Rețele

Notare: 50% examen 50% laborator

Examenul: verificare în ultima săptămână

# Curs

Profesor: [Liviu Dinu](http://nlp.unibuc.ro/people/liviu.html) (email: [ldinu@fmi.unibuc.ro](mailto:ldinu@fmi.unibuc.ro))

[Slide-uri curs](https://drive.google.com/open?id=1OJ6wtK5p4rirY5EqmD4oH3eTr9i5re9T)

[Materia condensată de la Sergiu](https://www.dropbox.com/sh/z0arf5le1antjx1/AADftWeX2JV8yOZolulCZxhKa?dl=0)

## Modele de examen

### Examen 2019

1. Protocolul simplex cu oprire și așteptare (pseudocod) = Simplex with Stop & Wait

Emițător:

while (mai sunt de trimis date) {

trimite un **frame** (cadru) de date

așteaptă **confirmarea**

}

Receptor:  
while (mai sunt de primit date) {

așteaptă un **frame** de date

procesează datele

trimite **confirmarea** la emițător

}

1. Algoritmul de dirijare cu vectori de distanțe = Distance Vector Routing

Teoria este descrisă mai jos.

1. Coduri CRC
   1. Codul CRC pentru mesajul 1101001101 și generatorul 11011

Mesajul și generatorul reprezintă coeficienții unor polinoame din Z2[X].

Convertim în polinoame:

Mesajul = X^9 + X^8 + X^6 + X^3 + X^2 + 1

Generatorul = X^4 + X^3 + X^1 + 1

Facem împărțirea cu rest a polinoamelor (ignorăm câtul):

X^9 + X^8 + X^6 + X^3 + X^2 + 1 | X^4 + X^3 + X^1 + 1

* X^9 + X^8 + X^6 + X^5

------------------------------------------------------

============== X^5 + X^3 + X^2 + 1 |

X^5 + X^4 + X^2 + X^1

-----------------------------

===== X^4 + X^3 + X^1 + 1

X^4 + X^3 + X^1 + 1

----------------------------

============== 0

Deci codul CRC pentru acest mesaj cu generatorul dat este **0**.

* 1. Să se trimită mesajul fără eroare.

Se trimite pe fir mesajul și codul CRC corespunzător.  
Dacă mesajul este trimis fără eroare, la destinație se poate calcula CRC-ul realizând împărțirea cu rest a mesajului la generator (presupunem că generatorul era deja cunoscut de ambele părți).  
Trebuie să se obțină același rest ca cel transmis pe fir.

* 1. Să se trimită mesajul cu erori.

Dacă mesajul este trimis cu erori, restul împărțirii calculat pe receptor va fi diferit (fie s-a modificat o parte din mesaj, fie s-a modificat codul CRC trimis pe fir).  
Receptorul refuză mesajul și așteaptă să fie retransmis.

### Examen 2017

1. Protocoale simple ale link layer: protocolul confirmare și retransmitere

Emițătorul trimite un cadru de date. Acesta așteaptă confirmarea (acknowledgment) pentru acesta.

* Dacă mesajul este transmis cu succes, receptorul transmite înapoi o confirmare.
  + Dacă confirmarea ajunge înapoi cu succes, emițătorul continuă cu următorul cadru de date.
  + Dacă confirmarea nu ajunge înapoi, emițătorul retransmite același cadru, până primește confirmarea (receptorul ignoră aceste cadre, doar trimite înapoi confirmări pentru ele).
* Dacă mesajul nu este transmis cu succes, emițătorul va aștepta un timp (timeout) și apoi va încerca să retransmită cadrul, până primește o confirmare.

1. Protocolul IP (Internet Protocol)

Teoria este descrisă mai jos.

1. Coduri Hamming
   1. Să se creeze codul Hamming pentru 1011010010 și să se explice procedura.

Folosim procedura descrisă aici: <https://en.wikipedia.org/wiki/Hamming_code#General_algorithm>

Mesajul pentru un cod Hamming trebuie să aibă lungimea **2^r - r - 1** pentru un **r** natural.

* Pentru r = 3 obținem un mesaj de lungime 4 (codul Hamming (7, 4)).
* Pentru r = 4 obținem un mesaj de lungime 11 (codul Hamming (15, 11)).

Vom folosi r = 4 pentru că este cea mai mică valoare în care să încapă mesajul.

