# Tema 7. NIVELUL TRANSPORT

***Tema descrie funcţiile pe care nivelul transport le îndeplineşte într-o reţea de calculatoare. Sunt evidenţiate parametrii de calitate a serviciilor realizate la nivel transport şi primitivele pe baza cărora se pot realiza aceste servicii. O atenţie cuvenită se acordă adresării la nivel transport, ceea ce permite multiplexarea şi recunoaşterea individuală a aplicaţiilor care pot rula simultan pe acceaşi staţie de lucru sau pe staţii diferite. De asemenea, se detaliază algoritmii de stabilire a conexiunilor la acest nivel. Apoi se prezintă cele două protocoale de bază de nivel transport (TCP şi UDP) care împreună cu protocolul IP stau la baza funcţionării actuale a Internetului.***

***După parcurgerea şi însuşirea acestei teme, studentul va cunoaşte:***

- ***Cum se se realizează comunicaţia la nivel transport între două entităţi pereche ca utilizatori finali, inclusiv fragmentarea şi reasamblarea mesajelor***
- ***Legătura cu protocoalele de pe nivelele adiacente***
- ***Ce sunt adresele de transport, cum se identifică aplicaţiile individuale şi cum se face multiplexarea în jos***
- ***Funcţionarea protocoalelor UDP şi TCP şi formatul datagramelor corespunzătoare***
- ***Programarea de aplicaţii cu socluri în Java***

*Orele de laborator urmăresc dezvoltarea de aplicaţii de comunicaţie la nivel transport folosind programarea cu socluri în UNIX şi Java.*

*Studenţii vor realiza ca temă de casă un program de comunicaţie client server folosind limbajul C sau Java*

*Timpul minim pe care trebuie să-l acordaţi studierii acestui modul este de 6 ore.*

***Cuvinte cheie**: **TCP, UDP, QoS, port, soclu, RPC, RTP, conexiune, congestie, multiplexare***

## Cuprins

**Bibliografie**

1. ***Iosif Praoveanu - Rețele de calculatoare, Ed. Universității Titu Maiorescu, Bucureşti 2009 - cap 5 Nivelul transport***
2. ***Andrew S. Tanenbaum - Rețele de calculatoare, ediția a 4-a, Editura Byblos, Bucureşti 2004 – cap. 5 Nivelul aplicație***
3. ***I. Jurca – Programarea rețelelore de calculatoare, Ed, de Vest, Timişoara, 2000, pag. 113-149***
4. ***http://thor.info.uaic.ro/~busaco/teach/courses/net/presentations/net5.pdf***

**Nivelul transport** are rolul de a transporta datele de la maşina sursă la maşina desinație într-o manieră sigură şi eficace din punct de vedere al costurilor, independent de rețeaua sau de rețelele fizice folosite. La acest nivel serviciile pot fi orientate pe conexiune, cazul cel mai uzual, sau neorientate pe conexiune. Bazându-se pe serviciile furnizate de nivelul rețea, nivelul transport furnizează nivelului aplicație o legătură sigură cap la cap şi intervine mai ales atunci când transferul de date la nivel rețea întâmpină dificultăți: cad rutere, apar congestii, se pierd frecvent pachete, apar întârzieri mari etc.

**Nivelul transport face posibil ca serviciile de transfer de date să fie mai sigure decât cele de nivel rețea.** Datele pierdute sau recepționate incorect pot fi detectate şi corectate de către nivelul transport.

## 5.1 Calitatea serviciilor (QoS) la nivel transport

Nivelul transport furnizând servicii nivelului aplicație care este direct legat de beneficiarii rețelei, este foarte importantă calitatea serviciilor de transport de date asigurate nivelului aplicație.

Calitatea serviciilor (QoS) asigurate de nivelul transport se poate aprecia printr-un număr de indicatori de calitate ale căror valori se precizează sau se negociază în faza de stabilire a conexiunii la nivel transport. Numărul parametrilor QoS poate fi mare, dar furnizorii de rețele garantează în general un număr mic.

Exemple de astfel de parametrii QoS sunt:

- ***Întârzierea la stabilirea legăturii*** se referă la timpul scurs între momentul lansării unei cereri de stabilire a conexiunii la nivel transport şi primirea confirmării de realizare a acesteia;
- ***Probabilitatea de insucces la stabilirea conexiunii*** corelată şi cu un interval maxim de timp convenit;

- ***Rata de transfer*** precizată sub forma numărului de octeți sau de biți de date utilizator transferate în unitatea de timp. Ea se precizează pentru ambele sensuri de transmisie şi este legată de anumite cerințe de transfer, cum ar fi: rată constantă (CBR – constant bit rate), rată de transfer variabilă (VBR - variable bit rate), rată disponibilă (ABR –avalable bit rate) etc.;
- ***Întârzierea*** dată de timpul scurs din momentul emiterii unui mesaj de către maşina sursă şi până în momentul recepționării lui de către maşina destinație;
- ***Rata reziduală a erorilor*** reprezentă procentul de pachete, mesaje sau date pierdute sau transmise incorect, erori care nu sunt corectate în niciun fel.
- *Prioritarea* în stabilirea de conexiuni, de transfer a datelor, de restabilire a defectelor de rețea etc.;
- ***Protecția datelor*** împotriva accesului neautorizat, pierderii, deteriorării, etc şi în general gradul de implementare a serviciilor de securitate până la nivelul transport;
- ***Rezilienţa*** reprezentând probabilitatea ca nivelul transport să închidă brusc o conexiune datorită unor probleme interne.
- ***Transferul de date în timp real*** reprezintă capabilitatea rețelei de a transfera datele pe măsură ce ele se produc, fără întârzieri variabile în nodurile rețelei. Acest parametru este foarte important în aplicațiile de transfer de date în timp real, cum ar fi rețelele vocale, videoconferință etc.
- ***CBR*** (transfer de date cu rată constantă) arată capabilitatea unui canal de comunicație de a transfera datele cu o rată constantă garantată;
- ***VBR*** *(*transfer de date cu rată variabilă) arată capabilitatea unui canal de comunicație de a asigura rate de transfer variabile în funcție de disponibilitățile de moment ale rețelei.

## 5.2 Primitivele de bază ale serviciului transport

În funcție de complexitatea sa şi de serviciile pe care poate să le asigure nivelului superior (în special nivelului aplicație), nivelul transport poate fi mai simplu sau mai complex. În cazul cel mai simplu, un serviciu de transport trebuie să permită stabilirea conexiunii între punctele finale ale aplicației, transferul datelor şi desfacerea conexiunii. În acest scop se pot folosi 5 primitive (acțiuni) simple de genul celor specificate mai jos.

