**Universitatea Tehnică a Moldovei**

**Lucrare de laborator 5**

**LA DISCIPLINA:** PROGRAMAREA IN RETEA

**CU TEMA:** Aplicație Client-Server UDP

**Student:**

Lupei Nicolae FI-171

**Profesor:**

Boldumac Oleg

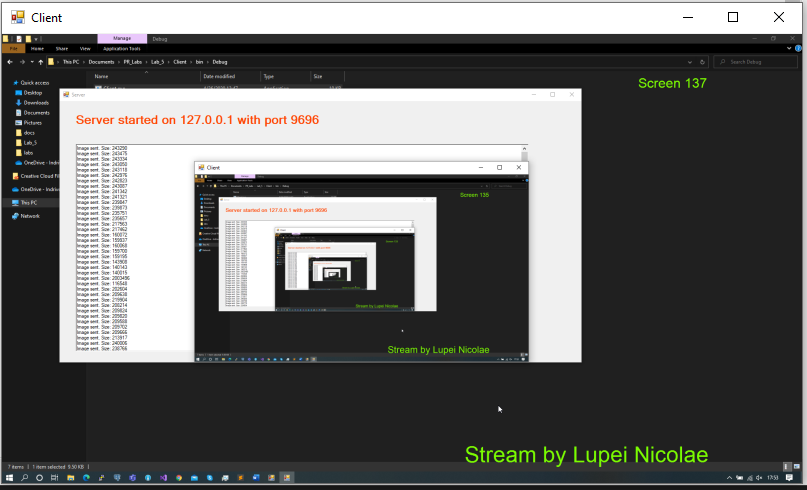
Chișinău – 2020

**Mersul lucrarii:**

**Server:**



**Client app:**



**Raspunsuri la intrebari:**

1. **Ce este un protocol orientat către tranzacții, fără conexiune ?**

User Datagram Protocol (sau UDP, în traducere liberă din engleză Protocolul Datagramelor Utilizator[1]) este un protocol de comunicație pentru calculatoare ce aparține nivelului Transport (nivelul 4 ) al modelului standard OSI.

Împreună cu Internet Protocol (IP), acesta face posibilă livrarea mesajelor într-o rețea. Spre deosebire de protocolul TCP, UDP constituie modul de comunicație fără conexiune. Este similar cu sistemul poștal, în sensul că pachetele de informații (corespondența) sunt trimise în general fără confirmare de primire, în speranța că ele vor ajunge, fără a exista o legătură efectivă între expeditor și destinatar.[2] Practic, UDP este un protocol ce nu oferă siguranța sosirii datelor la destinație (nu dispune de mecanisme de confirmare); totodată nu dispune nici de mecanisme de verificare a ordinii de sosire a datagramelor sau a datagramelor duplicate. UDP dispune, totusi, în formatul datagramelor, de sume de control pentru verificarea integrității datelor sau de informații privind numărul portului pentru adresarea diferitelor funcții la sursa/destinație.

Caracteristicile de baza ale UDP îl fac util pentru diferite aplicații.

orientat către tranzacții - util în aplicații simple de tip întrebare-răspuns cum ar fi DNS.

este simplu foarte util în aplicații de configurări, precum DHCP sau TFTP (Trivial FTP).

lipsa întârzierilor de retransmisie îl pretează pentru aplicații în timp real ca VoIP, jocuri online.

lucrează excelent în medii de comunicații unidirecționale precum furnizarea de informații broadcast, în servicii de descoperire (discovery services), sau în partajarea de informații către alte noduri (RIP).

1. **Ce tipuri de aplicații beneficiază în general de utilizarea protocolului UDP ?**

Aplicatii care beneficiaza de protocolul UDP sint aplicatiile care nu au nevoie ca sa primeasca mesaj daca pachetul a ajuns la destinatie sau nu, ca ex: SMTP

1. **De ce protocolul UDP nu garantează că datele vor fi transmise cu succes ?**

Protocolul UDP nu garanteaza daca datele au fost transmise fiindca nu exista o conexiune si clientul nu stie ce date trebuie sa primeasca, el doar asculta pe un port pentru a primi orice pachet fara a sti daca au mai fost anterior pachete sau nu, in asa caz aplicatia trebuie sa posede un set de reguli prin a notifica server-ul care a trimis ca pachetul a venit cu success.