Mesajul final va avea forma:

\_ \_ 1 \_ 0 1 1 \_ 0 1 0 0 1 0

unde în loc de \_ vor fi biți de paritate.

Calculăm paritatea pentru biții acoperiți de fiecare bit de paritate:

* primul \_ = acoperă pozițiile care au primul bit setat

**\_** \_ **1** \_ **0** 1 **1** \_ **0** 1 **0** 0 **1** 0

* al doilea \_ = acoperă pozițiile care au al doilea bit setat

\_ **\_** **1** \_ 0 **1** **1** \_ 0 **1** **0** 0 1 **0**

* al treilea \_ = acoperă pozițiile care au al treilea bit setat

\_ \_ 1 **\_** **0** **1** **1** \_ 0 1 0 **0** **1** **0**

* al patrulea \_ = acoperă pozițiile care au al patrulea bit setat

\_ \_ 1 \_ 0 1 1 **\_** **0** **1** **0 0 1 0**

* ultimul \_ = este extra, paritatea întregului mesaj (îl calculăm la final)

Mesajul final este:

1 0 1 1 0 1 1 0 0 1 0 0 1 0

* 1. Să se transmită mesajul (fără greșeli) și să se explice decodarea.

Trimitem mesajul 10110110010010

Pentru fiecare bit de paritate, calculăm ca mai sus sumele și observăm că dau 0.

Eliminăm biții de paritate și rămânem cu

1011001001

* 1. Să se transmită mesajul cu o greșeală și să se explice corectarea.

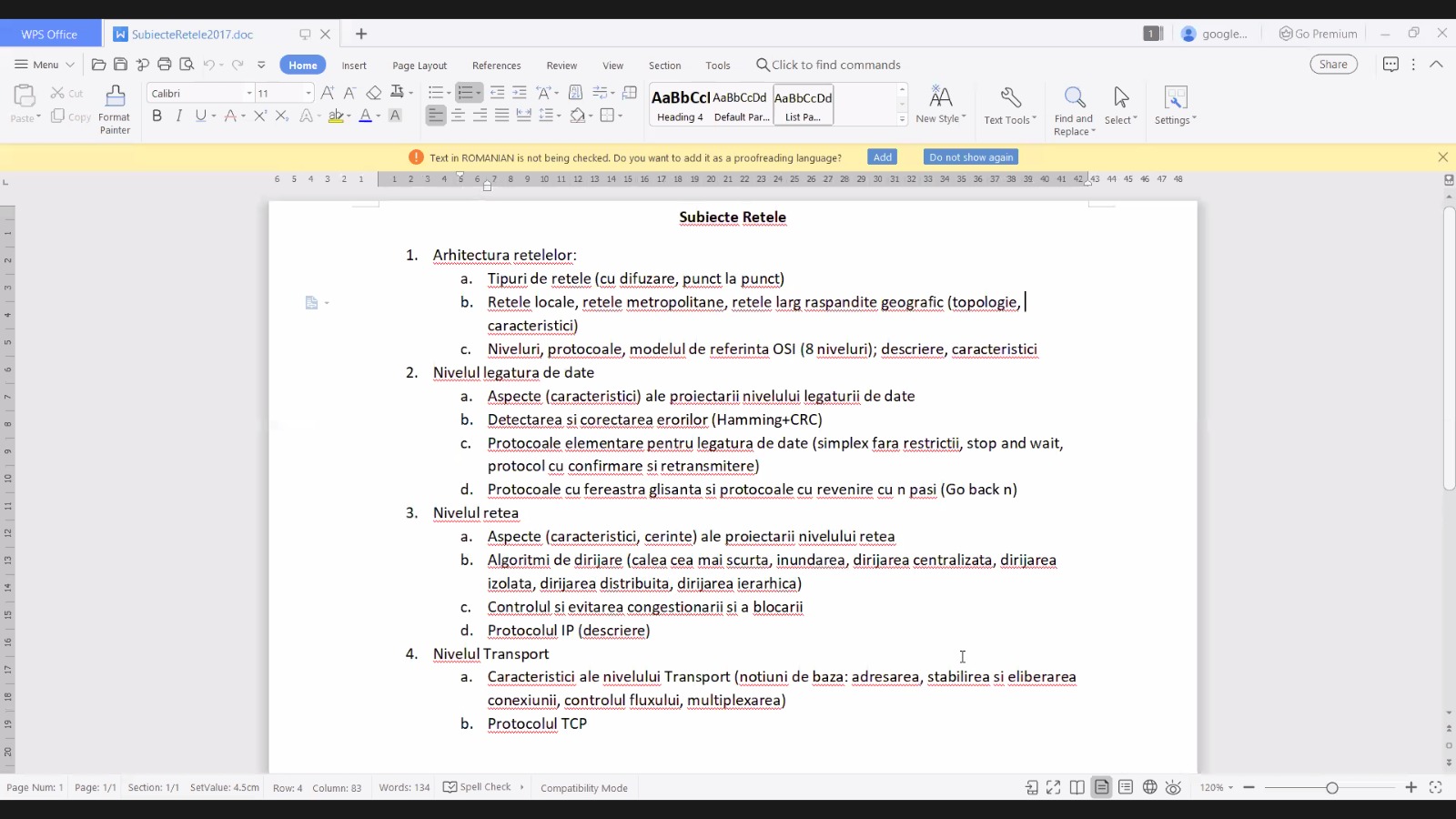
Să presupunem că s-a transmis 1011**1**110010010 (o singură greșeală).