*Tabelul 5.1*

| *Primitiva* | *PDU transmis* | *Explicații* |
|---|---|---|
| **LISTEN** | - | Se blochează în aşteptarea unei cereri de conectare |
| **CONNECT** | Connection REQ | Cerere de stabilire conexiune |
| **SEND** | Date | Transmitere informație utilizator |
| **RECEIVE** | - | Se blochează până la primirea datelor |
| **DISCONNECT** | Disconnection REQ | Cerere de desfacere a conexiunii (oricare parte) |

Orice mesaj schimbat între două entități corespondente la nivel transport se numeşte generic TPDU -Transport Protocol Data Unit (unitate de date protocol de nivel transport). El conține informația utilă (a unui proces, primitivă etc.) şi un antet de nivel transport. Mesajul astfel împachetat, este plasat nvelului inferior, rețea, care îi adaugă antetul de rețea şi îl trimite mai jos la nivel legătură de date unde este plasat în cadre spre a fi livrat mediului fizic.

Într-un exemplu simplu de conexiune la nivel transport între un client şi un server aflat la distanță, cele 5 primitive de mai sus sunt folosite după cum urmează.

1. În repaus serverul execută primitiva **LISTEN** care îl ține blocat în aşteptarea unei cereri de conexiune.
2. Când un client doreşte să se conecteze la server, el execută primitiva **CONNECT** care, apelând la o funcție de bibliotecă trimite o cerere de conectare (TPDU de tipul **Connection REQ**) spre server. În acelaşi timp blochează apelatorul pentru a nu executa alte acțiuni până ce nu a primit un răspuns.
3. Când soseşte apelul la server, entitatea transport (softul de nivel transport) de pe server verifică dacă serverul este blocat în **LISTEN** (deci dacă aşteaptă o cerere de conexiune). Dacă da, îl deblochează şi trimite înapoi clientului un TDPU de tipul **Connectoin Accepted**. Când aceasta ajunge la client îl deblochează şi conexiunea este stabilită.
4. Începe schimbul de date folosind primitivele **SEND** şi **RECEIVE**. Cea mai simplă posibilitate este ca una din părți să execute RECEIVE aşteptând date de la cealaltă, iar aceasta să execute SEND (să trimită date). Sosirea pachetului de date deblochează receptorul din starea RECEIVE, permite acestuia să prelucreze TDPU-ul recepționat şi să trimită înapoi şi un răspuns. Totul merge bine dacă fiecare parte ştie cine trimite şi cine ascultă pe rând.
5. Când se doreşte încheierea conexiunii, una dintre părți trimite un TDPU de tipul **Disconnectoin REQ** care declanşează procedura de încheiere.

## 5.3 Adresarea la nivel transport

**Adresa de transport** este folosită pentru a **defini o conexiune între două entități corespondente finale** pe care rulează o aplicație, un proces. Conexiunea la acest nivel trebuie să precizeze atât maşina pe care rulează aplicația, cât si aplicația propriu-zisă.

Prin urmare, **adresele de nivel transport sunt necesare pentru a identifica procesele care comunică simultan şi trimit date pe nivelurile inferioare** pentru a fi împachetate succesiv şi transferate mediului fizic.

De exemplu, pe un calculator gazdă pot rula simultan mai multe aplicații de tipul transfer de fişiere, poştă electronică, navigare pe Internet, transmisii de voce etc. Toate aceste date ajung într-o formă multiplexată pe acelaşi mediu fizic, linia de comunicație la care este conectată gazda. La celălalt capăt, aplicațiile pot rula pe servere sau pe gazde diferite. Identificarea aplicațiilor se face la

nivel transport prin adrese speciale care, în cazul protocolului TCP au lungimea de 16 biți (în total 65 535 puncte finale posibile).

Un punct final al unei conexiuni la nivel transport trebuie însă legat şi de o maşină (gazdă) anume, adică este nevoie şi de o adresă de rețea. În Internet, asocierea dintre o adresă de rețea (adresă IP) şi o adresă de transport (adresă TCP) formează ceea ce se numeşte **socket** (**soclu**).

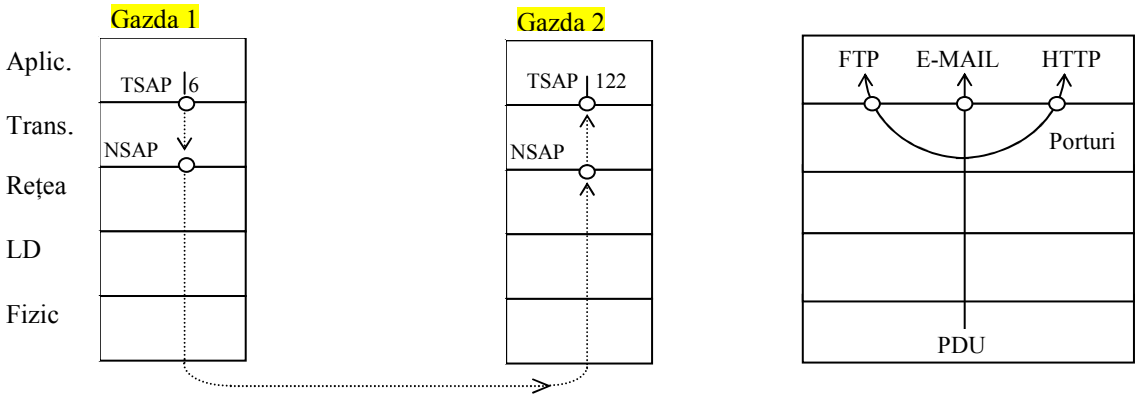



Fig. 5.1 Puncte de acces la servicii la nivel rețea şi transport

Stabilirea conexiunilor la nivel transport folosind soclurile TCP necesită un set de primitive specifice prin care **se crează socluri**, se **ascultă cererile** ce pot apărea (se urmăreşte activitatea lor), **se ataşează fluxuri de date, se transmit sau se recepționează date**, **se acceptă sau se refuză deschiderea unei conexiuni, se activează sau se blochează activitatea** pe un soclu etc.

