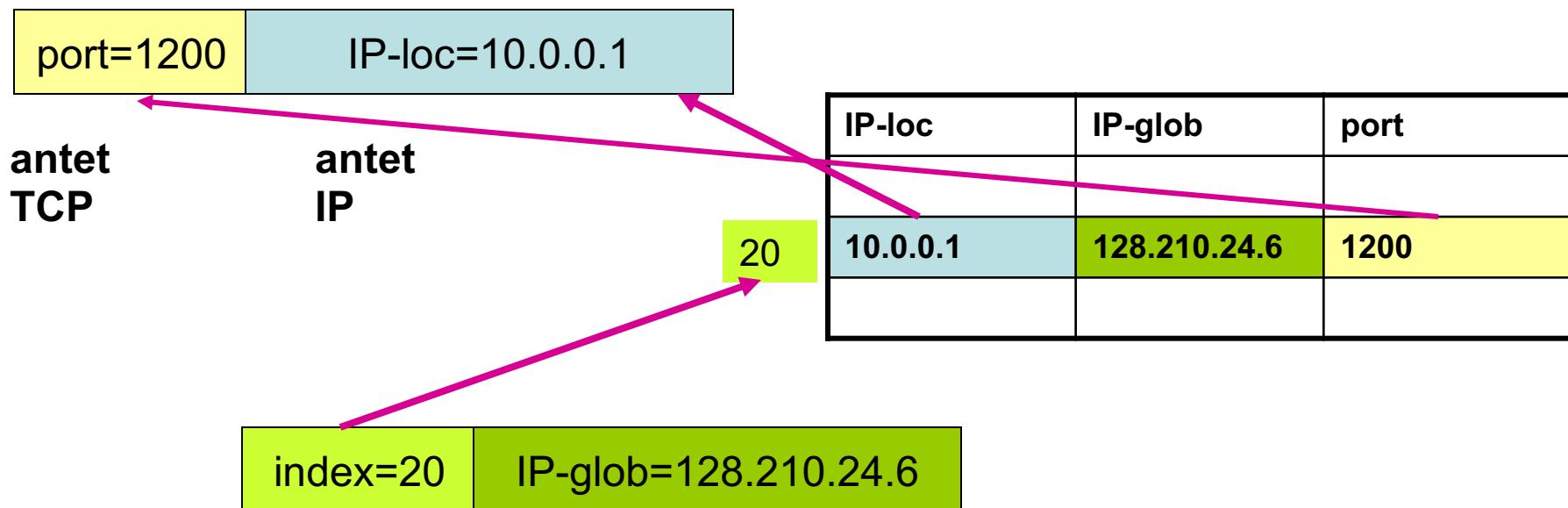
## Recepție

obține număr port din pachet (= index în tabela translatare)

extragă adresa IP locală și număr port

înlocuiește adresa IP și număr port din pachet

re-calculează sumele de control IP și TCP



# ICMP- Internet Control Message Protocol

Message type	Description
Destination unreachable	Packet could not be delivered
Time exceeded	Time to live field hit 0
Parameter problem	Invalid header field
Source quench	Choke packet
Redirect	Teach a router about geography
Echo	Ask a machine if it is alive
Echo reply	Yes, I am alive
Timestamp request	Same as Echo request, but with timestamp
Timestamp reply	Same as Echo reply, but with timestamp

ICMP folosește IP ptr transmisie & IP folosește ICMP pentru raportare de erori

Test accesibilitate (**ping** trimite **ICMP Echo** și așteaptă un timp răspunsul)

Trasare ruta (**traceroute** trimite serie de datagrame cu valori TIME TO LIVE crescătoare și primește mesaje ICMP **Time exceeded** din care extrage adresa ruterului)



# Folosire ICMP pentru aflare path MTU

**Path MTU = Maximum Transmission Unit minimă pentru o cale**

• Folosește mesaj eroare ICMP = fragmentare necesară dar nepermisă

- Sursa trimite probe cu DF în datagrama IP
- Dacă **datagrama > MTU** => sursa primește eroarea ICMP **Destination Unreachable** cu **Fragmentation Needed and Don't Fragment was Set**
- Sursa trimite probe mai scurte



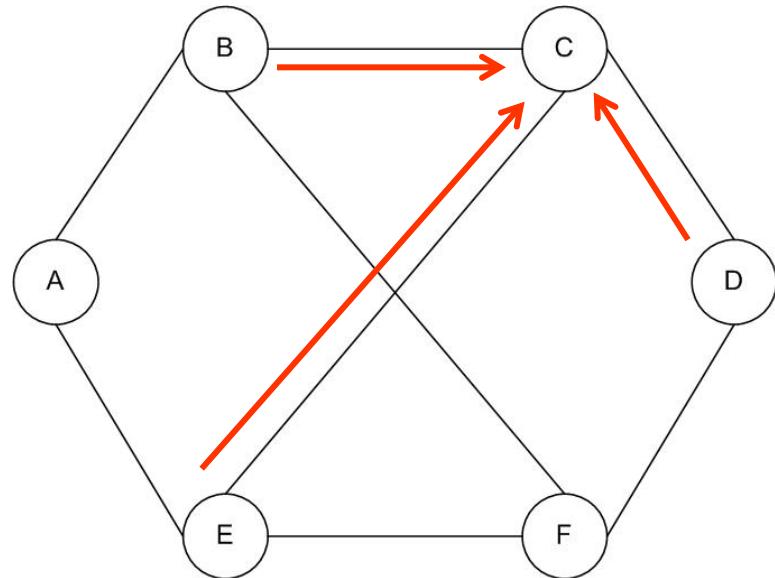
# Dirijarea - clasificare

- Fără tabele de dirijare
  - inundarea
  - hot potato
- Cu tabele de dirijare – criterii diverse
  - adaptarea la condițiile de trafic
    - statică
    - dinamică
  - locul unde se fac calculele
    - descentralizată
    - centralizată
    - distribuită
  - criterii de dirijare
    - calea cea mai scurtă
    - întârzierea medie globală
    - folosirea eficientă a resurselor
    - echitabilitatea
  - informații schimbate între noduri
    - starea legăturii
    - vectorul distanțelor
  - tipul rețelei
    - uniformă
    - ierarhică

# Vectorii distanțelor

Algoritm distribuit !

Fiecare nod trimite periodic vecinilor săi o lista cu distantele de la el la celelalte noduri.