1. **Diferența dintre blocking si non-blocking sockets**

În mod implicit, soclurile TCP sunt în modul „blocare”. De exemplu, când apelați recv () pentru a citi dintr-un flux, controlul nu este returnat în programul dvs. până când cel puțin un octet de date este citit de pe site-ul de la distanță. Acest proces de așteptare pentru apariția datelor este denumit „blocare”. Același lucru este valabil și pentru API-ul write (), API-ul connect (), etc. Când le executați, conexiunea se blochează până când operațiunea este completă.

Este posibil să setați un descriptor astfel încât să fie plasat în modul „care nu blochează”. Când este plasat în modul care nu blochează, nu aștepți niciodată finalizarea unei operații. Acesta este un instrument de neprețuit dacă trebuie să comutați între multe prize diferite conectate și doriți să vă asigurați că niciuna dintre ele nu provoacă „blocarea” programului.

Dacă apelați „recv ()” în modul care nu blochează, va returna orice date pe care sistemul le are în bufferul de citire pentru acea priză. Dar nu va aștepta aceste date. Dacă bufferul citit este gol, sistemul va reveni de la recv () spunând imediat `` „Operația s-ar bloca!” ''.

Același lucru este valabil și pentru API-ul send (). Când apelați send (), introduce datele într-un tampon și, după cum sunt citite de site-ul de la distanță, acestea sunt eliminate din buffer. Dacă tamponul va fi „complet”, sistemul va returna eroarea „Operațiunea ar bloca” data viitoare când încercați să-i scrieți.

Soclurile care nu blochează au un efect similar asupra acceptului () API. Când sunați acceptul () și nu există deja un client care să vă conecteze, acesta va returna „Operația ar bloca”, pentru a vă spune că nu poate finaliza acceptarea () fără a aștepta ...

API-ul de conectare () este puțin diferit. Dacă încercați să apelați connect () în modul care nu blochează și API-ul nu se poate conecta instantaneu, va returna codul de eroare pentru „Operation In Progress”. Când sunați din nou la conectare (), mai târziu, este posibil să vă spună „Operațiunea deja în desfășurare” pentru a vă anunța că încă încearcă să vă conectați, sau vă poate da un cod de returnare reușit, spunându-vă că conectarea a fost făcută.

Revenind la exemplul „browser web”, dacă ați introdus soclul care s-a conectat la serverul web în modul care nu blochează, puteți apoi să apelați connect (), să imprimați un mesaj spunând „conectarea la gazda www.floofy.com. .. "atunci poate face altceva, iar ei se întorc pentru a se conecta () din nou. Dacă conectarea () funcționează a doua oară, puteți imprima „Gazda contactată, așteptând răspunsul ...” și apoi începeți să apelați send () și recv (). Dacă conectarea () este încă pendinte, este posibil să verificați dacă utilizatorul a apăsat pe un buton „a anula” și, dacă da, apelați închidere () pentru a nu mai încerca să vă conectați.

Soclurile care nu blochează pot fi de asemenea utilizate în combinație cu API-ul select (). De fapt, dacă ajungeți într-un punct în care DORINȚI de fapt să așteptați datele de pe o priză care a fost marcată anterior drept „non-blocare”, puteți simula o înregistrare de blocare () doar apelând select () mai întâi, urmată de recv ( ).

1. **În protocolul TCP există Three Way Handshake, de ce în UDP nu există ?**

In TCP realizarea conexiunii presupune o secventă de 3 pachete transmise între cele

2 hosturi cunoscută sub numele de three-way handshake. Pentru stabilirea

conexiunii este necesar ca un host, numit Rețea, să aștepte pasiv realizarea

conexiunii. În același timp celălalt host, numit Client, va transmite un pachet prin care

cere realizarea conexiunii la adresa IP si la portul dorit.