*Tabelul 5.2 Primitive de soclu*

| **Primitiva** | **Parametrii** | **Valoarea întoarsă** | **Funcția** |
|---|---|---|---|
| **socket** ( ) | service type, protocol address format | socket descriptor or error code | *Ataşează o adresă locală la un soclu* |
| **bind** ( ) | host IP address + port number | success or error code | *Crează un nou punct de capăt al comunicației* |
| **listen** ( ) | socket descriptor, maximum queue length | success or error code | *Anunță disponibilitatea de a accepta conexiuni* |
| **accept** ( ) | socket descriptor, socket address | success or error code | *Blochează apelantul până la sosirea unei noi cereri* |
| **connect** ( ) | socket descriptor, local port number, destination port number, destination IP address, precedence, optional data (for example a user name and a password) | success or error code | *Procedura de stabilire a conexiunii* |
| **send** ( ) | socket descriptor, pointer to message buffer containing the data tosend, data length (in bytes), push flag, urgent flag, | success or error code | *Trimiterea datelor prin conexiunea stabilită* |
| **receive** ( ) | socket descriptor, pointer to a message buffer into which the data should be put, length of the buffer | success or end of file (EOF) or error code | *Recepția datelor prin conexiune* |
| **close** ( ) | socket descriptor | success or error code | *Închiderea soclului* |
| **shutdown**() | socket descriptor | success or error code | *Eliberarea conexiunii* |

### 5.3.1.Stabilirea conexiunii

Protocoalele de nivel transport sunt, în general, orientate pe conexiune. Cel mai potrivit este **algoritmul de stabilire a conexiunii în 3 paşi**. Acest nu necesită ca ambele părți să înceapă să transmită cu acelaşi număr de secvență. Gazda 1 alege un număr de secvență *x* pe care îl transmite gazdei 2 în pachetul *connection request.* Gazda 2 răspunde cu *connection accepted*, confirmă numărul de secvență *x* şi transmite şi numărul propriu de secvență *y.* Gazda 1 va confirma recepția numărului de secvență *y* în primul pachet de date trimis spre gazda 2. Dacă în timp va sosi la gazda 2 un pachet duplicat de la o cerere de conexiune mai veche, gazda 1 nu va şti în primul moment despre acest lucru. Gazda 2 îi va trimite confirmarea cererii că 1 a încercat să stabilească o conexiune, iar aceasta din urmă ştiind că nu a făcut o asemenea cerere, o va refuza. Cele două scenarii posibile descrise anterior se pot vedea în fig. 5.2.

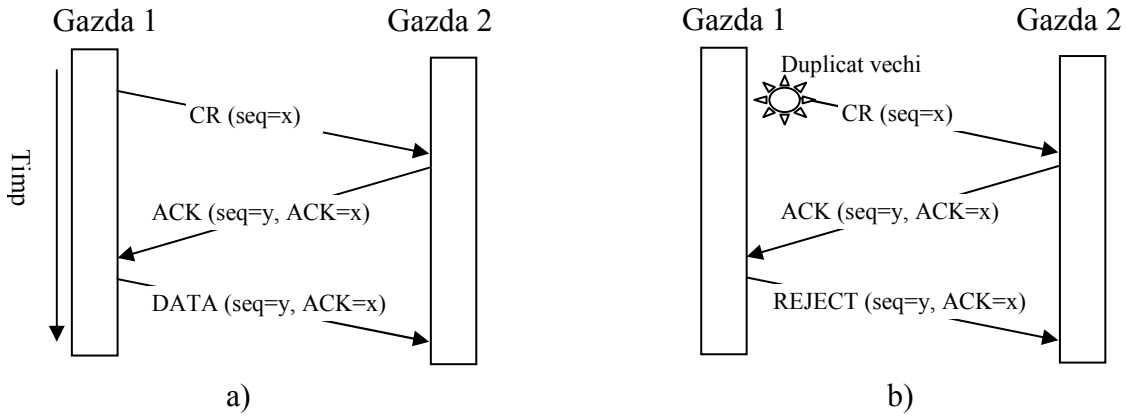



Fig. 5.2 Algoritmul stabilirii conexiunii în 3 paşi
a) cazul normal b) cazul cu pachete duplicate

Folosirea adreselor de nivel transport este utilă şi pentru multiplexarea la acest nivel. S-a precizat mai înainte că pe aceeaşi gazdă pot rula la un moment dat mai multe aplicații (procese utilizator) care sunt trimise în jos, la o singură adresă de rețea şi apoi spre nivelul fizic pentru a fi puse în canalul de comunicație. În rețeaua de comunicație ele pot urma aceeaşi cale sau pot avea căi diferite, după cum au aceeaşi destinație sau nu. Se spune s-a făcut o **multiplexare în sus** şi ea are loc la nivelul transport. Problema se poate pune şi în alt mod. O aplicație stabileşte o conexiune la nivel transport şi solicită o anumită rată de transfer. Se poate constata că o asemenea rată este insuficientă pentru aplicație şi atunci se mai cere deschiderea unei alte conexiuni pentru aceeaşi aplicație. Se spune atunci că s-a făcut o **multiplexare în jos** pentru a oferi bandă suficientă transferului de date.

Prin urmare, un protocol de transport trebuie să asigure, în principiu, următoarele servicii:

- **Stabilirea şi desfacerea conexiunilor;**
- **Adresarea;**
- **Multiplexarea;**
- **Controlul fluxului;**
- **Tratarea erorilor cap la cap;**
- **Refacerea după cădere a unei conexiuni**

### 5.3.2Un protocol de transport simplu

Cel mai simplu protocol de transport orientat pe conexiune se poate implementa folosind cele 5 primitive de transport descrise mai înainte. Aceste primitive sunt utilizate de nivelul transport pentru a trimite şi recepționa TPDU-uri. În general, entitatea transport poate să fie o parte a sistemului de operare a calculatorului gazdă, sau poate fi un pachet de funcții de bibliotecă rezidente în spațiul de adrese utilizator. În acest ultim caz, fiecare primitivă corespunde unei funcții de bibliotecă care execută un mic program. Fiecare funcție trebuie apelată cu parametrii corespunzători şi execuția ei întoarce un rezultat care poate fi un număr de conexiune, o stare etc. Parametrii funcțiilor de bibliotecă şi rezultatele întoarse sunt descrise în continuare.

```
id_conex=LISTEN(local)
id_conex =CONNECT(local,remote)
stare=SEND(connum,buffer,bytes)
stare=RECEIVE(connum,buffer,bytes)
stare=DISCONNECT(connum)
```

Primitiva **LISTEN** arată disponibilitatea serverului de a accepta cereri de conexiune la nivel transport. Utilizatorul primitivei este blocat până când se face o încercare de conectare la TSAP-ul specificat prin parametrul *local*.

Primitiva **CONNECT** are 2 parametrii, un TSAP local şi un TSAP aflat la distanță, adică 2 adrese de nivel transport. Dacă reuşeşte, ea întoarce *id_conex*, un număr nenegativ, folosit pentru a identifica conexiunea creată. Dacă nu reuşeşte, ea întoarce un număr negativ pus în *id_conex* prin care se poate descrie şi motivul eşecului (conexiune deja existentă, adresă locală incorectă, adresă îndepărtată inexistentă etc.).

