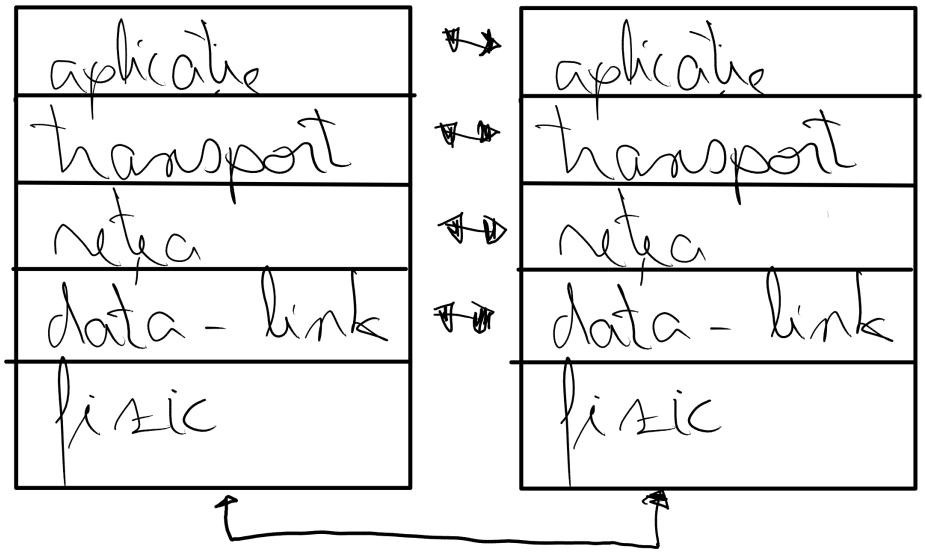
**Chestii**

- dirijare pachete
- mecanismul de adresare IP
- sistemul numelor de domain
- stiva de protocoale TCP/IP
- documente RFC

# Stiva TCP/IP

- aplicație
- transport
- rețea (internet)
- data link (date)
- fizic


## Nivel transport

**Protocoale**

- TCP
  - cu verificare

    - mai lent
- UDP
    - fără verificare
    - mai rapid

### Nivel aplicație

#### Protocoale

- bazate pe TCP
    - HTTP/HTTPS
    - FTP (File Transfer Protocol)
    - SMTP (Simple Mail Transfer Protocol)
    - POP (Post Office Protocol)
    - IMAP (Internet Message Access Protocol)
- bazate pe UDP
    - NTP (Network Time Protocol)
    - DNS (Domain Name System)

## Port

- port fizic din switch
- port TCP/IP
- Pot două procese să ocupe același port?
- NU

`socket()` returnează un descriptor de socket

```
sockfd = socket(AF_INET, TCP, IPV4);
```

`(tip_structură, ip_server, port_server)` este o structură

```
int code = connect(
    s, (AF_INET, ip_server, port_server), sizeof(structură)
);
if (code < 0) {
    // eroare
}
```

Dacă totul este ok, facem:

```
send(sockfd, buf, len, 0);
recv(sockfd, buf, len, flags);
```

Procesul server:

```
// addr = INADDR_ANY = 0.0.0.0
int code = bind(sockfd, addr, addrlen);
```

```
if (code < 0) {
    // eroare
}
```

Este posibil ca două procese sa asculte pe același port, dacă proprietățile socket-urilor sunt diferite (ip-uri diferite, protocoale diferite).

Adresă IPv4: - 4 octeți

Adresă IPv6: - 16 octeți

- `inet_addr(string)` convertește un string într-o structură de adresă
- `inet_ntoa(addr)` (network to ascii) convertește o adresă într-un string

Pachet: - ip server - port server - ip client - port client

### Server iterativ

Servește clienții pe rând.

```
// backlog = 5
listen(sockfd, backlog);
while (1) {
    int clientfd = accept(
          sockfd
        , out_client_addr
        , out_client_addrlen
    );
    recv(clientfd, ...);
    send(clientfd, ...);
    close(clientfd);
}
close(sockfd);
```

### Server concurent

Servește clienții în paralel.

```
// backlog = 5
listen(sockfd, backlog);
while (1) {
    int clientfd = accept(
          sockfd
        , out_client_addr
        , out_client_addrlen
    );
    pid_t pid = fork();
    if (fork == 0) {
        recv(clientfd, ...);
        send(clientfd, ...);
```

```
        close(clientfd);
        exit(0);
    }
}
close(sockfd);
```

### Funcții de conversie a ordinii octeților

- `htons()` host to network short
- `ntohs()` network to host short
- `htonl()` host to network long
- `ntohl()` network to host long
- ...