Când calculăm paritățile observăm că

* primul bit de paritate este greșit
* al treilea bit de paritate este greșit
* toți ceilalți biți sunt corect

Punând cap la cap aceste rezultate, și ne dăm seama că greșeala este pe poziția 5 (trebuie inversat acel bit).

## Subiecte care intră la examen



## Arhitectura rețelelor

### Tipuri de rețele

În funcție de modul în care se realizează conexiunea putem avea rețele:

* **Punct la punct** = Point-to-point: leagă două dispozitive între ele (de obicei un calculator cu un router, sau de la un router la alt router). Datele sunt transmise direct de la un dispozitiv la celălalt.  
  Exemplu: prin **cablu Ethernet**, folosind **PPPoE** (Point-to-Point Protocol over Ethernet)
* Cu **difuzare**: pachetul este transmis și receptat de toate dispozitivele în același timp, și fiecare dispozitiv ignoră pachetele care nu îi sunt destinate.

Exemplu: prin **wireless**, prin **satelit**, sau prin Global Systems for Mobile Communications (GSM) **2G/3G/4G** etc.

Conectând mai multe dispozitive putem construi rețele mai mari. În momentul în care la un singur nod de rețea conectăm minim alte 2 dispozitive, avem nevoie de un sistem care se asigură că fiecare mesaj ajunge la destinația corectă => **packet switching**

### Rețele personale = Personal Area Network (PAN)

Rețele de dispozitive care se află în apropiere sau sunt purtate de o persoană.

De exemplu, poți să-ți conectezi căștile la telefon prin **Bluetooth**.

### Rețele locale = Local Area Network (LAN)

Rețea privată, care se întinde într-o singură clădire, cum ar fi o casă, un birou. Conectează mai multe calculatoare personale și dispozitive (ex. imprimante).

Rețea **enterprise** = LAN folosit de o companie

**Wireless LAN (WLAN)** = un LAN fără cablu. Dispozitivele se conectează la un singur **router wireless**, cunoscut și ca **access point** sau **base station**.

Cel mai popular standard pentru WLAN este **WiFi**.

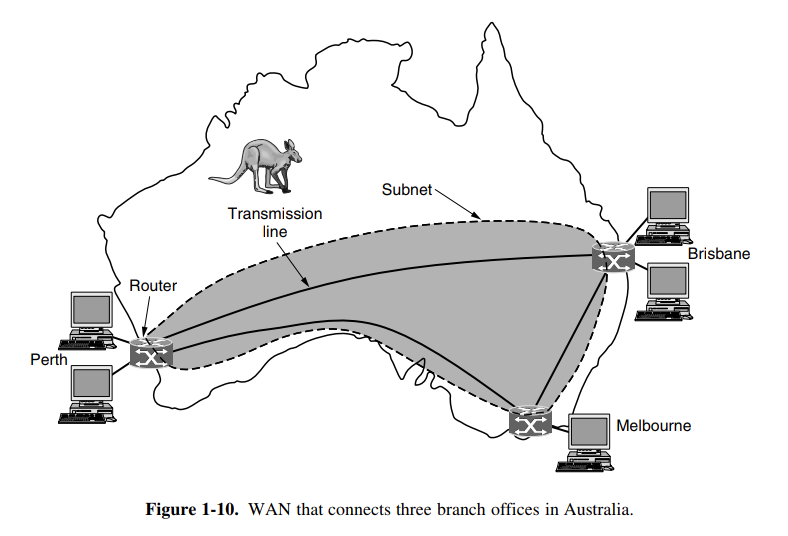
### Rețele metropolitane = Metropolitan Area Network (MAN)

Dacă conectăm mai multe LAN-uri aflate în aceeași zonă geografică (de obicei într-un oraș), obținem o rețea metropolitană.