Fiindca in UDP nu trebuie ca sa existe o conexiune , three-way handshake nu exista in UDP

1. **Numiti cele 2 apeluri de sistem necesare pentru a crea un server UDP**

Socket creează un nou descriptor de socket. Argumentele din apel specifică familia de protocoale folosite de aplicaţie (PF\_INET pentru TCP/IP) şi tipul serviciului (stream sau datagrama). Pentru un socket care foloseşte familia de protocoale Internet, tipul serviciului specifică dacă socket-ul foloseşte TCP sau UDP.

bind leagă o adresa IP locala şi portul protocolului la socket. Argumentele specifică descriptorul de socket şi adresa punctului de comunicaţie. Pentru protocoalele TCP/IP specificarea adresei se face folosind structura sockadd\_in, ce include adresa IP şi numărul de port al protocolului. Procesele server folosesc bind pentru a informa procesele client asupra portului unde se acceptă conexiuni.

1. **Care este rolul metodei bind() ?**

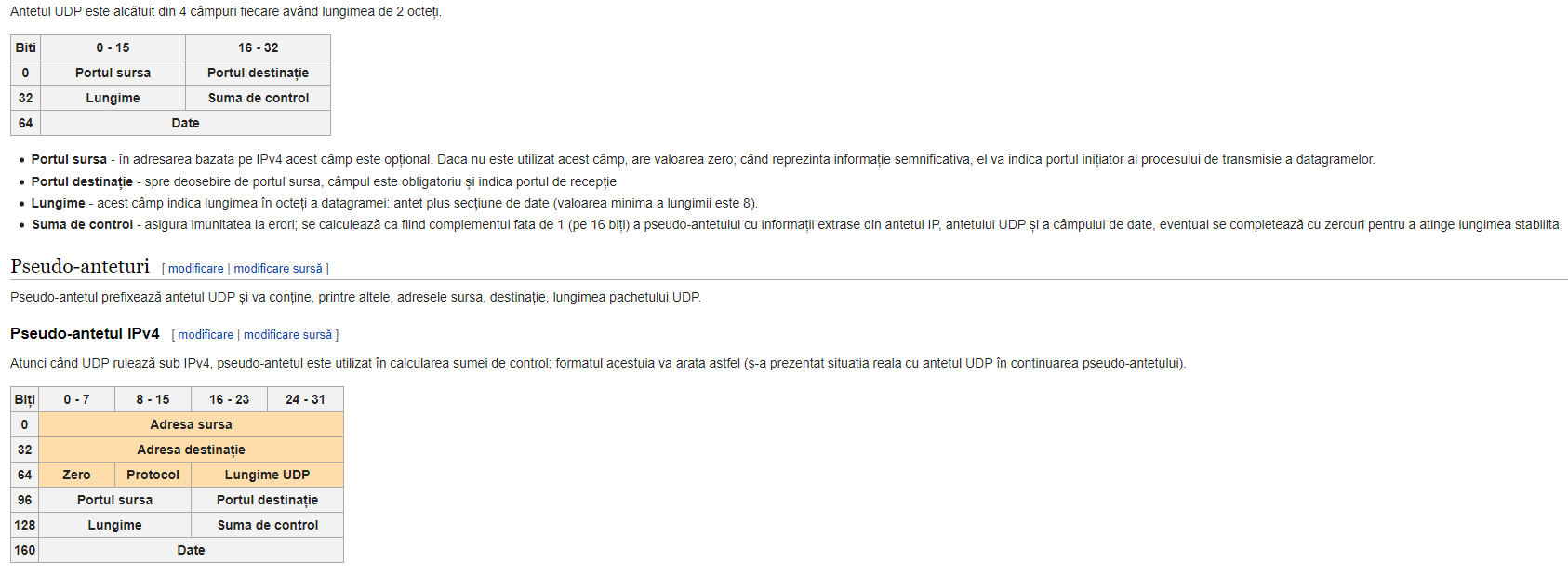
Bind leagă o adresa IP locala şi portul protocolului la socket. Argumentele specifică descriptorul de socket şi adresa punctului de comunicaţie. Pentru protocoalele TCP/IP specificarea adresei se face folosind structura sockadd\_in, ce include adresa IP şi numărul de port al protocolului. Procesele server folosesc bind pentru a informa procesele client asupra portului unde se acceptă conexiuni.