### Client

- ip server
- port server

## Comenzi

### Socket

- `netstat`, vechi
- `ss` (socket statistics), nou

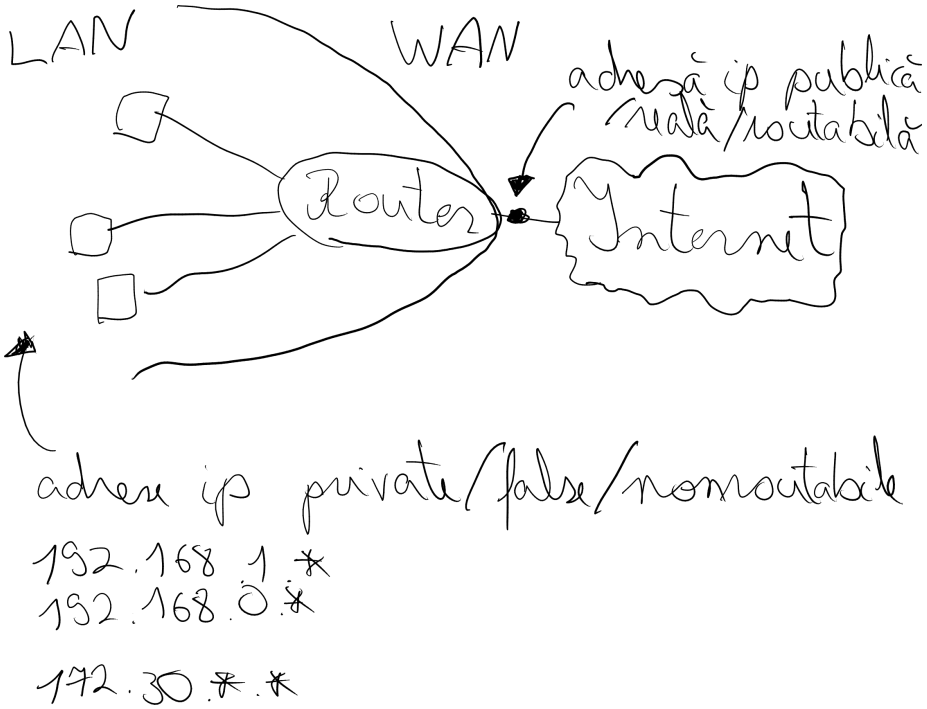
### Ip

- `ipconfig` windows
- `ifconfig` linux, vechi
- `ip addr` linux, nou

### Ping

- `ping`

### Adrese ip




#### Private

(false, nonroutabile)

- în LAN
- exemple:
  - 192.168.0.*
  - 192.168.1.*
  - 172.30..

#### Publice

(reale, routabile)

- în WAN

## NAT (Network Address Translation)

Face routerul

#### SNAT (Source Network Address Translation)

Din adresă privată în adresă publică

**DNAT (Destination Network Address Translation)**

(port forwaring, virtual servers)

Din adresă publică în adresă privată

## Probleme

**1**

- IP client: 192.168.1.17
- IP server: 192.168.1.37

Sunt obligatoriu în aceeași rețea?