Următorii vectori au fost primiți de nodul **C** (lista include distanțele de la B, D, E la nodurile A, B, C, D, E, F, în această ordine):

De la B: (5, 0, 8, 12, 6, 2);

De la D: (16, 12, 6, 0, 9, 10);

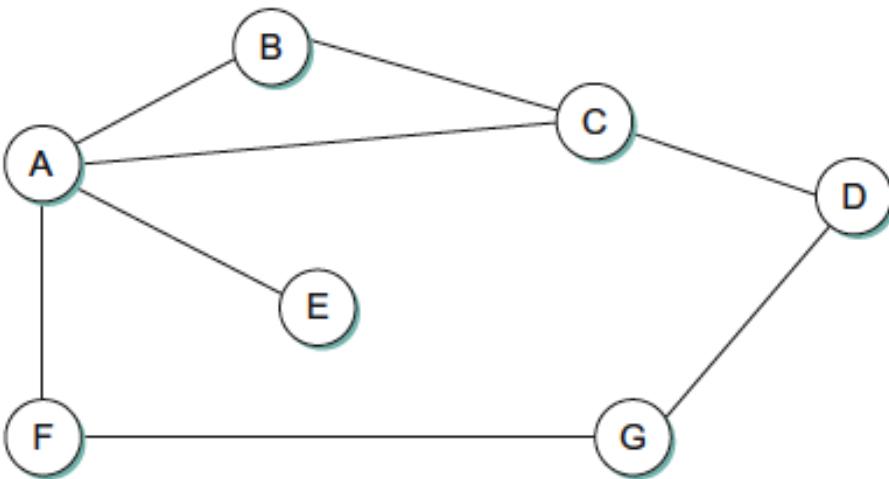
De la E: (7, 6, 3, 9, 0, 4).

De la La	B	D	E
A	5	16	7
B	0	12	6
C	8	6	3
D	12	0	9
E	6	9	0
F	2	10	4

În plus, întârzierea măsurată de la C la B, D și E este 6, 3 și 5 respectiv.

Costurile de la C La ↓	Prin →	B	D	E	Cost Min	Pas urmator
A		$5 + 6$	$16 + 3$	$7 + 5$	11	B
B		$0 + 6$	$12 + 3$	$6 + 5$	6	B
C		-	-	-	0	-
D		$12 + 6$	$0 + 3$	$9 + 5$	3	D
E		$6 + 6$	$9 + 3$	$0 + 5$	5	E
F		$2 + 6$	$10 + 3$	$4 + 5$	8	B

# Problema numărării la infinit



De la A B C D E F G cost 1 pentru orice legatura

1 2 2 3 0 2 3 ← distante initiale la E

$\infty$  **2** **2** 3 0 2 3 legatura A – E cade

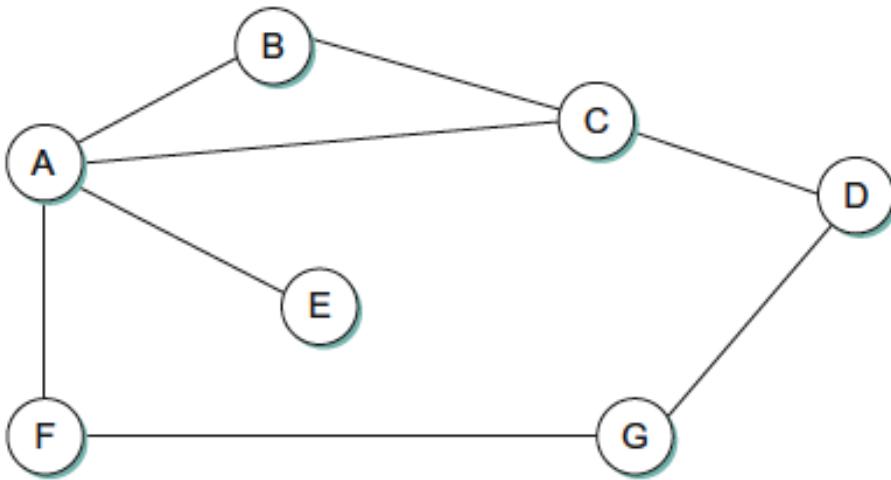
distantele catre E anunțate de: **A =  $\infty$ , B = 2, C = 2**

În funcție de ordinea evenimentelor, se pot face modificările:

$\infty$  **3** **2** 3 0 2 3 B alege ruta prin C, dist=3

B anunță dist. 3 lui A

## Problema numărării la infinit (2)



De la A B C D E F G

**4 3 2 3 0 2 3**

A alege ruta prin B dist=4

A anunta dist. 4 lui C

**4 3 5 3 0 2 3**

C recalculeaza dist=5

distanțele cresc teoretic la infinit

practic, se poate limita la un numar > diametru graf (ex. 16)

## Problema numărării la infinit - solutii (3)

**split horizon:** noile distante nu se trimit vecinului prin care trec actualele rute

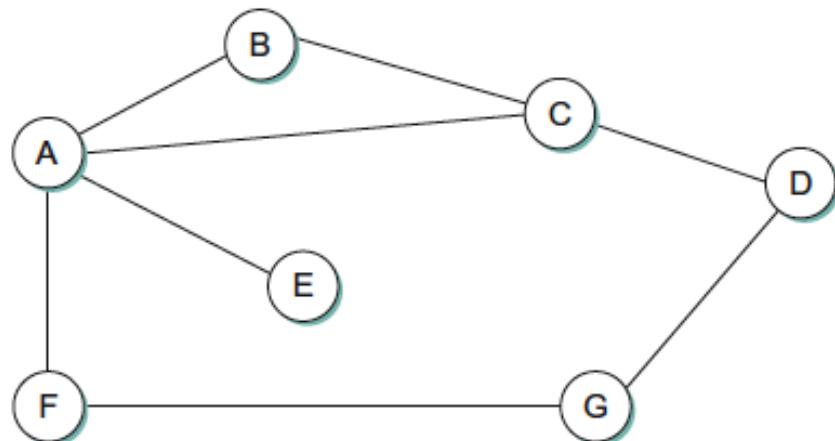
- B are ruta de distanta 2 catre E prin A
- B nu include noua distanta catre E in actualizarea trimisa lui A

**split horizon with poison reverse:** trimit o valoare f. mare

- B trimit distanta  $\infty$  catre A
- A nu va mai alege o cale prin B

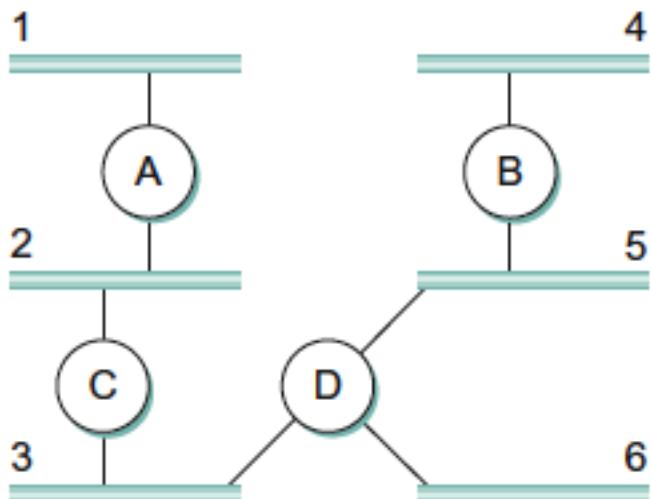
**Neajuns:** solutiile nu functioneaza in toate cazurile
 