Primitiva **SEND** trimite conținutul unui *buffer* specificat prin parametru ca mesaj pe conexiunea indicată de *id_conex*, eventual divizat în mai multe unități (*bytes*). Erorile posibile sunt întoarse în *status* şi ar putea fi: nu există conexiune, adresă de buffer incorectă, număr de bytes negativ etc.

Primitiva **RECEIVE** indică faptul că apelantul este în aşteptarea recepției datelor pe conexiunea indicată de *id_conex*, de la zona de memorie specificată de *buffer*, de lungime precizată prin parametrul *bytes*. În caz de eroare, rutina (funcția de bibliotecă) întoarce în *stare* un număr negativ care codifică tipul de eroare apărută.

Primitiva **DISCONNECT** pune capăt conexiunii specificată prin parametrul *id_conex.* În caz de succes întoarce 0 în variabila *stare*, altfel întoarce un număr negativ care codifică cauza insuccesului.

***Interfața cu nivelul rețea*** se face prin intermediul unor proceduri de tipul ***to_net*** şi ***from_net*****,** fiecare având câte 6 parametrii. Prima este folosită pentru a plasa TPDU-urile în jos spre nivelul rețea unde vor fi împachetete corespunzător, iar a doua este folosită pentru a extrage pachete de pe nivelul rețea pentru a fi prelucrate de nivelul transport. Semnificația celor 6 parametrii ai celor două proceduri este următoarea.

Primul parametru este identificatorul conexiunii *id_conex,* în corespondență biunivocă cu circuitul virtual de la nivelul rețea. Urmează bitul *Q,* (al doilea parametru) care dacă are valoarea 1 indică mesaje de contol şi bitul *M,* (al treilea parametru) care dacă are valoarea 1 indică faptul că mesajul continuă şi în pachetul următor. Al patrulea parametru este tipul pachetului, ales dintr-un set de 6 tipuri prezentate mai jos. La apelul *to_net*, nivelul transport completează toți parametrii pentru ca nivelul rețea să-i poată citi şi folosi; la apelurile *from_net*, nivelul rețea desasamblează un pachet sosit pentru entitatea transport.
Ultimii 2 parametrii (5 şi 6) sunt un pointer către zona de date şi un întreg care arată numărul de octeți de date.

*Tabelul 5.3*

| Tip de pachet | Semnificația |
|---|---|
| *Call Request* | Pentru stabilirea conexiunii |
| *Call Accepted* | Răspuns la *Call Request* |
| *Clear Request* | Pentru eliberarea conexiunii |
| *Clear Confirmation* | Răspuns la *Clear Request* |
| *Data* | Pentru transport date |
| *Credit* | Pachet de control pentru gestionarea ferestrei |

Un program în cod sursă scris în limbaj C++ pentru un protocol simplu de nivel transport este prezentat în [Tan].

## 5.4 Protocolul UDP (User Datagrame Protocol)

Acesta este un protocol simplu, neorientat pe conexiune, folosit pentru transmiterea rapidă de datagrame peste protocolul de rețea din Internet (IP). Este descris în RFC 768.

O datagramă UDP are un antet de 8 octeți şi informație utilizator.

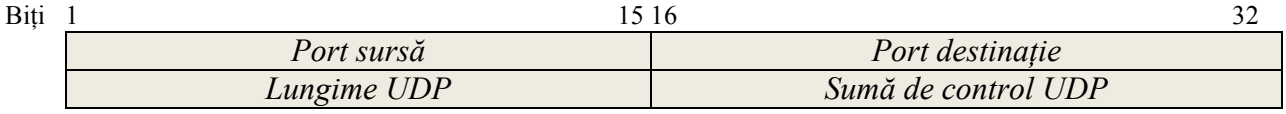



Fig. 5.3 Antetul UDP

Cele două porturi (*Port sursă* şi *Port destinație*) servesc pentru identificarea punctelor terminale sursă şi destinație. Fără aceste câmpuri, protocolul de transport nu ar şti ce să facă cu pachetele recepționate. Portul sursă este folosit de destinație atunci când trebuie să dea un răspuns sursei. Conținutul său este copiat în spațiul portului destinație din pachetul de răspuns. Câmpul *Lungime UDP* include antetul şi datele. Câmpul *Sumă de control* este opțional.

UDP este un protocol simplu, cu puține facilități. Nu realizează controlul fluxului, al erorilor, nu retransmite datagrame pierdute etc. El pur şi simplu oferă IP-ului un mijloc de multiplexare a proceselor (aplicațiilor) folosind porturile de nivel transport. Este utilizat în transferurile scurte de date, gen întrebare – răspuns în aplicațiile client - server. Un client trimite o cerere scurtă spre server şi aşteaptă un răspuns scurt. Dacă acesta nu vine într-un timp aşteptat, atunci repetă cererea.

Un exemplu tipic de utilizare este între un client şi serverul DNS (Domain Name System) pentru aflarea adresei IP corespunzătoare unui nume de gazdă, de exemplu *www.titu_maiorescu.ro*. Nu este nevoie de deschiderea unei conexiuni, nici de închidere, pentru un transfer a două mesaje scurte care traversează rețeaua.

Procedura de aflare a adresei IP a unui calculator gazdă, *get_IP_address(host_name)*, funcționează prin transmiterea unui pachet UDP spre serverul DNS şi aşteptarea răspunsului. Modelul se poate aplica tuturor programelor care apelează proceduri localizate pe stații aflate la distanță.

Un alt caz în care UDP este mult folosit este cel al aplicațiilor multimedia în timp real (voce, video –conferință, radio etc.). În acest scop s-a elaborat un protocol de transfer în timp real (**RTP – Real-Time Transport Protocol**), definit în RFC 1889. El este plasat deasupra UDP (în spațiul utilizator). Aplicațiile multimedia sunt trimise bibliotecii RTP. Aceasta le multiplexează, le codifică în pachete RTP şi le trimite printr-un soclu în nucleul sistemului de operare. La celălalt capăt al soclului, sunt generate pachete UDP şi apoi încapsulate în pachete IP.

RTP arată ca un protocol de aplicație rulând în spațiul utilizator, dar de fapt furnizează facilități de transport. Funcția sa principală este de multiplexare a mai multor fluxuri de date în timp real într-un flux de pachete UDP.