1. **Care este rolul metodelor sendto() și recvfrom() ?**

Sendto() transmite date prin conexiunea realizată. Datele sunt, de regulă, copiate în buffer-ele din nucleul SO, aplicaţia îşi continuă execuţia în timp ce datele sunt transportate prin reţea. Apelul este folosit de server pentru a transmite răspunsuri, iar de clienţi pentru a transmite cereri.

Recvfrom() citeşte datele ajunse la socket şi le depune într-un tampon utilizator. Dacă nu sunt disponibile date apelul se blochează. Procesele client şi server pot folosi read pentru a citi mesaje de la socket de tip UDP. Fiecare apel citeşte o datagrama. În cazul în care buffer-ul e prea mic surplusul este pierdut.

1. **Care este dimensiunea antetului unui pachet UDP în octeți ?**

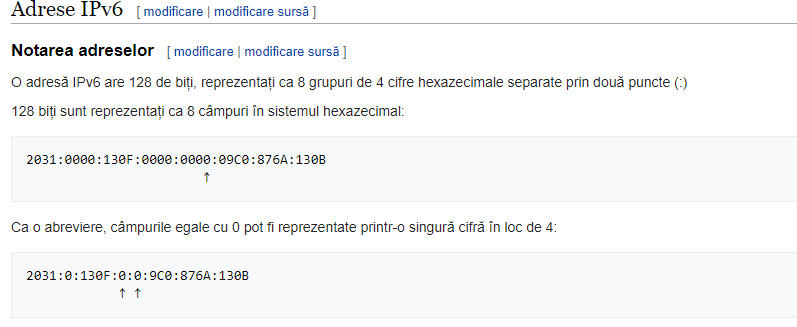


1. **Într-o conexiune UDP, clientul sau serverul trimite mai întâi datele ?**

Intr-o conexiune UDP datele pot sa trimita oricare din ei, fiindca nu se asteapta ca datele sa ajunga la destinatie.

1. Care este adresa de loopback IPv6 și care este rolul ei ?

IPv6 (prescurtare pentru engleză Internet Protocol version 6) este un protocol de nivel internet TCP/IP (respectiv nivel rețea din Modelul OSI) pentru rețelele cu comutare de pachete, inclusiv pentru Internet. Documentul ce specifică IPv6 este RFC 4294



1. **De unde știe un sistem de operare ce aplicație este responsabilă pentru un pachet primit din rețea ?**

Daca portul prin care este transmis pachetul nu este utilizat sau nu este blocat de firewall, in dependenta de aplicatiile care s-au inscris la porturi se distribue pachetul.

1. **Datele primite prin recvfrom() au întotdeauna aceeași dimensiune cu datele trimise cu sendto()?**

Nu au aceeasi dimensiune deoarece cel primit la recvfrom este mai mare fiindca contine mai multe detalii despre pachet.

1. **Este acceptabil să închei execuția programului dacă este detectată o eroare de rețea ?**

Eu cred ca nu, fiindca un program are mai multe functii nu numai cele legate de retea, plus sint programe care ruleaza in regim offline.

1. **De ce nu este folosit algoritmul Nagle în protocolul UDP ?**

Algoritmul Nagle - Algoritmul Nagle intarzie trimiterea de pachete mici pe retea bazanduse pe teoria ca vor mai aparea viitoare pachete curand. Daca asa se intampla acestea pot fi transmise ca o sigura unitate, astfel micsorand aparitia de suprasolicitare a protocolului.