- de ex. pentru bucle cu mai multe noduri



# RIP - Routing Information Protocol

- fiecare legatura are cost 1
- foloseste **distante** la retele (nu la noduri)
  - ruterul C are distanta 0 la reteaua 2 si 2 la reteaua 4
- transmit vectorii distanteelor la fiecare 30 secunde
- distante maxime de 15 hop-uri (16 inseamna infinit)
  - rețele de mici dimensiuni

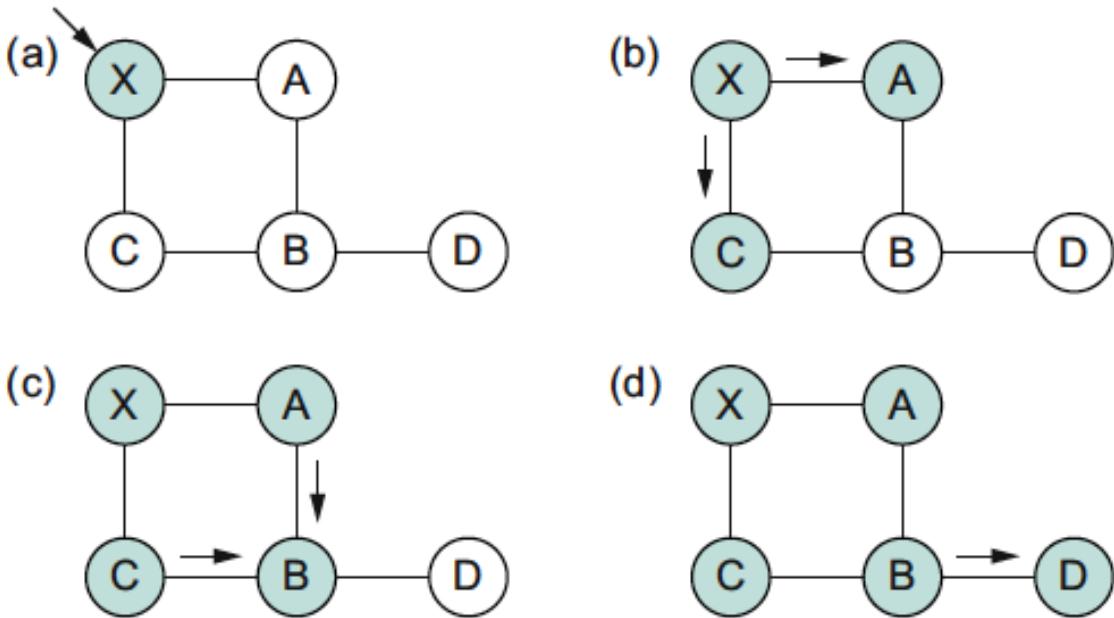




# Starea legaturii

- Presupune ca fiecare nod poate găsi **legaturile** cu vecinii și **costul** fiecărei legaturi
- Informațiile sunt **diseminate prin inundare** tuturor celorlalte noduri
  - LSP – Link State Packet **transmis prin inundare**;
  - Pachetul contine
    - Id-ul nodului care crează pachetul
    - lista **nodurilor** conectate cu **costul** fiecărei legaturi
    - un număr de secvență
    - durată de viață a pachetului (număr)
- Cu informațiile primite, **fiecare nod** va calcula **rutele cele mai scurte** către celelalte noduri

# Transmiterea prin inundare



- ID și nr. secvență:
  - nodul are o **copie a pachetului** sau pachetul este **vechi** - ignorat;
  - pachet **nou** - memorat și retransmis vecinilor (mai puțin vecinului de la care l-a primit)
- durata de viață:
  - **numar** decrementat la fiecare nod (hop) – asigura eliminarea pachete vechi



# Algoritmul căii celei mai scurte

## Algoritmul lui Dijkstra

- nnod numărul nodurilor rețelei;
- sursa nodul sursă;
- $I[i][j]$  costul legăturii  $(i,j)$ , având valorile
- 0 dacă  $i = j$ ;
  - lungmax** dacă  $i$  și  $j$  nu sunt adiacente;
  - o valoare între 0 și **lungmax** în celelalte cazuri;
- D[i] costul minim al legăturii de la sursă la  $i$ ;
- S mulțimea nodurilor deja selectate;
- V tabloul de dirijare;
- V[i] = vecinul prin care se transmit date de la nodul curent la nodul  $i$ .



```

void Dijkstra (int sursa)
{ int i, j, k;
  for (i=1; i < nnod; i++)
  {
    S[i] = 0;                                // nod neselectat
    D[i] = l[sursa][i];
    if (D[i] < lungmax)                      // distantele minime de la sursa
      V[i] = i;                             // initializeaza vecinii
    else
      V[i] = 0;
  }
  S[sursa] = 1;                                // selecteaza nodul sursa
  D[sursa] = 0;

  for ( i=1; i < nnod; i++)
  {
    gaseste nodul k neselectat cu D[k] minim;
    S[k] = 1;
    for (j=1; j <= nnod; j++)                // recalculeaza distantele
      if ((S[j] == 0) && (D[k] + l[k][j] < D[j]))
        { D[j] = D[k] + l[k][j];
          V[j] = V[k];                      // modifica tabela de dirijare
        }
  }
}

```



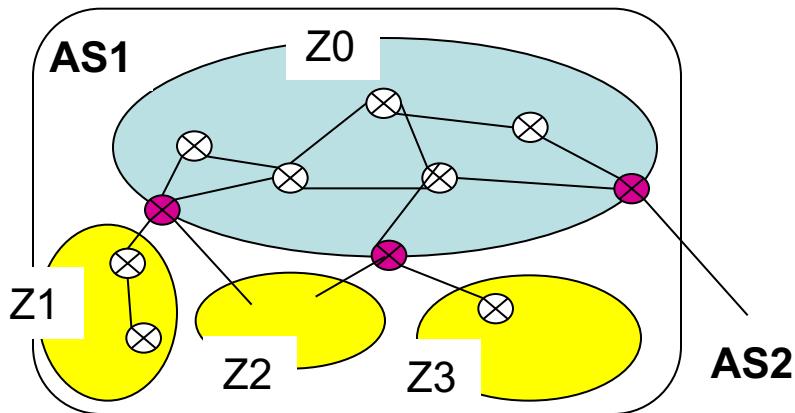
# Structura ierarhica a Internet-ului

- Internet-ul este partajat în mai multe domenii numite sisteme autonome **AS – Autonomous Systems**
  - ASs sunt rețele independente operate de organizații diferite
- Într-un AS se folosesc același algoritm de rutare “intra-domeniu”
  - denumit **interior gateway protocol**
  - ex. OSPF – Open Shortest Path First
- Rutarea între AS-uri (între domenii) folosesc, de asemenea, un protocol comun
  - denumit **exterior gateway protocol**
  - ex. BGP – Border Gateway Protocol (bazat pe RIP)