De obicei, routerele de LAN se conectează mai departe la rețeaua pe oraș a furnizorului de internet = **Internet Service Provider (ISP)**.

### Rețele larg răspândite geografic = Wide Area Network (WAN)

În momentul în care conectăm între ele rețele din zone aflate la distanță mare, obținem WAN-uri. Se folosesc cabluri speciale de transmisie sau sateliți.



În unele cazuri, o companie ar vrea să aibă o rețea, chiar dacă organizația este distribuită în mai multe zone geografice. Compania poate face acest lucru creând un **Virtual Private Network (VPN)**.

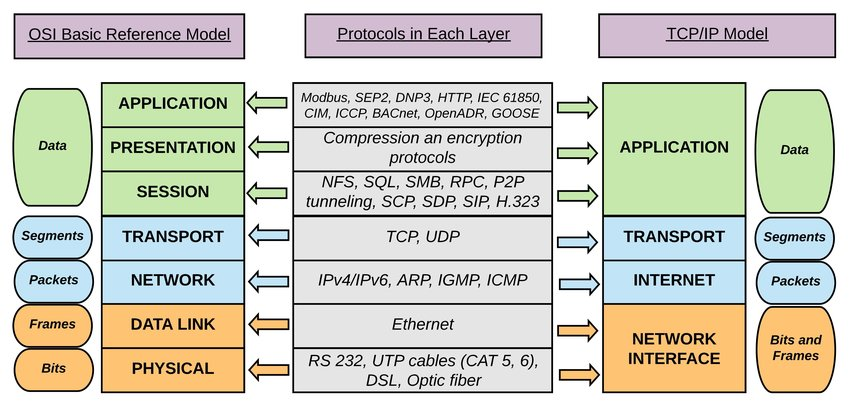
### Internetwork (Internet)

Dacă conectăm mai multe rețele între ele, obținem o rețea foarte mare numită **internetwork**. Cea mai cunoscută rețea de acest fel este adică **internetul**.

### Niveluri

Tehnologiile folosite pentru implementarea comunicării pe rețea sunt reprezentate ca o **stivă de protocoale**, împărțită pe **nivele**.

Cel mai folosit este **modelul TCP/IP**. Există și **modelul de referință OSI**, care este doar teoretic. În practică, protocoalele tind să se întindă pe mai multe straturi, așa că nu pot fi clasificate ușor:



### Protocoale

* Nivelul fizic
  + Prin cablu (ex: **Ethernet**)
  + Fără cablu (ex: **satelit**, **WiFi**, **2G/3G/4G**)
* Protocoale la nivelul legătură de date
  + **Point-to-Point Protocol (PPP)**
  + **Address Resolution Protocol (ARP)**
* Protocoale la nivelul rețea
  + **Internet Protocol (IP)**
  + **Internet Control Message Protocol (ICMP)**
* Protocoale la nivelul transport
  + **Transmission Control Protocol (TCP)**
  + **User Datagram Protocol (UDP)**
* Protocoale la nivelul aplicație
  + **Hyper Text Transfer Protocol (HTTP)**
  + **File Transfer Protocol (FTP)**

### Modelul OSI

Modelul OSI nu este cel folosit în practică; este doar un model teoretic, specificat de Open Standards Initiative (OSI); realitatea este că protocoalele sunt amestecate pe mai multe niveluri, nu avem o separare perfectă.

## Nivelul legătură de date = Data link layer

### Coduri Hamming

Un mod de a adăuga biți redundanți la mesaj pentru a permite corectarea automată a erorilor: [**Information Theory part 14: Error correction codes (Hamming coding)**](https://www.youtube.com/watch?v=cBBTWcHkVVY)(video de 5 minute)

### Coduri Cyclic Redundancy Check (CRC)

Un mod de a detecta erori, fără a le putea corecta.

Z2 = {0, 1} reprezintă numerele întregi modulo 2.

