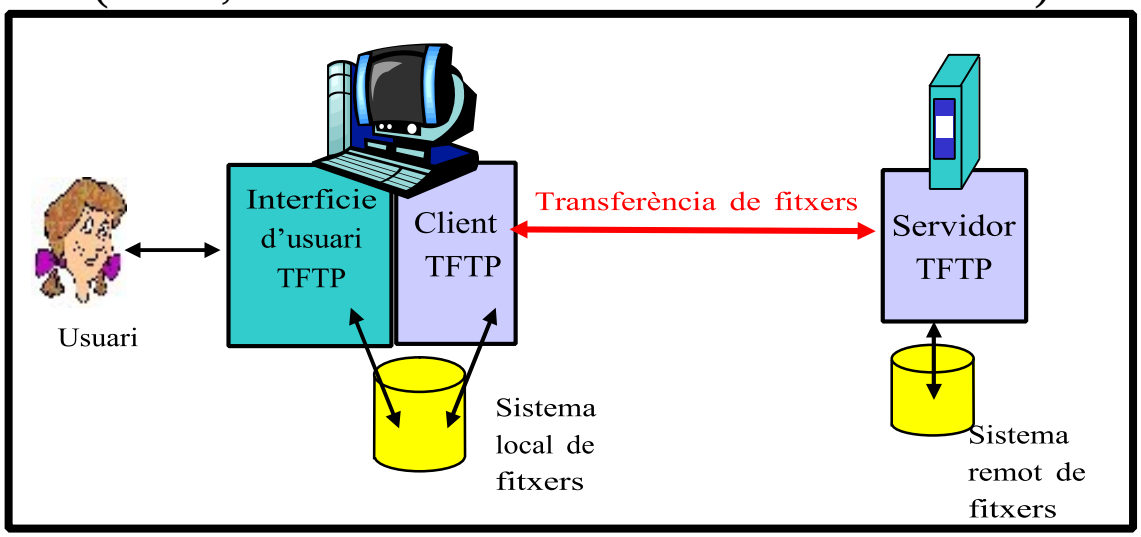
Xarxes de Computadors

Pràctica 1

Protocol Trivial de Transferència de Fitxers

(TFTP, Trivial File Transfer Protocol. RFC 1350)



1. Introducció

Aquesta pràctica té per objectiu la programació d'un protocol estàndard de Internet, el protocol triat és el TFTP (Trivial File Transfer Protocol, https://www.ietf.org/rfc/rfc1350.txt), i la seva implementació en Python ens assegurarà la transferència fiable de fitxers.

TFTP és molt més senzill que el estàndard de Internet FTP (File Transfer Protocol, RFC 959). No hi ha mecanismes per controlar l'accés o la identificació d'usuaris. Per tant, TFTP esta nomes indicat per accedir a directoris públics. Degut a la seva simplicitat, TFTP s'implementa fàcilment i de forma compacta. Per exemple, alguns dispositius sense disc utilitzen TFTP per descarregar el codi executable per arrancar, com és el cas de la descarrega d'imatges del Sistema Operatiu dels equips en el nostre laboratori, el protocol s'emmagatzema en la memòria EPROM degut a la seva simplicitat.

TFTP s'executa per sobre del protocol UDP (User Datagram Protocol, RFC 768). L'entitat TFTP que inicia la transferència ho fa enviant una sol·licitud per part del client de lectura o d'escriptura en un segment UDP al port 69 de servidor destí. Aquest port es reconeix per part del mòdul UDP com l'identificador del mòdul TFTP. Mentre dura la transferència cada extrem utilitza un identificador per la transferència (TID, Transfer IDentifier) com el seu número de port.

En aquesta pràctica tractarem sobre alguns dels temes més importants de l'assignatura de Xarxes de Computadors, com són els protocols de nivell de transport, els protocols ARQ (Automatic Repeat reQuest) i la programació de sockets, així com l'estudi de la transmissió de dades per un canal fiable i les seves característiques.

L'avaluació de la pràctica tindrà una part obligatòria que consistirà en la implementació del protocol TFTP segons la norma RFC 1350, i amb la opció de mida de camp de DADES segons la norma RFC 1783, per tant deurà funcionar contra els programes existents en la distribució TCP/IP del Sistema Operatiu del nostre computador. També s'avaluarà una part de millores de la part obligatòria, que consistirà en la implementació d'un protocol de Transmissió Continua pel Control de Flux i/o amb la opció de temps d'espera (time_out) segons la norma RFC 1784.

El lliurament de la pràctica es realitzarà el dia 11 de maig a la classe de Pràctiques, on es tindrà que demostrar el seu correcte funcionament, caldrà lliurar també els Manuals d'Usuari (funcionament) i del Manual del Programador (disseny i codi).