# Structura ierarhica AS

Fiecare AS este partitionat în mai multe **zone (areas)** – fiecare zonă reprezentând un **grup de rețele**

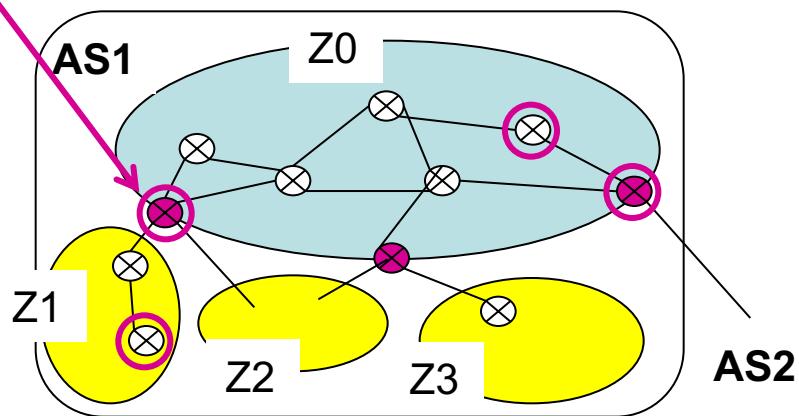
- zona Z0 – coloana vertebrală (**backbone**)
  - zone “**stub**” Z1, Z2 ... – legate la **backbone**
  - rutele între noduri din **zone stub** diferite trec prin zona Z0
- Ierarhizarea crește **scalabilitatea**
- un ruter dintr-un AS nu trebuie să stie cum se ajunge la fiecare **rețea** din AS, fiind suficient să stie cum se ajunge la **zona** în care se află rețeaua respectivă → reduce volumul tabelelor de dirijare



# Structura ierarhica AS (2)

Tipuri de rutere

- **interne** unei zone
- **de coloană vertebrală** (backbone)
- **de graniță zonală** (apartin zonei 0 si zonelor conectate)
  - nodul incercuit face parte din zonele Z0, Z1, Z2
- **de graniță AS**





# Mesaje OSPF

Mesajele OSPF permit schimbul de informații între noduri

Sunt transmise în **pachete IP** cu 89 ca număr de protocol

**Hello** – stabilește și pastrează legături cu vecinii

descoperă nodurile (ruter) cu care este conectat direct

**Link state request** - Cerere stare legătură

cere info despre anumite legături de la un alt ruter

**Link state update** - Actualizare stare legătură

trimit info despre legături, ca răspuns la o cerere

**Link state ack** - Confirmare stare legătură

confirmă primirea unui mesaj de actualizare

**Database description** – trimit LSBD – Link State Data Base

mesajele contin info despre AS sau zonă

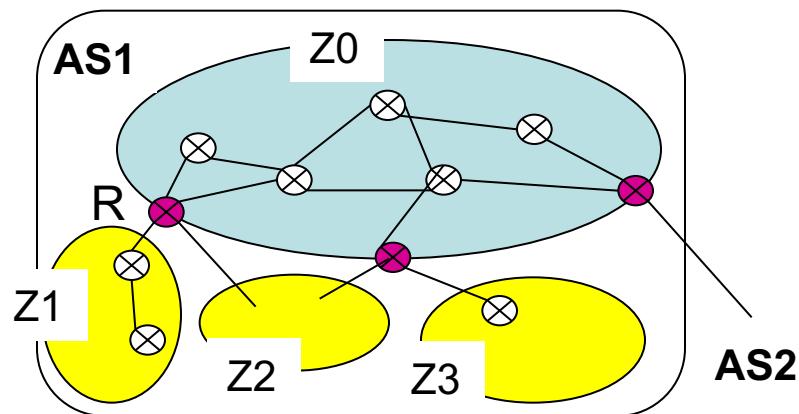
# Calcul rute - Nivel 1 (zona)

Folosind **inundarea**, fiecare ruter informeaza celelalte rutere din **zona** despre legaturile sale si costurile acestora

- ex. informatiile de starea legaturilor schimbată **intre noduri din Z1** nu se transmit în afara acestei zone!

Fiecare ruter (**inclusiv cele de granita zonala**) din **zonă** calculează separat căile cele mai scurte către ruterele din aceeași zonă

- în final, un **ruterul de graniță zonală R** va cunoaște căile cele mai scurte către oricare rețea din **Z1** și **Z2**



## Calcul rute - Nivel 2 (AS)

**Ideea:** calea unui pachet între două zone diferite are trei parti:

- de la nodul sursă la zona backbone
- traversează backbone
- de la backbone la rețeaua de destinație

Ruterele de **coloana vertebrală (backbone)**

primesc informații de la **ruterele de granită zonale** și calculează cele mai bune rute la rețele din orice zona

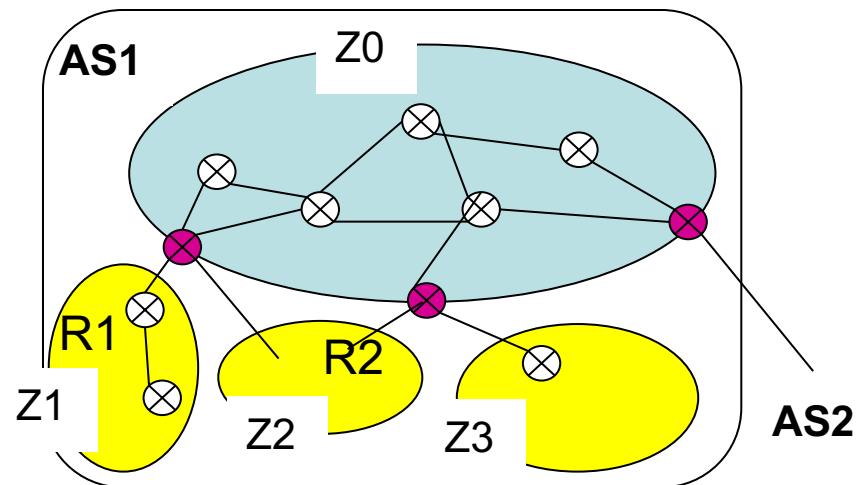
ex. **R1** oferă căile cele mai scurte pentru **Z1** și **Z2**

ruterele din **Z0** calculează căile către orice rețea din **Z1, Z2**

Rezultatele sunt **difuzate de la Z0**

înapoi la **zonele stub**, care actualizează căile cele mai scurte la rețele din alte zone