Un exemplu var ajuta sa intelegeti de ce e important acesta. In primul rand cel mai primar headear din protocolul TCP/IP este de 40 octeti si poate fi mai mare daca sunt folosite campuri optionale. Acum considerati un program care trimite mai multe pachete mici de 2 Octeti. Fara algoritmul Nagle fiecare dintre aceste pachete va fi trimis separat cu cel putin 20 octeti de date peste. Evedent este o pierdere. Algoritmul Nagle contra-ataca acest efect prin intarzicerea transmiterii de pachete pentru un timp scurt, cum ar fi 200 millisec. Pachetul este

1. **Ce instrumente listează socket-urile UDP deschise în sistemele de operare Windows și Linux ?**

Pentru a enumera toate porturile deschise de pe un sistem Linux , puteți utiliza comanda netstat sau utilitatea ss după cum urmează.

De asemenea, este crucial să menționăm că comanda netstat a fost depășită și în schimb comanda ss și- a luat locul în afișarea statisticilor de rețea mai detaliate.

Windows: https://www.howtogeek.com/howto/28609/how-can-i-tell-what-is-listening-on-a-tcpip-port-in-windows/

1. **Același program poate folosi UDP și TCP ?**

Da, un program poate utiliza ambele protocoale, depinde de program si ce face el, daca programul este complex si are un algoritm complex care tine doar de TCP, nu poate fi interschimbat TCP sau UDP

1. **Diferența dintre aplicații UDP Unicast, Broadcast, și Multicast**

Diferența dintre Unicast, Broadcast și Multicast

Datele sunt transportate printr-o rețea prin trei metode simple, adică Unicast, Broadcast și Multicast. Deci, să începem să rezumăm diferența dintre aceste trei:

Unicast : de la o sursă la o destinație, adică unu la unu

Broadcast : de la o sursă la toate destinațiile posibile, adică One-to-All

Multicast : de la o sursă la destinații multiple, care afirmă un interes în primirea traficului, adică One-to-Many

Notă : Nu există o clasificare separată pentru aplicații de la mai mulți la mulți, de exemplu, videoconferință sau jocuri online, unde mai multe surse pentru același receptor și unde receptorii sunt adesea dubli ca sursele. Acest model de serviciu funcționează pe baza multicast-ului multiplu și din acest motiv nu necesită un protocol unic. Designul original multicast, adică RFC 1112, acceptă atât ASM (oricare-sursă-multicast) bazat pe modelul de servicii mult-la-multe, cât și SSM (multicast specific sursei) bazat pe un model unu-la-mulți.

1. **Ce face mai ușor multiplexarea cu UDP decît cu TCP ?**

Multiplexarea înseamnă multiprocesare și depinde foarte mult de prize care este pur și simplu o adresă IP + număr de port: IP: port # . Multiplexarea este folosită de ambele protocoale TCP și UDP pentru a segmenta mai multe solicitări de servicii.

Când un astfel de client PC1 solicită o pagină web de la serverul de găzduire al ccnahub, serviciul de protocol HTTP PC1 va utiliza portul 80 ca port de destinație, care este portul de ascultare al serverului web pentru serviciul HTTP. Rețineți că serverul de găzduire al ccnahub ar putea să ruleze și alte servicii, în afară de serviciul HTTP, precum e-mail, ftp, ssh și așa mai departe.

Dacă PC1 a solicitat servicii FTP și e-mail de la același server de găzduire, aplicația FTP a PC1, cum ar fi FileZilla, va folosi portul TCP 21 ca port de destinație, care este portul de ascultare al serviciului FTP pe serverul de găzduire pentru a transfera fișiere. Același concept este utilizat atunci când se folosește aplicația de e-mail, precum Outlook, PC1 va folosi portul TCP 25 ca port de destinație, care este portul de ascultare SMTP de la Serverul de găzduire pentru a trimite e-mailuri.

Prin urmare, atunci când serverul de găzduire al ccnahub primește mai multe solicitări de la același dispozitiv client folosind mai multe aplicații (Web, FTP și e-mail), Multiplexing Logic se conectează (indiferent de ce se folosește TCP sau UDP), toate cererile vor fi gestionate și vor răspunde fără Probleme.