2. Model de Comunicacions Client/Servidor.

Una aplicació de sockets està formada per un codi que s'executa en ambdós extrems de la comunicació. El programa que inicia la transmissió s'anomena Client. El Servidor, per la seva banda, és un programa que espera passivament l'arribada de connexions entrants, procedents de clients remots. Les aplicacions de servidor es carreguen generalment durant l'arrencada del sistema i escolten activament si hi ha connexions entrants en el seu port ben conegut. Llavors les aplicacions client intenten connectar amb el servidor i es produeix un intercanvi de missatges a nivell de transport. Quan finalitza la sessió, normalment és el client qui conclou la connexió.

3. Sockets, Descriptors de Sockets, Ports i Connexions.

Els sockets són els punts finals de comunicació els quals son identificats mitjançant els corresponents descriptors de socket, o paraules de llenguatge natural que descriuen l'associació entre el socket i una determinada màquina o aplicació. Una connexió (o parella de sockets) consta d'una parella d'adreces IP que es comuniquen entre si, així com d'una parella de ports, on el nombre de port és un enter positiu de 32 bits que normalment s'escriu en forma decimal. Alguns nombres de ports de destinació són ben coneguts i indiquen el tipus de servei al que ens connectem. Els protocols TCP/IP requereix que les aplicacions que utilitzin ports ben coneguts es comuniquin entre si. Això es fa perquè les aplicacions clients suposin que la corresponent aplicació servidor està escoltant en el port ben conegut associat a l'aplicació. Per exemple, el nombre de port per a HTTP, que és el protocol emprat per a transferir pàgines HTML a través de la World Wide Web és el port 80 de TCP (Transport Control Protocol). Per omissió, els navegadors Web intenten obrir una connexió amb el port TCP 80 de host de destinació tret que s'especifiqui algun altre port en la URL (com el 8000 o el 8080).

El port identifica un punt de connexió dintre de la pila local (això és, el port nombre 80 s'utilitza generalment per a un servidor Web). Un socket identifica una parella formada per una adreça IP i un nombre de port (això és, el port 192.168.1.20:80) seria el port del servidor Web del host 192.168.1.20, els dos, en conjunt, es consideren un socket. Una parella de sockets identifica els quatre components (adreça i port d'origen, adreça i port de destinació). Com els ports ben coneguts són exclusius, de vegades s'utilitzen per a fer al·lusió a una determinada aplicació de qualsevol host que pugui estar executant l'aplicació. Si s'empra la paraula socket, no obstant això, s'estarà al·ludint a una aplicació concreta d'un host concret. Les connexions o parelles de sockets, denoten connexions de sockets entre dos sistemes específics que s'estan comunicant. TCP permet múltiples connexions simultànies que impliquen al mateix nombre de port local, sempre que les adreces IP o els nombres de port remots siguin distints per a cada connexió. Els nombres de port es divideixen en tres rangs:

- Els ports del 0 al 1023 són ben coneguts. S'associen a serveis de forma estàtica. Per exemple, els servidors HTTP sempre admetran sol·licituds en el port 80.
- Els nombres de port del 1024 al 49151 estan registrats. S'utilitzen per a múltiples propòsits.
- Els ports dinàmics i privats són els que van del 49152 al 65535, i no se'ls han d'associar serveis.

En realitat, les màquines comencen a assignar ports "dinàmics" a partir del 1024. Si es vol desenvolupar un protocol o aplicació que requereixi l'ús d'un enllaç, socket, port, protocol, etc., es necessari posar-se en contacte amb la Internet Assigned Numbers Authority (Autoritat d'Internet per a l'Assignació de Nombres o IANA) per a rebre una assignació de nombre de port. L'especificació oficial que enumera les assignacions de nombres de ports es troba en la URL https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml, que ens indica la llista de nombres de ports amb informació sobre les aplicacions associades.

Tant en UNIX com en Windows, es pot emprar l'ordre netstat per a comprovar l'estat de tots els sockets locals actius. La següent taula és un exemple del resultat de netstat.