**Ex.** ruterele din **Z2** vor să știe să aleagă între **R1** și **R2** pentru rutare spre rețele din alte zone



# BGP – Border Gateway Protocol

Algoritmi orientați pe aspectele politice, de securitate, economice

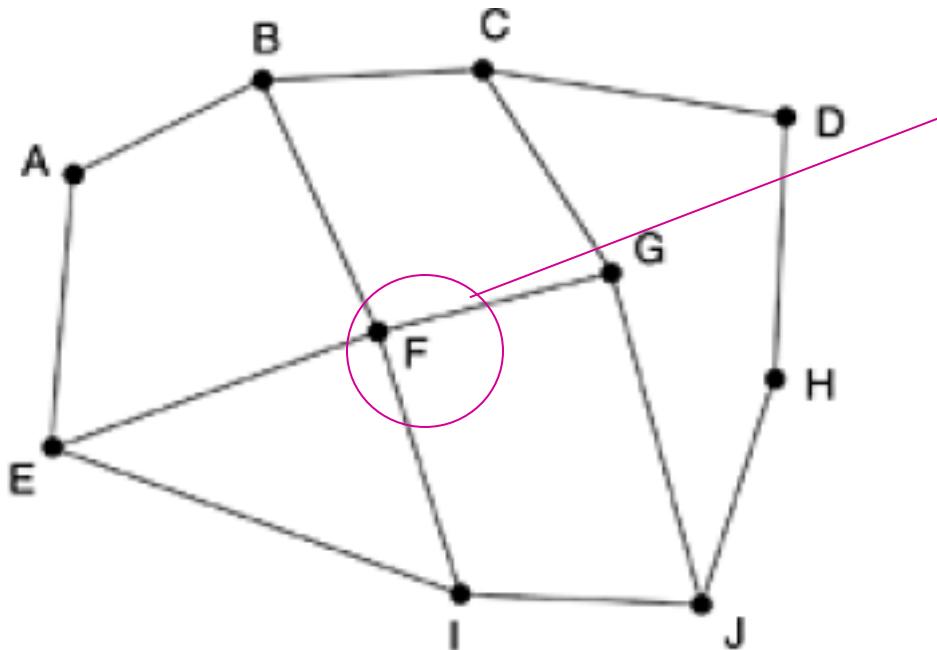
Rețea = ASes și conexiunile

Protocol = vectorul distanțelor

Tabelele de dirijare conțin și rutele spre destinație

Comunică vecinilor căile utilizate efectiv (ocoleste numaratoarea la infinit)

Ex. pp. F foloseste calea FGCD la D



G se defectează  
Informatiile primite de F  
de la vecinii ramasi, despre D:

De la B: "Eu folosesc BCD"  
De la I: "Eu folosesc IFGCD"  
De la E: "Eu folosesc EFGCD"

F elimina caile care contin G  
și alege calea FBCD



# Dirijarea în rețele ad hoc

**AODV – Ad hoc On demand Distance Vector** - Determină ruta la cerere

rețea ad hoc = graf

Muchie = conexiune – nodurile pot comunica direct (radio)

Fiecare nod = ruter + gazdă

Conține

**Tabela dirijare**

destinație,

pas următor,

distanță

nr secv destinație

altele

**Tabela history**

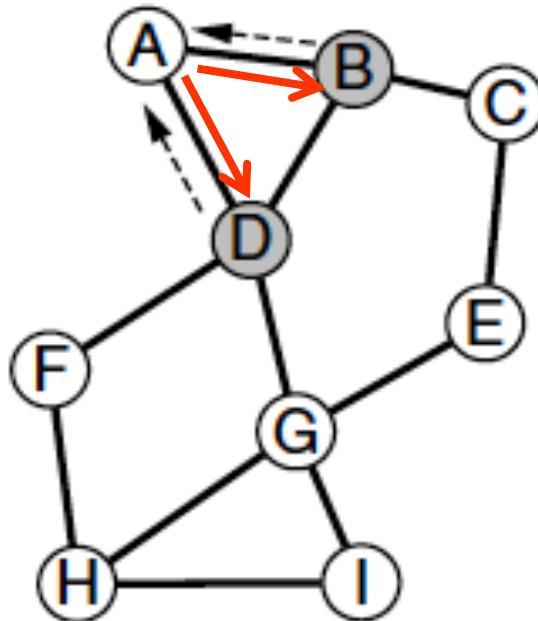
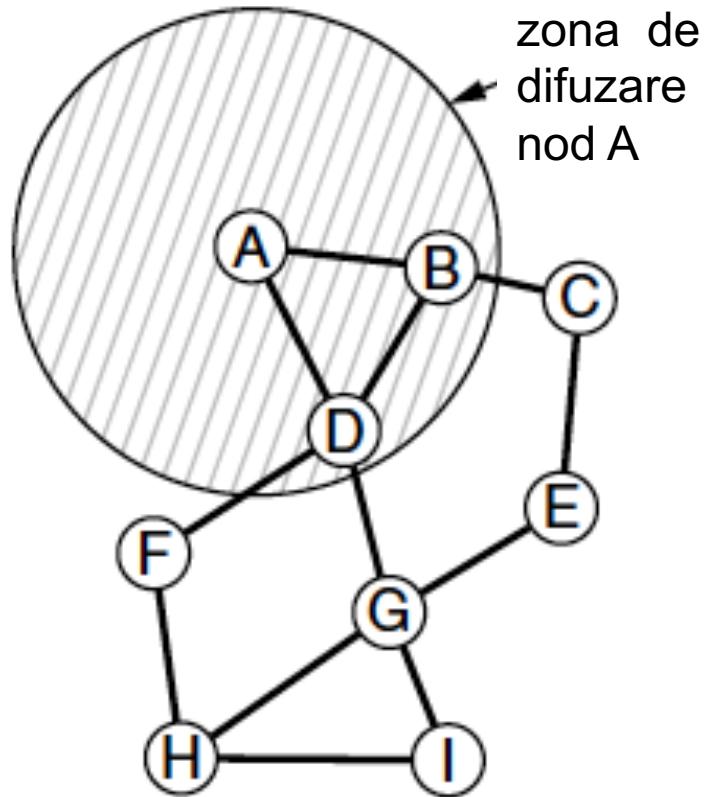
identitatile cererilor precedente