Z2[X] reprezintă polinoamele cu nedeterminata X și coeficienții din Z2 (0 sau 1).

Un exemplu este polinomul X^3 + X + 1 = 1 \* X^3 + 0 \* X^2 + 1 \* X^1 + 1

Dacă am polinomul **1**\*X^3 + **0**\*X^2 + **1**\*X^1 + **1** din Z2[X], acesta are coeficienții 1, 0, 1, 1. În binar poate fi reprezentat ca 1011.

Dacă am un int pe 32 de biți, pot stoca polinoame de grad maxim 31.

Mesajul în sine poate avea N biți. Deci este un polinom de grad maxim N - 1.

CRC reprezintă restul împărțirii polinomului codat în mesaj, la polinomul codat în header.

Rezultatul o să fie un polinom (rest) de grad maxim 31, care este apoi codificat într-un număr pe 32 de biți (CRC-ul mesajului pentru polinomul respectiv).

[**Computation of cyclic redundancy checks**](https://en.wikipedia.org/wiki/Computation_of_cyclic_redundancy_checks)

### Simplex fără restricții

Model teoretic în care datele sunt transmise într-o singură direcție, de la **emițător** la **receptor**.

Ambele dispozitive sunt tot timpul pregătire.

Timpul de procesare al mesajelor este neglijat.

Presupunem că receptorul are un buffer de memorie infinit.

### Stop and wait

În realitate, este posibil ca emițătorul să trimită date mult prea rapid ca să aibă timp receptorul să le proceseze.

În acest caz, putem folosi un protocol de tipul **stop and wait**:

* **emițătorul** transmite un cadru de date după care se **oprește**
* **așteaptă** ca răspuns un cadru fictiv, care îi confirmă că **receptorul** a terminat de procesat informația
* se transmite următorul cadru și se reia procesul

### Confirmare și retransmitere

Pentru a elimina cazul în care un pachet se pierde pe drum, protocolul poate cere ca fiecare mesaj primit să fie urmat de o **confirmare** (acknowledgement) trimis înapoi.

Dacă emițătorul nu primește în timp util o confirmare, atunci va retransmite acel pachet până primește o confirmare.

Receptorul nu are de unde să știe dacă confirmarea inițială se pierde pe drum. În acest caz, emițătorul va retransmite același pachet, în loc de cel următor, și astfel receptorul ar avea date duplicate. De aceea e nevoie ca fiecare cadru să includă un **sequence number** prin care receptorul poate să-și dea seama dacă a primit următoarele date, sau o retransmitere a ce avea deja (caz în care retransmite acknowledgement-ul).

### Protocoale cu fereastră glisantă = Sliding window

În cazul stop & wait, e posibil să dureze foarte mult până se primește confirmarea.

O soluție este să avem mai multe pachete în curs de trimitere în același timp.

Acest lucru se poate implementa cu o **fereastră glisantă**: se trimit câteva pachete, și pe măsură ce se confirmă receptarea lor, fereastra avansează și se transmit următoarele date din secvență.

Toate pachetele la stânga ferestrei sunt sigur deja recepționate. Toate pachetele la dreapta ferestrei nu au fost încă transmise nici măcar o dată, deci nu pot fi confirmate.

### Protocoale cu revenire cu N pași = Go back N

Un mod de a implementa o fereastră glisantă este prin revenire cu N pași. Emițătorul transmite, în ordine, până la N pachete. Pe măsură ce primește ACK-uri, avansează fereastra. Dacă nu primește în timp util ACK pentru primul pachet din fereastră, le retransmite pe toate (dacă primește un ACK de la un pachet de după primul, consideră că toate până la acela au fost recepționate).

## Nivelul rețea = Network layer

În momentul în care avem cel puțin 3 noduri conectate (indirect) între ele, avem nevoie de o metodă prin care să ne putem **referi la o anumită destinație**, și la un algoritm prin care să determinăm o **rută** până la acea destinație.

**Forwarding** = în momentul în care un router primește un pachet, trebuie să determine la ce destinație să-l trimită. Procesul prin care ia pachetul și îl trimite mai departe se numește **forwarding**.

**Routing** = pentru a putea forwarda un pachet, router-ul trebuie să știe care e calea „cea mai bună” pentru acesta. Determinarea unei **rute** (unui drum) optime de la un nod sursă la un nod destinație se face prin **algoritmi de dirijare** **(routing algorithms)**.