Protocol	Direcció local	Direcció remota	Estat
TCP	SHANNON:1089	baym-cs157.msgr.hotmail.com:1863	ESTABLISHED
TCP	SHANNON:1214	si2ks.eupvg.upc.es:microsoft-ds	ESTABLISHED
TCP	SHANNON:1233	web096.halls.umist.ac.uk:3255	ESTABLISHED
TCP	SHANNON:1245	8.Red-81-33-150.pooles.rima-tde.net:2666	TIME_WAIT
TCP	SHANNON:1272	ool-182f2046.dyn.optonline.net:6348	ESTABLISHED
TCP	SHANNON:1279	c-24-3-169-8.client.comcast.net:6350	ESTABLISHED
TCP	SHANNON:1282	ip-66.net-81-220-108.rev.numericable.fr:6346	CLOSING
TCP	SHANNON:1302	trfl-unfiltered-164.wireless.wiktel.com:6346	FIN_WAIT_1
TCP	SHANNON:1313	dsl-209-205-172-105.i-55.com:6348	ESTABLISHED
TCP	SHANNON:1317	c-24-18-252-38.client.comcast.net:6346	CLOSING

4. Programació de Sockets

Una aplicació de xarxa consta d'un parell de programes: el programa Client i el programa Servidor. Quan s'executen aquest dos programes, es crea un procés client i un procés servidor, i aquest dos processos es comuniquen entre sí llegint i escrivint en sockets. Quan es crea una aplicació de xarxa, la principal tasca del desenvolupador és escriure el codi dels dos programes Client i Servidor.

Existeixen dos tipus d'aplicacions client/servidor. Un tipus d'aplicació és l'aplicació client/servidor que és una implementació d'un protocol estàndard definit en una RFC. Per aquesta implementació, els programes client i servidor han de ser conformes amb les regles descrites a la RFC. Per exemple, el programa client podria ser una implementació del client TFTP, definit en la RFC 1350, i el programa servidor una implementació del servidor TFTP, també definit en la RFC 1350. Si un desenvolupador escriu el codi del programa client, un altra desenvolupador independent escriu el codi del programa servidor, i tots dos segueixen cuidadosament les regles de la RFC, llavors els dos programes seran capaços d'interoperar. De fet, la majoria de les aplicacions de xarxa d'avui dia involucren la comunicació entre programes client/servidor que han estat creats per desenvolupadors independents (per exemple, un navegador Chrome que es comunica amb un servidor web Apache, o un client FTP en un PC que carrega un fitxer en un servidor FTP UNIX). Quan un programa client o servidor implementa un protocol definit en una RFC (com és el cas d'aquesta pràctica), té que utilitzar el nombre de port associat al protocol (per TFTP, el port 69).

L'altra tipus d'aplicacions client/servidor és el de les aplicacions propietàries. En aquest cas, els programes client i servidor no tenen necessariament que seguir cap RFC existent. Un únic desenvolupador (o equip de desenvolupament) crea tots dos programes client/servidor, tenint tot el control del codi. Donat que el codi no implementa cap protocol de domini públic, d'altres desenvolupadors independents no seran capaços de construir codi que interoperi amb les aplicacions. Quan es desenvolupa una aplicació propietària, el desenvolupador ha de tenir cura de no utilitzar cap dels números ben coneguts de ports definits a les RFC.

Durant la fase de desenvolupament, una de les primeres decisions que ha de prendre el desenvolupador és si l'aplicació va sobre TCP o sobre UDP. Recordeu que TCP es orientat a connexió, i que proporciona un canal fiable de flux de bytes, a través del qual flueixen les dades entre els sistemes finals. UDP és sense connexió, i envia paquets independents de dades entre dos sistemes finals, sense cap garantia d'entrega.

4.1. Programació de Sockets amb TCP

Els processos que s'executen en màquines distintes es comuniquen entre si enviant missatges a sockets. Cada procés és anàleg a una casa, i el socket del procés és anàleg a una porta. Com es mostra en la Figura 1, el socket és la porta entre el procés d'aplicació i TCP. El programador de l'aplicació té control complet sobre el costat de la capa d'aplicació del socket, però té un control petit sobre el costat de la capa de transport (com a molt, el programador de l'aplicació pot fixar alguns pocs paràmetres TCP, com la grandària màxima del búffer i de segment). Analitzant la interacció dels programes client i servidor, el client s'encarrega d'iniciar el contacte amb el servidor. Perquè el servidor pugui reaccionar al contacte inicial del client, el servidor ha d'estar preparat. Això implica dues coses. La primera és que el programa servidor no pot estar inactiu, ha d'estar executant-se com un procés abans que el client intenti iniciar el contacte. La segona és que el programa servidor ha de tenir algun tipus de porta (això és, socket) que rebi el contacte inicial del client que s'executa sobre una màquina arbitrària. Utilitzant l'analogia casa/porta per a procés/socket, farem de vegades referència al contacte inicial del client com cridant a la porta.