**Tabela reverse route**

calea spre sursa unui pachet de cerere

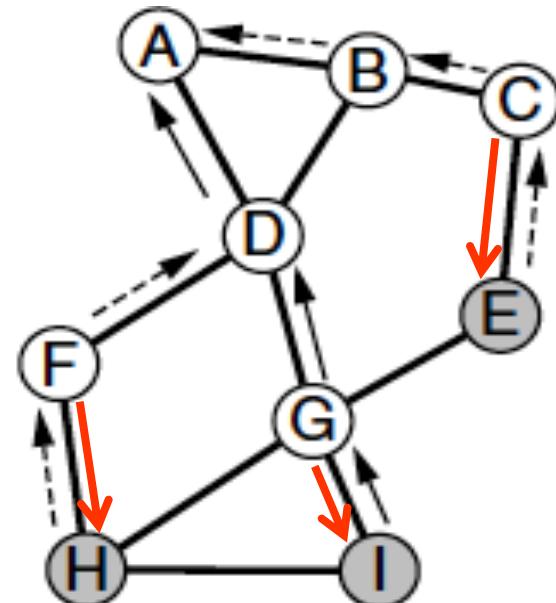
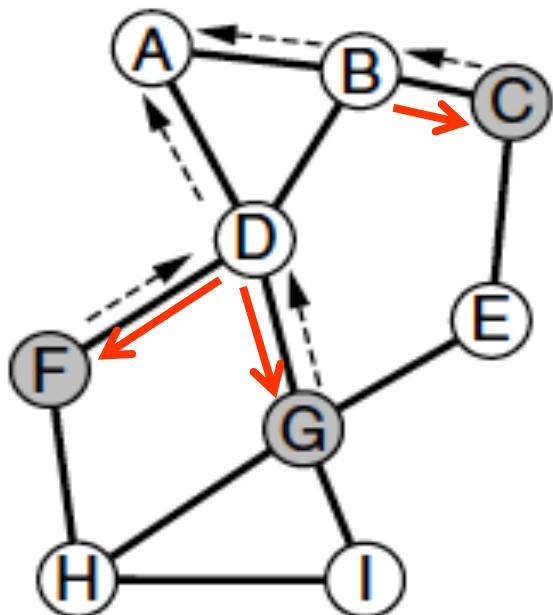
# Cum functioneaza?

**Exemplu:** A vrea să comunice cu I care nu e în tabela sa  
trebuie să descopere ruta  
trimite (broadcast) o **cerere de ruta**, care ajunge la nodurile B și D



- daca B si D nu cunosc ruta spre I, cererea este re-transmisa, prin **inundare**, celorlalte noduri
  - mai intai C, F si G
  - apoi E, H si I
- daca cererea ajunge la I, acesta va returna un **raspuns** catre A
  - raspunsul parcurge **calea inversă** prin G si D la A (sageti negre pline)

→ fiecare nod trimite raspunsul vecinului de la care a primit cererea (sageti punctate)



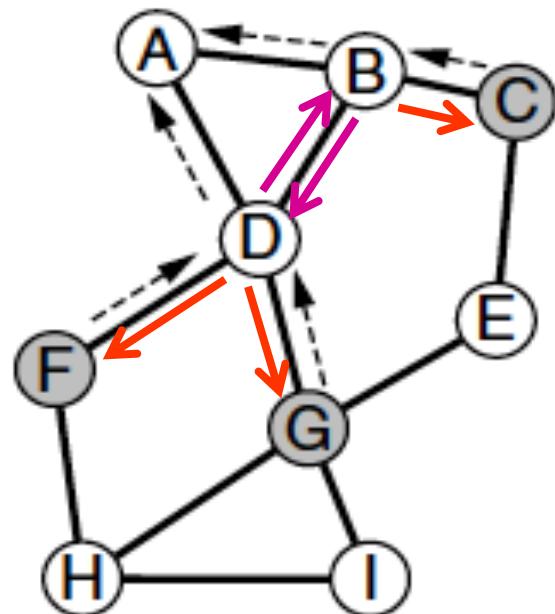
# Probleme

## Eliminarea replicilor

- pachete transmise prin inundare
- B primește de la D o copie a pachetului
- D primește de la B o copie a pachetului
- pentru a descoperi și elimina copiile
  - pachetul conține un **ID al cererii**
  - nodurile pastrează cererile primite în **tabela history**

## Lungimea caii A → I

- pachetul de cerere are un camp **Hop count**
- initializat de A la 0
- incrementat de fiecare nod care retransmite pachetul
- returnat de I în răspunsul lui



## Probleme-2

Nodul A poate avea o ruta veche

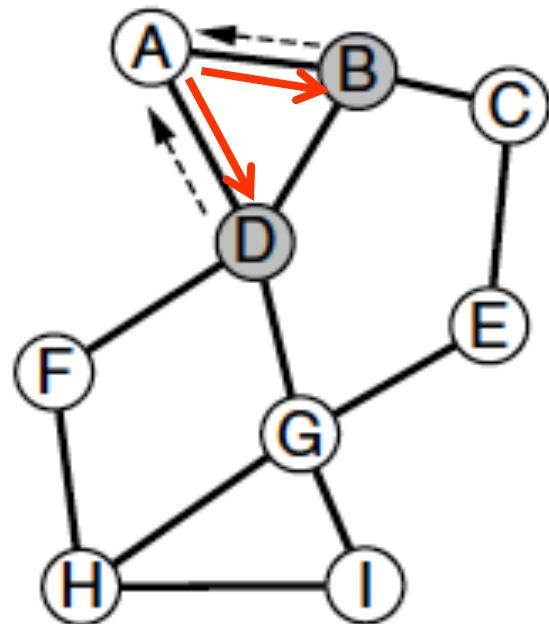
- trimite o cerere pentru actualizarea ei

Se folosește **Sequence #** pentru a deosebi  
rutele noi de cele vechi

- inclus de A în pachet cerere
  - asociat de fiecare nod rutei curente din tabela de dirijare

## Exemplu

- daca nodul D are o ruta mai noua decat cea din cerere, nu va mai re-difuza cererea si va trimite raspunsul pe calea inversa





# Pachete ROUTE REQUEST

A difuzează un pachet ROUTE REQUEST

Identificat unic prin **Source address + Request ID**

Folosește **Sequence #** pentru a deosebi rutele noi de cele vechi

Prelucrarea **ROUTE REQUEST** în fiecare nod

Verifica după în tabela **history** locală (Source address + Request ID)