1. **În protocolul UDP este un antet „Total length”, cum se calculează și care este rolul lui**

UDP este un protocol simplu, bazat pe datagram, de nivel de transport, care păstrează limitele mesajelor:

* Nu oferă corectarea erorilor, secvențierea, eliminarea duplicatelor, controlul debitului sau controlul congestiei.
* Poate furniza detectarea erorilor și include adevărata sumă de verificare de la *sfârșitul capătului* la nivelul transportului.
  + Câmpul **Checksum** ( [figure\_10-2.png](https://notes.shichao.io/tcpv1/figure_10-2.png) ) este de la capăt și este calculat peste pseudo-antetul UDP, care include câmpurile Adresă IP sursă și destinație din antetul IP. Astfel, orice modificare adusă acestor câmpuri (de exemplu, de către NAT) necesită o modificare a sumei de control UDP.
* Oferă funcționalitate minimă în sine, astfel încât aplicațiile care o utilizează au un control mare asupra modului în care sunt trimise și procesate pachetele. Aplicațiile care doresc să se asigure că datele lor sunt livrate sau secvențiate în mod fiabil trebuie să implementeze singure aceste protecții.
* Fiecare operație de ieșire UDP solicitată de o aplicație produce exact un datagrama UDP, ceea ce determină să fie trimis un datagramă IP.
  + Acest lucru este în contrast cu un protocol orientat în flux, cum ar fi TCP ( [Capitolul 15](https://notes.shichao.io/tcpv1/ch15/) ), unde cantitatea de date scrise de o aplicație poate avea o relație mică cu ceea ce este efectiv trimis într-un singur datagram IP sau cu ceea ce este consumat la receptor.

[ [RFC0768](https://tools.ietf.org/html/rfc768) ] este specificația oficială a UDP și a rămas ca un standard fără revizuiri semnificative mai mult de 30 de ani.

* UDP nu furnizează nicio corecție a erorilor așa cum s-a menționat: trimite datagramele pe care aplicația le scrie stratului IP, dar nu există nicio garanție că acestea vor ajunge vreodată la destinație.
* Nu există niciun mecanism de protocol care să împiedice traficul UDP de mare rată să afecteze negativ alți utilizatori ai rețelei.

Avantajele UDP \*

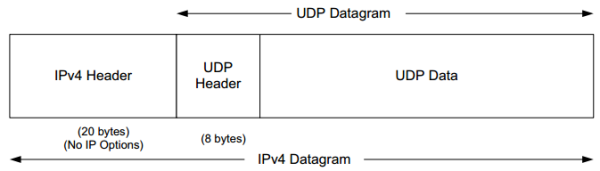
Având în vedere această lipsă de fiabilitate și protecție, am putea fi tentați să concluzionăm că nu există deloc beneficii în utilizarea UDP. Acest lucru nu este adevărat. UDP are următoarele avantaje:

* Datorită caracterului său de conexiune, are mai puține aeriene decât alte protocoale de transport.
* Operațiunile de difuzare și multicast ( [capitolul 9](https://notes.shichao.io/tcpv1/ch9/) ) sunt mult mai simple folosind un transport fără conexiune, cum ar fi UDP.
* Capacitatea unei aplicații de a alege propria unitate de retransmisie poate fi o considerație importantă.

Încapsularea unei diagrame de date UDP \*

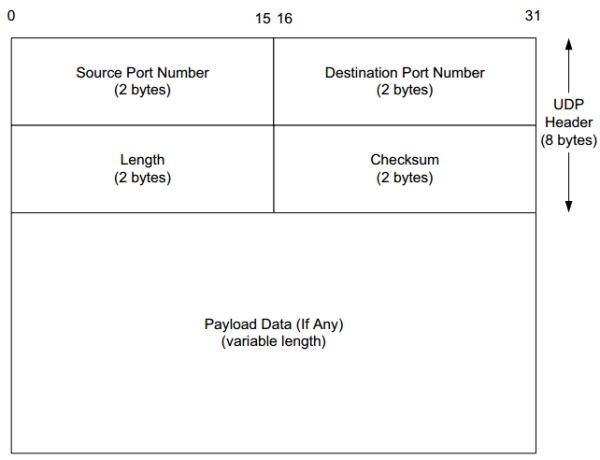
Figura următoare arată încapsularea unui datagram UDP ca un singur datagrama IPv4.