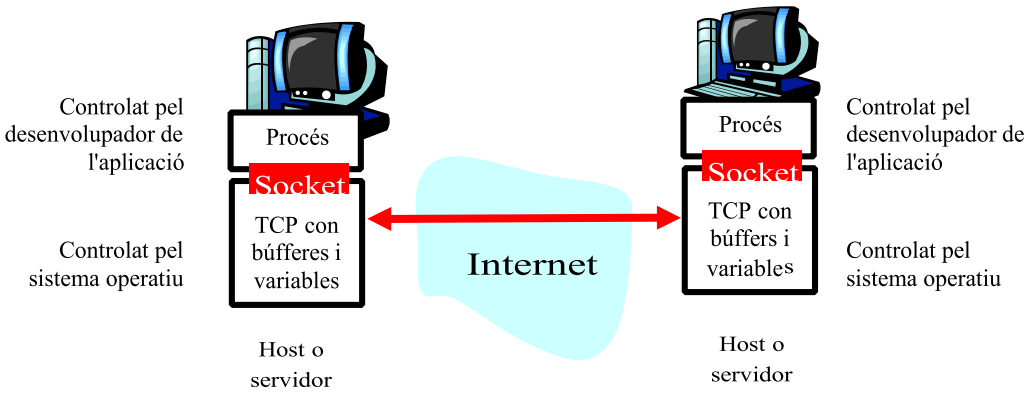


Figura 1. Processos que es comuniquen a través de sockets.

Quan el procés servidor està executant, el procés client pot iniciar una connexió TCP amb el servidor. En el programa client, això es fa creant un socket. Quan el client crea el seu socket, especifica l'adreça del procés servidor, concretament l'adreça IP del servidor i el nombre de port del procés. Una vegada creat el socket, TCP inicia en el client una sincronització en tres passos, i s'estableix una connexió TCP amb el servidor. La sincronització en tres passos és completament transparent per als programes client i servidor. Durant la sincronització en tres passos, el procés client crida a la porta del procés servidor. Quan el servidor escolta la petició, crea una nova porta (això és, un nou socket), que es dedica a aquest client particular. En el següent exemple, la porta de acollida és un objecto ServerSocket, al que cridarem socketAcogida. Quan un client crida aquesta porta, el programa invoca el mètode accept( ) de

socketAcogida, el que crea una nova porta per al client. AL final d'aquesta fase de sincronització existeix una connexió TCP entre el socket del client i el nou socket del servidor. És per això pel que ens referim a aquest nou socket com el socket de connexió del servidor. Des de la perspectiva de l'aplicació, la connexió TCP és una canonada virtual directa entre el socket del client i el socket de connexió del servidor. El procés client pot enviar bytes arbitraris sobre el socket; TCP garanteix que el procés servidor rebrà (per mitjà del socket de connexió) cada byte en l'ordre d'enviament. Més encara, de la mateixa forma que la gent pot sortir i entrar per la mateixa porta, el procés client també pot rebre bytes del seu socket, i el procés servidor també pot enviar bytes sobre el seu socket de connexió. Això s'il·lustra en la Figura 2. El desenvolupament d'aplicacions client/servidor també és denominat programació de sockets, donat el paper central que juguen els sockets en les aplicacions client/servidor. Abans de presentar l'exemple d'aplicació client/servidor, és interessant discutir la noció de flux. Un flux és una seqüència de caràcters que flueix a o des d'un procés. Cada flux és un flux d'entrada, o un flux de sortida pel procés. Si el flux és d'entrada, llavors ha d'estar lligat a alguna font d'entrada pel procés, com a l'exemple l'entrada estàndard (el teclat), o un socket sobre el qual flueixen les dades des de Internet. Si el flux és de sortida, llavors ha d'estar lligat a qualsevol font de sortida pel procés, com per exemple la sortida estàndard (el monitor), o un socket des del qual flueixen les dades a Internet.

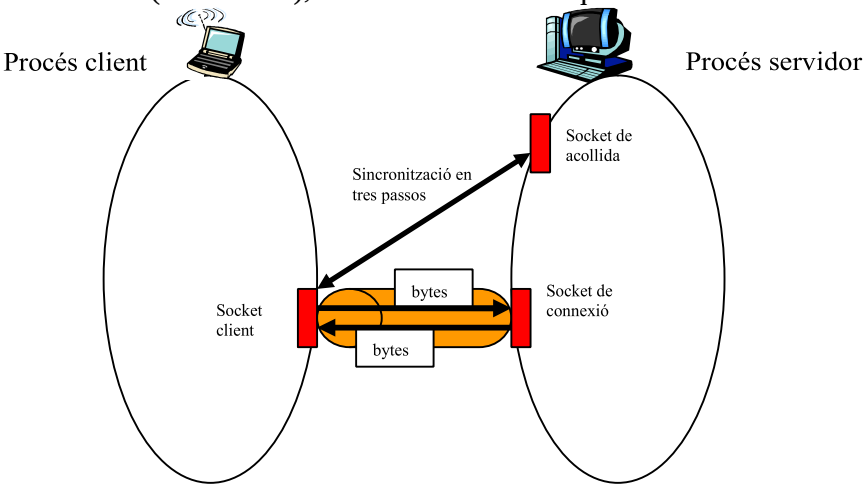


Figura 2. Socket client, socket de acollida, i socket de connexió.