Transmite **ROUTE REPLY** dacă găsit ruta nouă, adică

**Dest sequence #** în routing table > **Dest sequence #** în packet

Altfel,

incrementează **Hop count** și re-difuzează ROUTE REQUEST

memorează informația în **reverse route table**

**Source sequence #** folosit pentru actualizare tabela dirijare locală

Source address	Request ID	Destination address	Source sequence #	Dest. sequence #	Hop count



# Pachete ROUTE REPLY

I construiește ROUTE REPLY și-l trimit pe legătura inversă

Source address, Destination address sunt copiate

Hop count pus pe zero

Destination sequence # luat din contorul propriu

Lifetime = cât timp rămâne valid

Prelucrarea la alte noduri

Actualizează tabela dirijare locală

Transmite pe legătura inversă

Trece prin anumite noduri – celelalte șterg intrarea în reverse route table

Source address	Destination address	Destination sequence #	Hop count	Lifetime
----------------	---------------------	------------------------	-----------	----------

# Intreținerea rutelor

Actualizare: (1) la cerere; (2) la defectari

G cade → D descoperă (se folosesc mesaje Hello periodice)

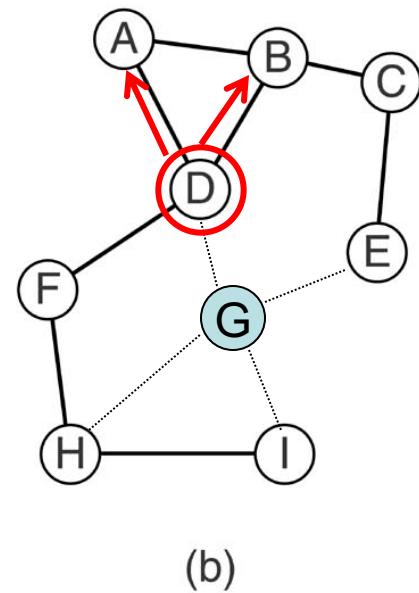
D află că G a fost utilizat pe rute către E, G și I

D anunță vecinii activi (**active neighbors**) care folosesc G, anume {A, B}

D golește intrările pentru E, G și I din tabela de rutare

Dest.	Next hop	Distance	Active neighbors	Other fields
A	A	1	F, G	
B	B	1	F, G	
C	B	2	F	
E				
F	F	1	A, B	
G				
H	F	2	A, B	
I				

(a)



(b)



# IPv6 - Motivații

## Spațiul de adrese

32 biți = peste un milion de rețele

Dar...multe sunt Clasa C, prea mici pentru multe organizații

## Tip servicii

Aplicații diferite au cerințe diferite de livrare, siguranță și viteză

IPv4 are **tip de serviciu** dar adesea nu este implementat

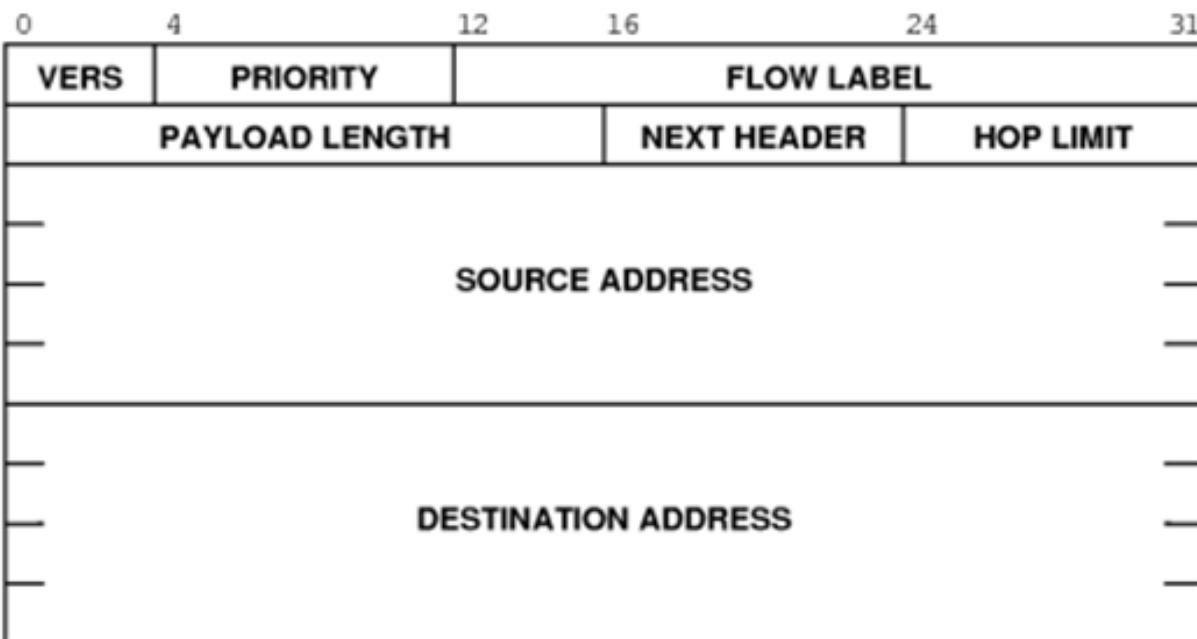
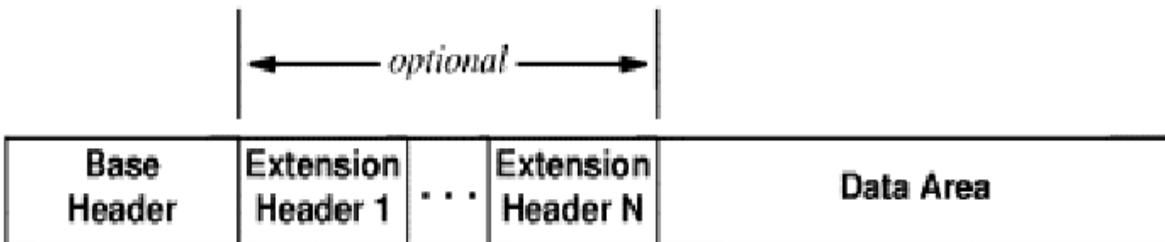
# IPv6 - format Base header

lungime fixă = 40 octeti

Priority - clasa de trafic

**FLOW LABEL** - asociază  
datagramele unui flux

**Diferențe circuit virtual**  
două fluxuri cu **aceeași**  
**etichetă** se  
diferentiază prin adr  
sursă + adr dest  
aceeași pereche  
sursă+dest poate avea  
mai multe fluxuri





# Conține mai puține info decât antet IPv4

Restul de info în extensii

NEXT HEADER definește tipul datelor (ex. TCP)

NEXT HEADER definește tipul antetului de extensie urmator (ex. Route Header)



(a)



(b)