* Încapsularea IPv6 este similară, dar alte detalii diferă ușor ( [Secțiunea 10.5](https://notes.shichao.io/tcpv1/ch10/#udp-and-ipv6) ).
* Câmpul **Protocol** IPv4 are valoarea 17 pentru a indica UDP.
* IPv6 utilizează aceeași valoare (17) în câmpul **Next Header** .
* [Mai târziu în acest capitol este](https://notes.shichao.io/tcpv1/ch10/#ip-fragmentation) descris ce se întâmplă atunci când mărimea datagramei UDP depășește dimensiunea MTU și datagrama trebuie fragmentată în mai multe pachete IP-layer.

[](https://notes.shichao.io/tcpv1/figure_10-1.png)

Antetul UDP

Următoarea figură arată datagrama UDP, inclusiv sarcina utilă și antetul UDP (care are întotdeauna o dimensiune de 8 octeți):

[](https://notes.shichao.io/tcpv1/figure_10-2.png)

* Numerele de port acționează ca căsuțe poștale și ajută o implementare a protocolului să identifice procesele de trimitere și primire ( [Capitolul 1](https://notes.shichao.io/tcpv1/ch1/) ). Sunt pur *abstracte* : nu corespund nici unei entități fizice de pe o gazdă. În porturile UDP numerele sunt pozitive pe 16 biți:
  + Numărul portului sursă este opțional; acesta poate fi setat la 0 dacă expeditorul datagramei nu necesită niciodată răspuns.

Protocoalele de transport, cum ar fi TCP și UDP, și SCTP [RFC4960] folosesc numărul portului de destinație pentru a ajuta demultiplexul datelor primite de la IP. Deoarece IP demultiplexează datagrama IP de intrare la un anumit protocol de transport bazat pe valoarea câmpului **Protocol** din antetul IPv4 sau câmpul **Next Header** din antetul IPv6, acest lucru înseamnă că numerele de port pot fi făcute independent între protocoalele de transport. Adică, numerele de port TCP sunt folosite doar de TCP, iar numerele de port UDP numai de UDP și așa mai departe. O consecință simplă a acestei separații este aceea că două servere complet distincte pot utiliza același număr de port și adresă IP, atât timp cât utilizează protocoale de transport diferite.

În ciuda acestei independențe, dacă un serviciu binecunoscut este furnizat (sau poate fi furnizat) atât de TCP, cât și de UDP, numărul portului este alocat în mod normal să fie același pentru ambele protocoale de transport. Acest lucru este pur pentru comoditate și nu este necesar de protocoale. Consultați [ [IPORT](http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml) ] pentru detalii despre cum sunt alocate formal numerele de port.

* Câmpul UDP **Length** este lungimea antetului UDP și a datelor UDP în octeți. Valoarea minimă pentru acest câmp este 8, cu excepția cazului în care UDP este utilizat cu jumbograme IPv6 (vezi Secțiunea 10.5). Trimiterea unei dateagrame UDP cu 0 octeți de date este acceptabilă, deși rară.
  + Câmpul Lungime UDP este redundant; Antetul IPv4 conține lungimea totală a datagramei ( [Capitolul 5](https://notes.shichao.io/tcpv1/ch5/) ), iar antetul IPv6 conține lungimea sarcinii utile. Lungimea unui datagram UDP / IPv4 este apoi lungimea totală a datagramei IPv4 minus lungimea antetului IPv4. Lungimea unei dateagene UDP / IPv6 este valoarea câmpului Lungime de sarcină conținută în antetul IPv6 minus lungimile oricăror anteturi de extensie (cu excepția cazului în care se utilizează jumbograme). În ambele cazuri, câmpul Lungime UDP ar trebui să corespundă cu lungimea calculată din informațiile stratului IP.