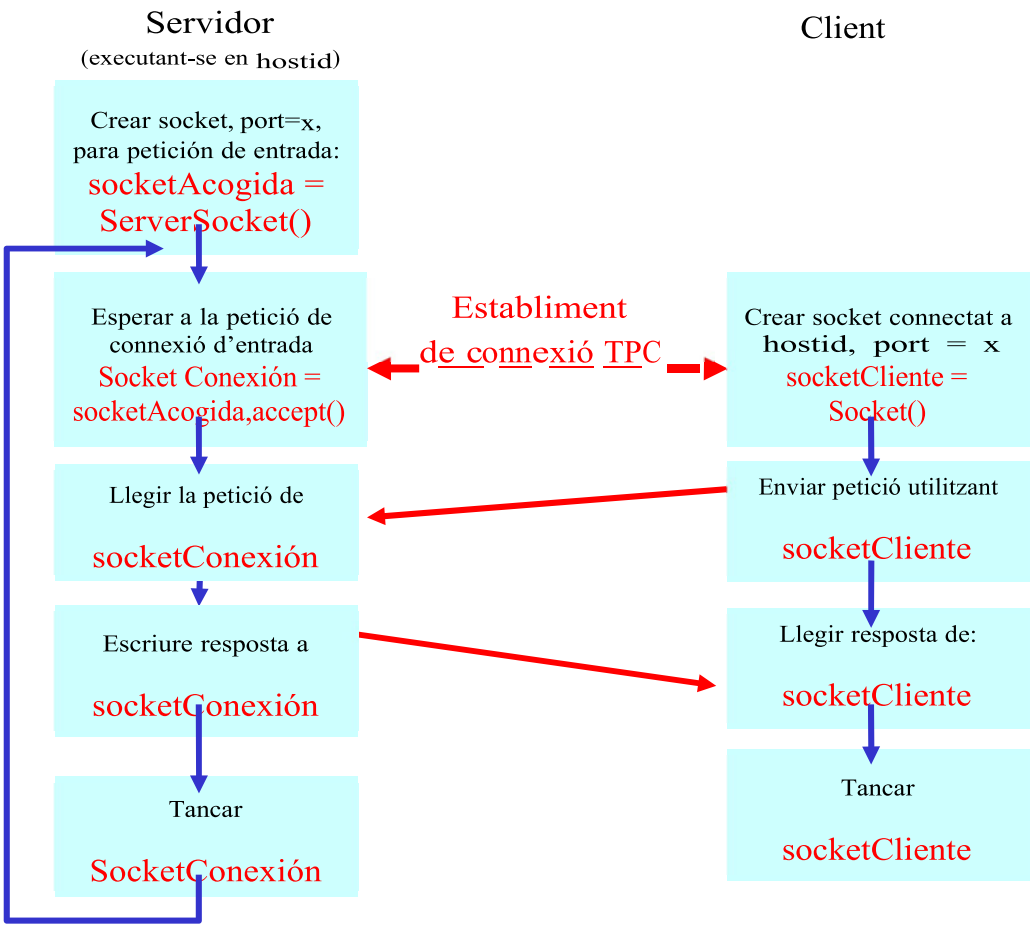


Figura 3. Aplicació client/servidor utilitzant serveis de transport orientats a la connexió.

4.2. Programació de Sockets amb UDP

Anteriorment vam veure que quan dos processos es comuniquen sobre TCP, des de la perspectiva dels processos és com si existís una canonada entre ells. Aquesta canonada es manté fins que un dels processos la tanca. Quan un dels processos vol enviar alguns bytes a l'altre procés, simplement insereix els bytes en la canonada. El procés emissor no ha d'incloure en els bytes l'adreça de destinació, ja que la canonada està connectada lògicament amb la destinació. És més, la canonada proporciona un canal fiable de flux de bytes: la seqüència de bytes rebuts pel procés receptor és igual que la seqüència de bytes que l'emissor va inserir en la canonada. UDP també permet que es comuniquin dues (o més) processos que s'executen en diferents host. No obstant això, UDP es diferencia de TCP en molts aspectes fonamentals. En primer lloc, UDP és un servei sense connexió: no existeix una fase inicial de sincronització durant la qual s'estableixi una canonada entre els processos. Atès que UDP no utilitza canonada, quan un procés desitja enviar una col·lecció de bytes a altre procés, el procés emissor ha d'afegir l'adreça del procés destinació a la col·lecció de bytes. I això ha de ser fet per a cadascuna de les col·leccions que enviï el procés emissor. Per tant, UDP és similar a un servei de taxi: cada vegada que un conjunt de persones pren un taxi, el grup ha de comunicar al conductor l'adreça de destinació. L'adreça de destinació és una tupla que consta de l'adreça IP del host de destinació i el nombre de port del procés destinació. Es denominarà paquet a la col·lecció de bytes d'informació juntament amb l'adreça IP de destinació i nombre de port.