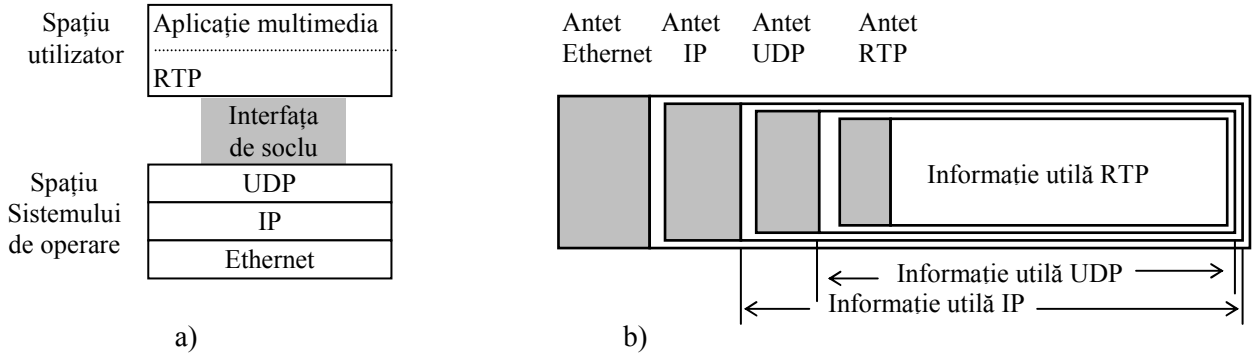



Fig. 5.4 a) Stiva de protocoale Ethernet/IP/UDP/RTP b) încapsularea

Fiecărui pachet RTP i se dă un **număr de ordine** pe baza căruia destinatarul poate face cel mult o aproximare prin interpolare în caz de pierdere de pachete, retransmisia nefiind o soluție practică. În consecință, **RTP nu are mecanisme de confirmare sau retransmisie.**

Totuşi în unele cazuri în care apar eşecuri repetate ale transmisiei şi creşte numărul de pachete eronate, RTP poate apela un alt protocol de control al transportului în timp real (**RTCP – Real time Transport Control Protocol**). Acesta nu transportă date utilizator, dar poate fi folosit pentru reducerea debitului la emisie, cerere suplimentară de bandă, semnalarea congestiei etc.

Antetul RTP conține 3 cuvinte de 32 biți şi eventual extensii.

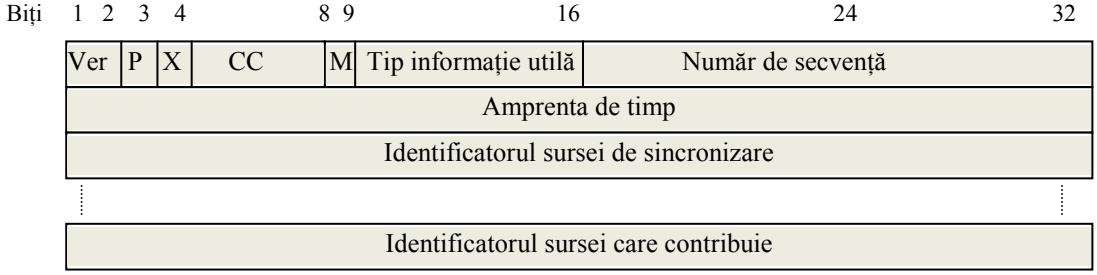



Fig. 5.5 Antetul RTP

***Ver(siune)*** arată versiunea RTP, în prezent fiind 2.
***P*** arată că pachetul este multiplu de 4 octeți.
***X*** arată antet extins.
***CC*** arată câte surse contribuabile sunt prezente (maxim 15).
***M*** bit de marcare specific aplicației (de exemplu începutul unui cadru video)..
***Tip informație utilă*** arată ce algoritm de codare a fost utilizat (necomprimat, MP3, etc.).
***Număr de secvență*** este un contor incrementat cu fiecare pachet RTP
***Amprenta de timp*** este stabilită de sursa fluxului pentru a şti când a fost făcut primul pachet.
***Identificatorul sursei de sincronizare*** spune cărui flux îi aparține pachetul. Este folosit pentru multiplexarea/demultiplexarea mai multor fluxuri de date într-un singur flux de pachete.
***Identificatorul sursei care contribuie*** este folosit atunci când se mixează mai multe surse audio.

## 5.5 TCP (Transmission Control Protocol)

TCP este protocolul principal care **permite transportul sigur al datelor de la un capăt la altul al unei conexiuni într-o rețea mare şi nesigură**, formată din mai multe rețele.

A fost **proiectat special pentru Internet** şi este larg folosit în acest scop.

**TCP** a fost definit oficial în **RFC 793**, dar a fost îmbunătățit şi actualizat prin mai multe recomandări (**RFC 1122, RFC 1323**).

Fiecare maşină care suportă TCP dispune de o entitate de transport TCP (modul software) fie ca proces utilizator, fie ca procedură de bibliotecă, fie ca parte a nucleului sistemului de operare.

Entitatea TCP acceptă blocuri compacte de date utilizator de la procesele locale pe care le împarte în fragmente ce nu depăşesc 64 K octeți (de regulă 1460 de octeți pentru a încăpea într-un singur cadru Ethernet împreună cu antetele TCP şi IP) şi expediază fiecare fragment ca o datagramă IP separată.

Când datagramele conținând informație TCP sosesc la destinație, entitatea TCP de pe maşina respectivă reconstruieşte fluxul original de date utilizator şi îl livrează aplicației de la care provine.

Deoarece protocolul IP peste care rulează de regulă TCP, nu oferă nici o garanție că datagramele vor fi livrate corect (ordine, pierdere), **TCP are sarcina de a verifica corectitudinea livrării şi poate decide retransmisia la nevoie**. Pe scurt, TCP-ul oferă fiabilitatea pe care toți utilizatorii o doresc, şi pe care IP-ul nu o asigură.

### 5.5.1 Adresele TCP

TCP este un protocol (serviciu) **orientat pe conexiune** care **asociază nişte puncte finale** numite **socluri** (sockets) **aflate pe maşini corespondente**. Fiecare soclu are un număr de soclu (adresă TCP) compusă din **adresa IP** a maşinii gazdă şi un număr local de 16 biți numit **port**.

Soclu=Adresa IP + Port

Pentru a obține **o conexiune TCP** trebuie stabilită explicit o **conexiune între un soclu de pe o maşină sursă şi un soclu de pe o maşină destinație**. Conexiunea este definită prin soclurile celor două capete.

Numărul de porturi care se pot folosi este $2^{16}$=65 536. Dintre acestea, cele mai mici de 265 se numesc porturi general cunoscute şi sunt rezervate serviciilor standard. Câteva porturi rezervate sunt date în Tabelul 5.4. Porturile nerezervate se pot atribui după bunul plac al utilizatorilor.