### Algoritmi de dirijare = Routing algorithms

Acești algoritmi trebuie să determine ce rută va urma un pachet pentru a ajunge de la nodul sursă la nodul destinație.

Fiecare ruter reține de obicei un **tabel cu rute** (**routing table)**, care îi spune unde trebuie să redirecționeze fiecare pachet, în funcție de destinație. Datele de ieșire de la un algoritm de dirijare sunt tabelele cu rute pentru fiecare nod din rețea.

#### Calea cea mai scurtă = Open Shortest Path First (OSPF)

În cazul în care cunoaștem toată topologia rețelei (avem tot graful), putem folosi **algoritmul lui Dijkstra** pentru a găsi cel mai scurt drum între nodul sursă și nodul destinație.

#### Inundarea = Flooding

Cel mai simplu algoritm de rutare: în momentul în care vine un pachet, îl copiem și-l trimitem pe fiecare linie care iese din router (în afară de cea pe care a venit).

#### Dirijarea centralizată

Există un singur nod care are informația despre toată rețeaua. Acesta calculează tabelele de rutare pentru fiecare router și le transmite fiecăruia.

Dacă pică nodul central, nu se mai pot actualiza tabelele de rutare din rețea.

#### Dirijarea izolată

Fiecare router își calculează singur tabelul, fără ajutor de la alte noduri din rețea.

#### Dirijarea distribuită

Routerele colaborează pentru a calcula rutele optime.

Avantajul este că în acest fel se pot găsi cele mai bune rute chiar dacă un nod din rețea pică.

#### Dirijarea ierarhică

Este ineficient pentru un router să rețină toate nodurile posibile atât din rețeaua în care se află cât și din alte rețele. Mai simplu ar fi ca fiecare regiune să aibă un router, și pentru a transmite un pachet la un nod din altă zonă, pur și simplu transmit la routerul responsabil de acea zonă, care îl transmite mai departe. Astfel se formează o ierarhie de routere.

De exemplu, un router local din București nu trebuie să știe cum să transmită un pachet până în Berkely. Routerul din București transmite la un router pe România, care apoi transmite la un router din Europa, care poate ajunge la un router din America, care trimite pachetul la un router din California, care trimite pachetul în Berkely.

### Controlul și evitarea congestionării și blocării

În momentul în care se transmit foarte multe date pe rețea, pachetele pot începe să **întârzie** (delay) sau să se **piardă** (loss), caz în care spunem că rețeaua este **congestionată**.

#### Prealocarea resurselor = Admission control

În cazul rețelelor bazate pe circuite virtuale, se poate interzice crearea unei noi rute dacă nodurile din drum nu au suficiente resurse disponibile.

#### Distrugerea pachetelor = Load shedding

Dacă nu are destul spațiu să-l memoreze, un nod poate alege să ignore pachetul.

#### Pachete de permisiune

La crearea rețelei se inițializează un număr fix de pachete de permisiune. Când vrea să transmită, nodul capturează un pachet de permisiune și transmite datele. Când acestea sunt confirmate, nodul șterge pachetul de permisiune și emițătorul generează altul.

#### Pachete de șoc = Choke Packets / Explicit Congestion Notification

Sunt transmise de routere înapoi la emițător, cerându-i să micșoreze rata cu care trimite date.

### Internet Protocol (IP)

Protocolul de bază al nivelului rețea. Fiecare pachet este precedat de un header.

Header-ul conține informații despre pachet:

* sursa și destinația pachetului (folosind **adrese IP**, care sunt pe 32 de biți în IPv4 și pe 128 de biți în IPv6)
* lungimea pachetului (în bytes)
* **Time To Live (TTL)**: cât de mult are voie să existe pachetul înainte de a fi șters. A fost introdus pentru a evita congestionarea rețelei în situația în care un pachet intră într-o buclă infinită, și rămâne permanent în rețea.

De asemenea, în header se mai pot specifica și opțiuni în plus pentru pachetul respectiv.

# Laborator

Laborant: Sergiu Nisioi (email [sergiu.nisioi@gmail.com](mailto:sergiu.nisioi@gmail.com))

Materiale: <https://github.com/senisioi/computer-networks>

[Laboratoare Înregistrate](https://drive.google.com/drive/folders/1RCizAAG4o4_TUdGCwHpDHr50bphMsqP6?usp=sharing)