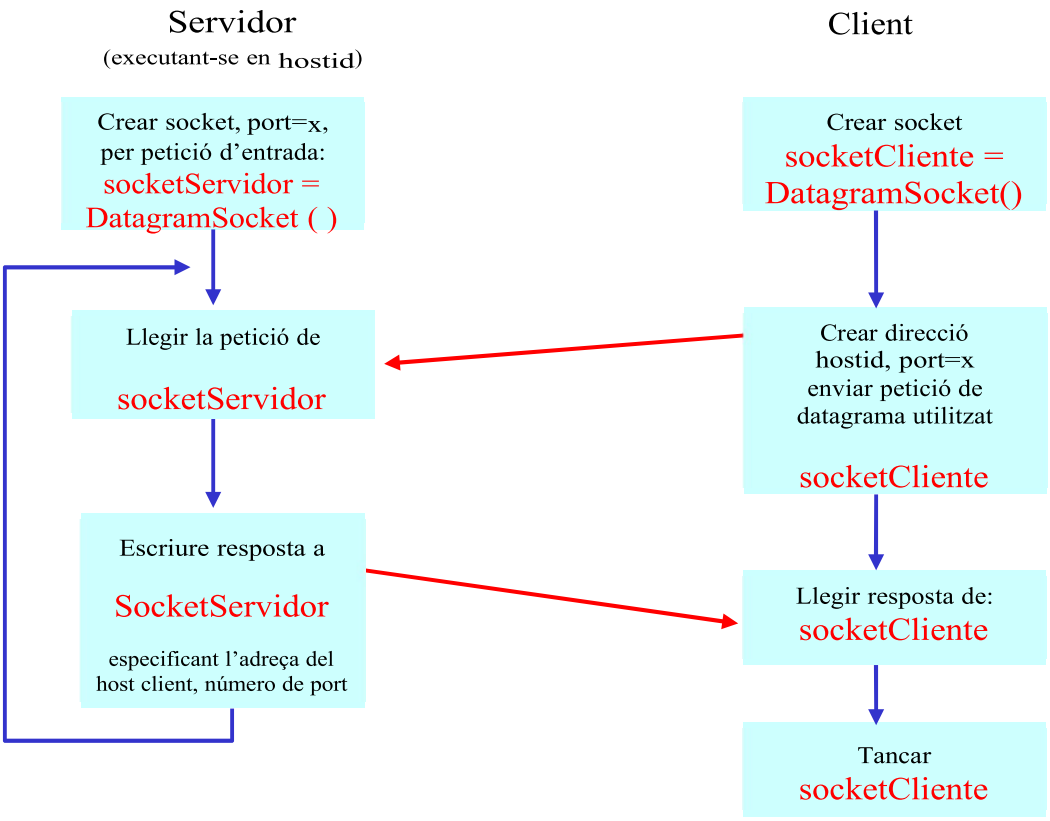


Figura 5. Aplicació client/servidor utilitzen serveis de transport sense connexió.

Després d'haver creat un paquet, el procés emissor impulsa al paquet sobre la xarxa a través d'un socket. Continuant amb l'analogia del taxi, a l'altre costat del socket hi ha un altre taxi esperant al paquet. És llavors quan el taxi duu al paquet cap a la seva adreça destinació. No obstant això, el taxi no garanteix que finalment vagi a dur el datagrama a la seva destinació última; el taxi podria tenir una avaria. Dita d'altra forma, UDP proporciona un servei de transport no fiable als processos que es comuniquen (no es garanteix que el datagrama arribi a la seva destinació final).

5. Transferència Fiable de Dades

La comunicació necessita al menys dos dispositius treballant junts, un per enviar i un altra per rebre. Aquest acord tan senzill necessita molta coordinació per que es produeixi un intercanvi de informació intel·ligible. Cal tenir en compte tres conceptes fonamentals:

- la gestió de l'enllaç
- el control d'errors
- el control de flux

En qualsevol sistema cap dispositiu deuria poder transmetre fins tenir evidencia de que el seu receptor és capaç de rebre i esta preparat per acceptar la transmissió. Que ocorre si el receptor no espera la transmissió, esta ocupat o no funciona? Si no hi ha forma de determinar l'estat del receptor, l'emissor pot desaprofitar el seu temps enviant dades a un receptor que no funciona.