*Tabelul 5.4 CâtevapPorturi rezervate*

| Port | Protocol | Descriere |
|---|---|---|
| 21 | FTP | Transfer de fişiere |
| 23 | Telnet | Conectare la distanță |
| 25 | SMTP | E-mail |
| 69 | TFTP | Portocol simplu de transfer de fişier |
| 79 | Finger | Căutare informații despre utilizator |
| 80 | HTTP | World wide web |
| 110 | POP-3NNTP | Acces prin e-mail la distanță |
| 119 | | Ştiri USENET |

Câteva **caracteristici ale conexiunilor TCP**:

1. **Toate conexiunile sunt full duplex şi punct la punct.**
2. **TCP nu suportă difuzarea parțială sau totală.**
3. **O conexiune TCP este un flux de octeți şi nu un flux de mesaje.**
4. **Dimensiunile mesajelor nu se conservă de la un capăt la altul.** De exemplu, entitatea TCP emițătoare citeşte blocuri de câte 512 octeți pentru a fi trimise destinatarului. Transmiterea se poate face bloc cu bloc, sau concatenat ca 2 blocuri de câte 1024 octeți sau un bloc de 2048 octeți.
5. Fiecare segment TCP are un **număr propriu de secvență** de 32 de biți. Acesta este folosit atât pentru **secvențiere** (refacerea mesajului prin aşezarea segmentelor în ordine firească) cât şi pentru **confirmarea** segmetelor recepționate.
6. Baza funcționării TCP este **protocolul cu fereastră glisantă**. Atunci când un transmițător emite un segment, el porneşte un cronometru. Când segmentul ajunge la destinație, entitatea TCP receptoare trimite înapoi un segment cu informație utilă (dacă aceasta există, sau fără în caz contrar) şi numărul de secvență următor pe care îl aşteaptă, ceea ce constituie confirmarea recepției corecte. Dacă transmițătorul nu primeşte această confirmare într-un timp prestabilit, retransmite segmentul neconfirmat.
7. TCP are şi posibilitatea de a transmite date **în regim de urgență** prin întreruperea unui proces în derulare şi livrarea imediată a datelor acumulate.

Când o aplicație trimite date spre TCP pentru transfer, ele pot fi transferate imediat, sau pot fi stocate într-un buffer pentru a colecta o cantitate mai mare.

Un segment (datagramă) TCP conține antetul format dintr-o parte fixă de 20 de octeți şi o parte opțională de unu sau mai multe cuvinte de 32 de biți, plus încărcătura cu date utilizator (payload) de lungime opțională (inclusiv zero). Entitatea TCP este cea care decide cât de mare este încărcătura. Ea poate acumula într-un singur segment informație provnită de la mai multe scrieri (blocuri de date), sau poate fragmenta informația provenită de la o singură scriere în mai multe fragmente. Există două limite care restricționează dimensiunea unui segment. În primul rând un segment inclusiv antetul trebuie să încapă în dimensiunea maximă de 65 535 octeți de informație utilă admisă de IP. În al doilea rând fiecare rețea are o unitate maximă de transfer (**MTU - maximum transfer unit**) în care trebuie să încapă segmentul. De regulă, MTU este 1500 octeți dată de rețeaua Ethernet.

Formatul datagramei TCP este în fig. 5.6 iar semnificația câmpurilor din antet este explicată în continuare.

Câmpurile ***Port sursă*** şi ***Port destinație*** identifică punctele finale ale conexiunii şi constituie totodată un identificator al conexiunii.

Câmpurile ***Număr de secvență*** şi ***Număr de confirmare*** au semnificația funcțiilor de secvențiere şi confirmare descrise mai sus.

***Lungimea antetului TCP*** indică numărul de cuvinte de 32 de biți care sunt conținute în antetul TCP. Având 3 biți la dispoziție, rezultă că această lungime poate fi de maxim 8 cuvinte, adică spațiul opțional poate avea cel mult 3 cuvinte de 32 biți.

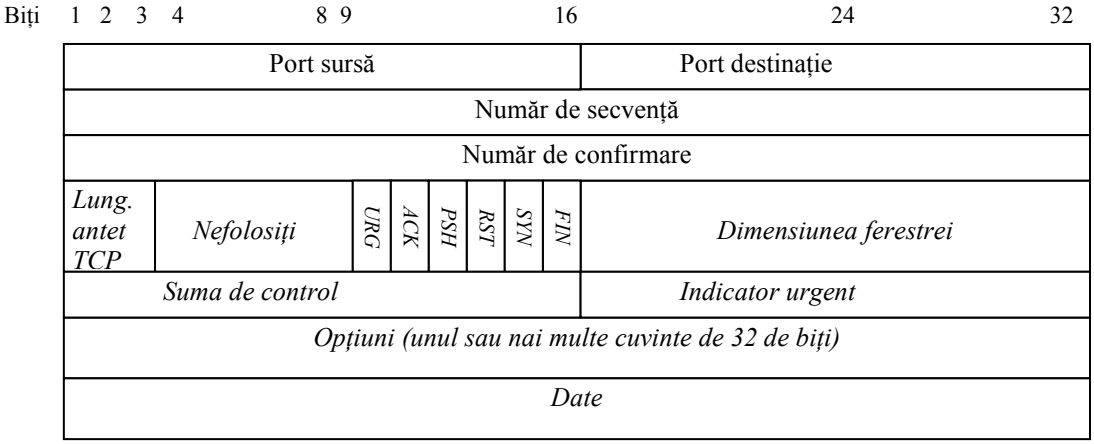



Fig. 5.6 Antetul TCP

Bitul ***URG*** poziționat pe 1 arată că ***Indicatorul urgent*** este valid. Acest indicator este folosit pentru a arăta deplasarea în octeți față de numărul curent în secvență la care se află informație urgentă. Această facilitate ține loc de mesaj de întrerupere.

Bitul ***ACK*** pe 1 indică faptul că *Numărul de confirmare* este valid. Dacă este poziționat pe 0, segmentul în discuție nu conține o confirmare si câmpul *Număr de confirmare* este ignorat.

Bitul ***PSH*** indică informația forțată, adică trebuie livrată aplicației îndată ce a fost recepționată, fără a mai fi memorată în buffere din rațiuni de eficiență.

Bitul ***RST*** este folosit pentru a desființa o conexiune care a devenit inutilizabilă din cauza unor defecțiuni ale mașinilor gazdă sau din alte motive. Este de asemenea utilizat pentru a refuza deschiderea unei conexiuni inițiată de un segment invalid de call request.

Bitul ***SYN*** este folosit pentru stabilirea unei conexiuni. Cererea de conexiune conține SYN=1 şi ACK=0, iar răspunsul la o astfel de cerere este confirmată prin combinația SYN=1 şi ACK=1.