## IPv6 – antete extensie

Hop-by-hop header – info pentru rutere – deocamdata:

suport datagrame excedând 64K (jumbograme)

specifica lungimea;

campul de lungime din antetul de baza este 0

Destination header – info aditionale pentru destinație nefolosit

Routing – lista rutere de vizitat

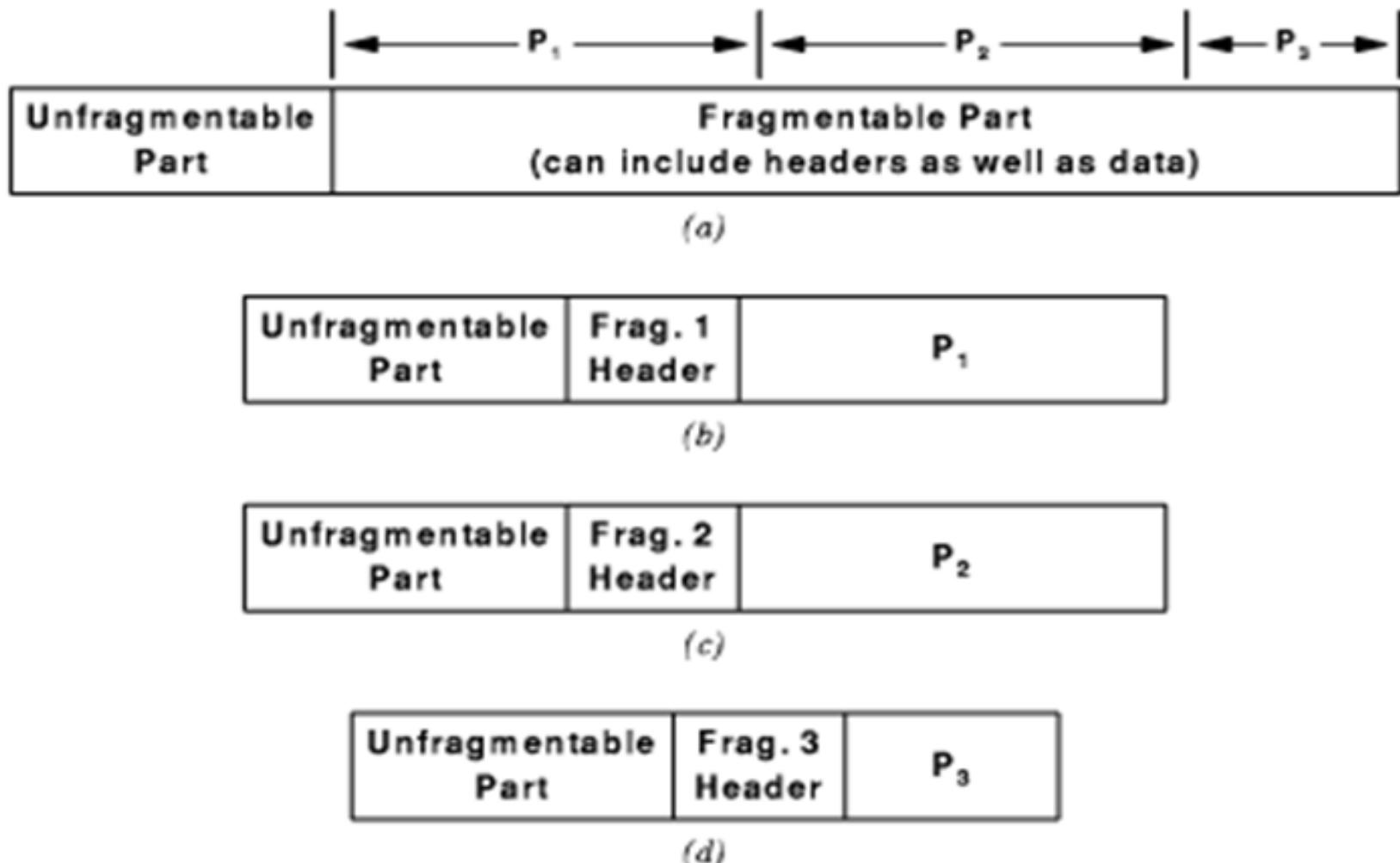
Fragmentation – identificare fragmente

Authentication – verificare identitate transmițător

Encrypted security payload – info despre conținut criptat



# Fragmentarea





## Fragmentare IPv6 – la sursă

Ruterele ignoră datagramele mai lungi decât MTU

### Sursa

Fragmentează pachetele

Descoperă path MTU

### Caracter dinamic

- calea se poate schimba

Eficiență – antet nu are spațiu pierdut

Flexibilitate – noi antete pentru noi caracteristici

Dezvoltare incrementală – ruterele care tratează anumite antete coexistă cu altele care le ignoră



adrese 128-bit

Includ prefix rețea și suffix gazdă

Fără clase de adresă – limita prefix/suffix oriunde

Tipuri speciale de adrese:

- unicast
- multicast
- cluster – colecție de calculatoare cu același prefix; datagrama livrată unuia din ele (permite duplicare servicii)



## Notația adresei

16 numere

105.220.136.100.255.255.255.255.0.0.18.128.140.10.255.255

Notație hexazecimală

69DC:8864:FFFF:FFFF:0:1280:8C0A:FFFF

Compresie zerouri

FF0C:0:0:0:0:0:B1

FF0C::B1

adrese IPv6 cu 96 zerouri prefix sunt interpretate ca adrese IPv4



# Studiu individual

A. S. Tanenbaum Rețelele de calculatoare, ed 4-a, BYBLOS 2003

## 5.1 CERINȚELE DE PROIECTARE ALE NIVELULUI REȚEA

5.2.2 Dirijarea pe calea cea mai scurtă

5.2.3 Inundarea

5.2.4 Dirijare cu vectori distanță

5.2.10 Dirijarea în rețele AD HOC

5.6.1 Protocolul IP

5.6.2 Adrese IP

5.6.4 Protocole de control în Internet

5.5.5 Protocolul de dirijare folosit de porțile interioare: OSPF

5.6.5 Protocolul de dirijare pentru porți externe: BGP

5.6.8 IPv6



# Studiu individual

A. S. Tanenbaum Computer networks, 5-th ed. PEARSON 2011

## 5.1 NETWORK LAYER DESIGN ISSUES

5.2.2 Shortest Path Algorithm

5.2.3 Flooding

5.2.4 Distance Vector Routing

5.2.11 Routing in Ad Hoc Networks

5.6.1 The IP Version 4 Protocol

5.6.2 IP Addresses

5.6.3 IP Version 6

5.6.4 Internet Control Protocols

5.6.6 OSPF—An Interior Gateway Routing Protocol

5.6.7 BGP—The Exterior Gateway Routing Protocol