Les funcions de la gestió de l'enllaç controlen l'establiment dels enllaços i el dret d'un dispositiu particular a transmetre en un moment donat.

El segon aspecte per aconseguir un canal fiable de dades es el control d'errors, que és basa en la petició de repetició automàtica (ARQ, Automatic Repeat Request), i s'implementa de forma senzilla: cada vegada que es detecta un error en un intercanvi, es retorna un reconeixement negatiu (NACK, Negative ACKnolegment) i es retransmeten els blocs específic. La retransmissió de dades és produirà en un d'aquest tres casos:

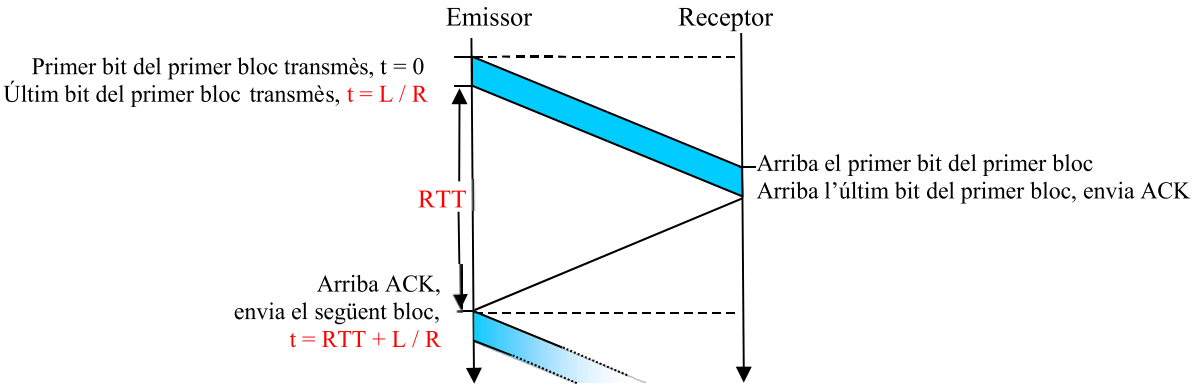
- Blocs de dades malmesos
- Blocs de dades perduts
- Reconeixements de blocs de dades perduts

Per últim el control de flux, és el concepte fonamental per tal de definir un canal fiable de dades. En la majoria dels protocols, el control de flux és un conjunt de procediments que li diuen a l'emissor quantes dades pot transmetre abans d'esperar un reconeixement del receptor. No s'ha de permetre que el flux de dades desbordi el receptor. Qualsevol dispositiu de recepció te una velocitat limitada per processar les dades que rep i una quantitat limitada de memòria en la qual emmagatzemar aquestes dades. El receptor ha de ser capaç d'informar a l'emissor abans d'assolir aquests límits i demanar a l'emissor que transmeti menys dades o que s'aturi temporalment. Les dades rebudes han de ser processades i comprovades abans de utilitzar-les. La velocitat d'aquest processament acostuma a ser més lenta que la velocitat de transmissió. Per aquesta raó, cada receptor té un bloc de memòria, anomenat buffer, reservat per emmagatzemar les dades rebudes fins que pugin processar-les. Si el buffer comença a omplir-se, el receptor ha de ser capaç de dir-li a l'emissor que aturi la transmissió fins que torni ha ser capaç de rebre dades.

S'han desenvolupat dos mètodes per controlar el control de flux de dades: Stop & Wait i Transmissió Continua.

5.1. Stop & Wait

Al mètode de control de flux Stop & Wait (Parada i Espera), l'emissor espera un reconeixement després de cada bloc que envia. Només s'envia el següent bloc quan s'ha rebut un reconeixement. Aquest procés d'enviar i rebre alternativament es repeteix fins que l'emissor envia un bloc de finalització de la transmissió. L'avantatge del Stop & Wait és la seva simplicitat: cada bloc es comprova i es reconeix abans de que s'enviï la següent. El desavantatge és la seva ineficiència, ja que el mètode es molt lent. Cada bloc ha de recórrer el camí fins el receptor i un reconeixement ha de viatjar del receptor a l'emissor abans de poder enviar aquest les dades següents. Amb d'altres paraules, cada bloc de dades està sol al canal. Cada bloc enviat i rebut utilitza tot el temps necessari per travessar l'enllaç. Si la distància entre l'emissor i el receptor es gran, el temps que es gasta esperant el reconeixement (ACK) entre cada bloc de dades pot ser una part important del temps total de transmissió.