Bitul ***FIN*** este folosit pentru a încheia o conexiune. El arată că transmițătorul nu mai are informații de transmis. Totuşi închiderea unei conexiuni este un proces lung, în care receptorul încă

mai poate primi date întârziate din rețea. De aceea, segmentele SYN şi FIN conțin numere de secvență pentru ca prelucrarea lor să se facă în ordinea firească. În TCP, fluxul de control este tratat prin ferestre glisante de dimensiune variabilă.

Câmpul ***Dimensiune fereastră*** indică numărul de octeți care pot fi trimişi începând de la octetul confirmat.

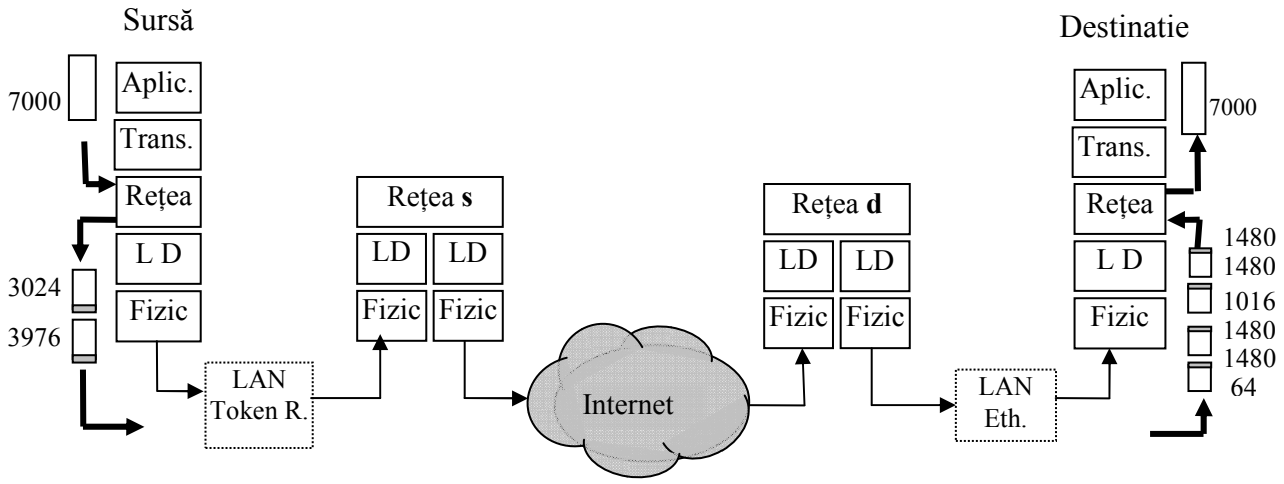
Câmpul ***Sumă de control*** conține suma de control calculată pentru antet, informație utilă şi pseudo-antet. Pseudo-antetul conține adresele IP ale sursei şi detinației, numărul de protocol (pentru TCP este 6) şi câmpul *lungime segment TCP*.

Câmpul ***Opțiuni*** a fost introdus pentru a permite adăugarea unor facilități suplimentare care nu au fost prevăzute în antetul obişnuit. Cea mai importantă opțiune este aceea de a specifica încărcarea unei maşini cu informație utilă TCP maximă pe care este dispusă să o accepte. Utilizarea unor segmente de date lungi este mai eficientă decât în cazul segmentelor scurte, mai ales în cazul unor maşini performante care pot procesa blocuri mari de date. În acest scop, la stabilirea conexiunii fiecare maşină anunță dimensiunea maximă acceptată şi aşteaptă de la parteneri să i se comunice acelaşi lucru. Dacă nici o maşină nu foloseşte această opțiune, mărimea implicită a segmentului de date utilizator este 536 octeți, astfel încât 536+20 antet=556, lumgime minimă pe care trebuie să o accepte orice maşină din Internet. Nici dimensiunile prea mari ale segmentelor nu sunt întotdeauna eficiente. De exemplu transmiterea unui segment de date de 64 Kocteți pe o linie $E_3$ de 34 Mbps durează 15 ms. Dacă întîrzierea de propagare dus- întors este 50 ms, valoare tipică pentru o linie transcontinentală, atunci emițătorul aşteptând confirmări va fi inactiv 70% din timp. În cazul unei legături prin satelit geostațional unde întârzierea de propagare este şi mai mare, eficiența transmisiei este şi mai mică.

### 5.5.3 Fragmentarea şi reasamblarea (Suplimentar)

*Fragmentarea şi reasamblarea se fac la nivelul transport şi rețea, pentru a permite transferul fragmentelor de date de lungimi mari prin rețele care nu suportă pachete de date cu lungimi mai mari decât o valoare dată, numită **MTU** (maximum transfer unit) şi care depinde de rețea. Procesul de fragmentare şi reasamblare, precum şi controlul acestora prin câmpurile antetului TCP se pot analiza în fig. 5.7.*

*Utilizatorul "Sursă" conectat intr-o rețea locală de tip Token Ring are de transmis un mesaj de 7000 octeți unui destinatar conectat într-o rețea locală de tip Ethenet. Deoarece rețeaua Token Ring nu suportă pachete mai lungi de 4000 de octeți (K), mesajul va trebui spart în două părți. Având în vedere că nivelul rețea mai adaugă un antet de 20 K, înseamnă că lungimea maximă a unui fragment ar putea fi 4000-20=3980K. În realitate toate fragmentele (mai puțin ultimul) trebuie să fie multiplu de 8 octeți, deci practic primul fragment fa avea 3976 K. Cel de al doilea fragment va fi de 3024 K. Cele două pachete sunt trimise ca atatre prin Internet până ajung la rețeaua destinație (rețeaua d). Aceasta fiind de tip Ethenet, lungimea maximă a pachetelor este 1500 K şi blocurile de date mai lungi vor trebui fragmentate corespunzător. Lungimea antetului fiind de 20 K, fragmentele din rețeaua Ethernet vor avea 1480 K. Prin urmare, cele două fragmente, de 3976 K şi 3024 K, vor fi împărțite în câte trei blocuri mai scurte, ca în fig.5.7. Informațiile privind lungimea totală a mesajului, poziția fragmentelor în mesajul inițial, dacă mai există sau nu fragmente etc. se scriu în antetul datagramelor IP. Pe baza lor se poate reconstitui mesajul inițial.*




| ***LAN Token ring***: | (*i*) | (*ii*) |
|---|---|---|
| Identification | 20 | 20 |
| Total length | 7000 | 7000 |
| Fragment offset | 0 | 497 |
| (User data) | 3976 | 3024 |
| M-bit | 1 | 0 |

| ***LAN Ethernet***: | (*i*) | (*ii*) | (*iii*) | (*iv*) | (*v*) | (*vi*) |
|---|---|---|---|---|---|---|
| Identification | 20 | 20 | 20 | 20 | 20 | 20 |
| Total length | 7000 | 7000 | 7000 | 7000 | 7000 | 7000 |
| Fragment offset | 0 | 185 | 370 | 497 | 682 | 867 |
| (User data) | 1480 | 1480 | 1016 | 1480 | 1480 | 64 |
| M-bit | 1 | 1 | 1 | 1 | 1 | 0 |