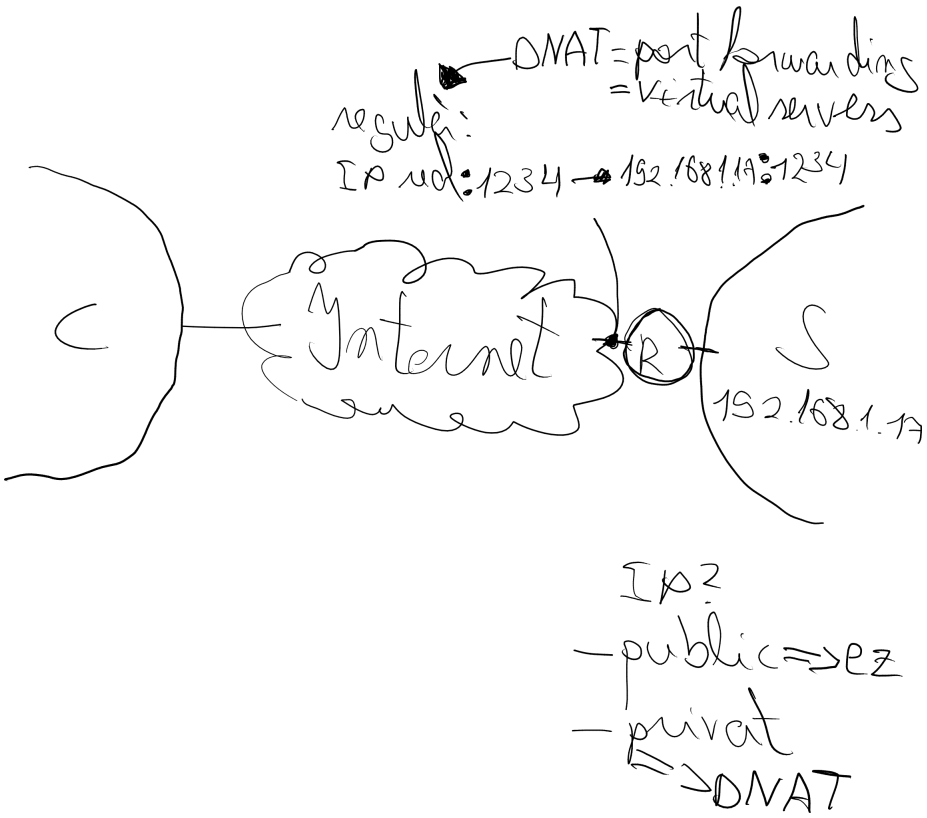
NU

**Exemple**

**VM**


VM UBUNTU
NAT
bridge
WINDOWS
by default,
VM-ul nu e
accesibil
afară din PC
NAT
Router
Internet

### Worldwide




### Malicious DHCP

IP-ul routerului trebuie să facă parte din aceeași clasă cu IP-urile calculatoarelor din rețea.

TODO: Server `DHCP` care nu e pe router.

Dimensiunea maximă a unui pachet IP: 64KB.

MTU = Maximum Transfer Unit

Este posibil ca un server să servească pe aceeași combinație de (ip, port) și TCP și UDP.

## Socket-uri UDP

### Client

```
c = socket(AF_INET);
// nu e nevoie de connect()
sendto(c, buf, buf_len, flags, dest, dest_len);
```

```
recvfrom(c, buf, buf_len, flags, src, src_len);
close(c);
```

#### Server

```
s = socket(AF_INET, SOCK_DGRAM);
bind(s, ...);
// nu e nevoie de listen()
// nu e nevoie de accept()
while (1) {
    recvfrom(c, buf, buf_len, flags, src, src_len);

    // faci fork()
    // fiul face socket() si bind() pe un port nou
    // si trimite raspunsul prin acel socket

    // sendto trimite datele unui partener, ale cărui date de identificare (ip, port) sunt
    sendto(c, buf2, buf2_len, flags, src, src_len);
}
```

## Structură pachet TCP

- header
  - port sursă
  - port destinație
  - număr de secvență (număr de octeți transmiși kinda)
  - număr de confirmare
  - flaguri
    - `SYN`
    - `ACK`
    - `FIN`
    - ...
  - checksum
  - ...
- date

#### 3-way handshake

- `SYN`
- `SYN`,`ACK`
- `ACK`

#### close()

Se trimite un pachet cu flag-ul `FIN`.

### Structură pachet UDP

- header
  - port sursă
  - port destinație
  - lungime
  - checksum
- date

### Structură pachet IP

- header
  - versiune
  - lungimea pachetului
  - header checksum
  - TTL (Time To Live)
  - IP sursă
  - IP destinație
  - ...
- pachet TCP/UDP