Fig. 5.7 Fragmentarea şi reasamblarea datagramelor în TCP-IP

## 5.6 TCP şi UDP în conexiuni fără fir

În principiu, protocoalele de transport ar trebui să fie independente de stiva de protocoale peste care rulează, adică să meargă la fel de bine şi pe rețele fir şi pe radio etc. În particular, aceste rețele ar trebui să fie transparente pentru TCP/IP. Totuşi acest lucru este parțial valabil. Protocoalele TCP/IP au fost gândite şi elaborate pentru canale pe fir, cu rate de eroare mici. Deşi ele pot rula şi pe canale radio, performanțele lor sunt foarte modeste din cauza implementării algoritmului de control al congestiei. Pierderea frecventă a pachetelor în rețele fără fir necesită retransmiterea acestora, ceea ce duce la apariția congestiei. TCP utilizează mai multe contoare de timp pentru a gestiona problema congestiei. Unul dintre cele mai folosite este **contorul de retransmisie** care stabileşte timpul maxim în care un pachet trebuie confirmat. Altul este **contorul de persistență** destinat prevenirii situațiilor de interblocare prin sondarea periodică a timpului de livrare a pechetelor. El permite stabilirea dimensiunii ferestrei glisante de transmisie. Altul este **contorul de menținere în viață** care verifică dacă o conexiune inactivă un timp mai este utilă, sau cealaltă parte a închis conexiunea. Atunci când expiră un contor, TCP încetineşte ritmul şi trimite pachete scurte, cu confirmări dese. Mai mult chiar, rețelele pot fi eterogene: o porțiune cu fir, alta radio etc. Luarea unei decizii de depăşire de timp este dificilă în aceste cazuri. S-ar putea ca o rețea WAN bazată pe TCP/IP să aibă doar un segment foarte scurt, la capăt, de tipul fără fir. O soluție aceptabilă în asemenea situații este segmentarea unei conexiuni TCP printr-o rețea eterogenă în mai multe conexiuni TCP pe zone de rețea omogene. În aceast situație, ratele de transmisie se pot regla independent.

Deşi UDP-ul nu suferă de aceleaşi probleme ca TCP, comunicaţia fără fir poate produce şi ea probleme în reţelele bazate pe UDP. Protocoalele de nivel aplicaţie care se bazează pe UDP se aşteaptă ca transmisia să fie foarte fiabilă. Cum reţelele fără fir suferă mult la acest capitol, refacerea pachetelor pierdute des poate conduce la un dezastru în ceea ce priveşte performanţele.

**Rezumat**

Rolul principal al nivelului transport este de a livra fluxuri de octeţi cap la cap, pe conexiuni sigure, într-un mod fiabil. El este pus în funcţiune prin primitive de serviciu care permit stabilirea conexiunii, utilizarea ei şi desfacerea. Stabilirea unei conexiuni se face prin protocolul în trei paşi. Conexiunile sunt definite prin adresele de nivel transport, cele mai uzuale fiind soclurile Berkeley. O altă sarcină importantă a nivelului transport este controlul congestiei. Aceasta apare atunci când rata de succesiune a pachetelor în reţea depăşeşte capacitatea de transport şi preluctate în noduri. Duplicarea pachetelor întîrziate complică stabilirea conexiunilor duce la autoînterţinerea congestiei. Controlul congestiei se face prin reducerea ratei de transmisie sau a ferestrei de transport folosind contoare de timp şi mecanisme de confirmare. Internetul are 2 protocoale de transport principale : UDP şi TCP, ultimul fiind de departe şi cel mai folosit. TPDU-urile se numesc segmente şi au un antet fix de 20 de octeţi pentru toate segmentele, şi o parte opţională. Partea opţională permite specificarea încărcării maxime cu informaţie utilă a TPDU, pe care o aceptă fiecare maşină. Segmentele pot fi fragmentate în PDU-uri mai mici de către rutere (la nivel reţea) dar trebuie reasamblate la recepţie. Parametrii de performanţă ai protocolului TCP depind pe de o parte de performanţele maşinilor (viteză de prelucrare, dimensiunile bufferelor), de capacitatea canalelor de comunicaţie şi de numărul de TPDU-uri existente în reţea. TPDU-uri multe şi scurte determină timpi mari de prelucrare în noduri, micşorând capacitatea de transfer efectivă a reţelei. Reţelele de arie largă şi eterogene complică utilizarea TCP.

## Întrebări de control

1. Care sunt locul şi rolul nivelului transport?
2. Ce se înţelege prin calitatea serviciilor asigurate de reţelele de calculatoare? Definiţi parametrii de calitate ai serviciilor.
3. Care sunt principalele servicii asigurate de nivelul transport? Explicaţi utilitatea lor.
4. Care este formatul adresei de nivel transport?
5. Ce este un punct de acces la servicii? Dar un port? Dar un soclu?
6. De ce este necesară combinarea unei adrese de reţea cu un TSAP?
7. Care este rolul primitivelor de transport? Exemplificaţi un mod de folosire a lor.
8. Ce se înţelege prin multiplexarea aplicaţiilor?
9. TDPU-urile încapsulează pachete sau invers? Argumentaţi.
10. Care sunt cele mai cunoscute porturi rezervate? În ce scop s-a făcut rezervarea?
11. De ce este necesar ca timpul maxim de viaţă al unui pachet să acopere nu numai dispariţia pachetului ci şi a confirmării sale?
12. Fragmentarea şi reasamblarea datagramelor sunt procese controlate de IP şi invizibile TCP-ului. TCP-ul mai are vreun rost în aceste procese? Aveţi în vedere posibilitatea soirii aleatoare a pachetelor.

13. Explicați de ce protocolul TCP este orientat pe conexiune deşi protocolul de sub el (IP) este neorientat pe conexiune, pachetele de nivel rețea aparținând aceluiaşi mesaj fiind rutate independent unul de celălalt?
14. TCP este un protocol cu fereastră glisantă cu confirmare. Precizați mecanismul de confirmare.
15. De ce UDP, spre deosebire de TCP, este considerat neorientat pe conexiune?
16. Un proces de pe maşina 1 a fost asociat portului *p* şi un proces de pe maşina 2 a fost asociat portului *q*. Este posibil ca între cele porturi să fie deschise mai multe conexiuni TCP simultane?