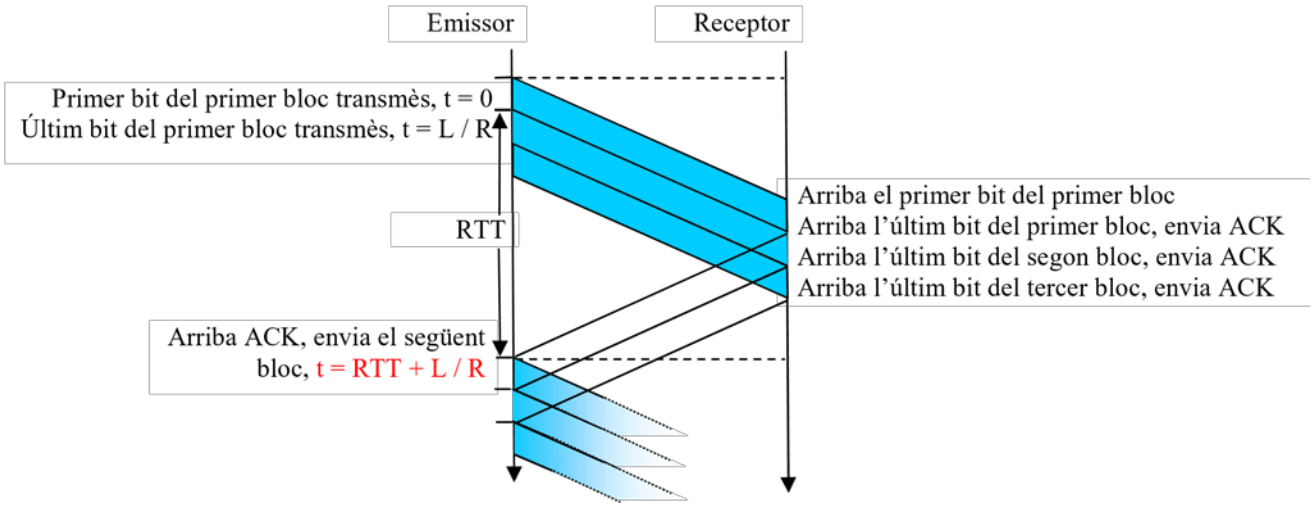
5.2. Transmissió Continua

Al mètode de control de flux anomenat Transmissió Continua (Finestra Lliscant), l'emissor pot transmetre un grup de blocs de dades abans de necessitar un reconeixement. Els blocs es poden enviar un darrere d'altre, el que significa que l'enllaç pot transportar diferents bloc a la vegada i la seva capacitat es pot aprofitar de forma més eficient.

El receptor notifica el reconeixement únicament per alguns dels blocs, utilitzant un únic ACK per confirmar la recepció de múltiples blocs de dades.

La finestra lliscant utilitza unes caixes imaginaries en l'emissor i el receptor. Aquesta finestra pot mantenir blocs en qualsevol dels extrems i proporciona un límit superior pel nombre de blocs que es podem transmetre abans de rebre un reconeixement. Els blocs podem ser reconeguts en qualsevol moment sense esperar fins que la finestra s'ompli i poden ser transmesos mentre la finestra no esta omplerta. Per saber quins blocs s'han transmès i quins blocs s'han rebut, la finestra lliscant introdueix un esquema de identificació basat en el tamany de la finestra. Els blocs s'enumeren en módul-n, el que signifiquen que s'enumeren de 0 a n-1. Per exemple, si n = 8, els blocs s'enumeren 0,1,2,3,4,5,6,7,0,1,2,3,4,5,6,7,0,1, ... La mida de la finestra es n-1 (en aquest cas, 7). En d'altres paraules, la finestra no pot cobrir el mòdul complet (8 blocs); cobreix un bloc menys.

Quan el receptor envia un ACK, inclou el número del bloc que espera rebre a continuació. És a dir, per reconèixer la recepció d'un grup de blocs, on l'últim bloc està numerat amb el número 4, el receptor envia un ACK amb el número 5. Quan l'emissor veu un ACK amb el número 5, sap que s'han rebut els cinc primers blocs numerats amb 0,1,2,3 i 4. La finestra pot mantenir n1 blocs en qualsevol extrem; per tant, podem enviar un màxim de n-1 blocs abans de necessitar un reconeixement.



6. Protocol TFTP

6.1. Propòsit

El protocol TFTP és un protocol simple de transferència de fitxer, i per això mateix se li va donar el nom de Protocol Trivial de Transferència de Fitxers. S'implementa sobre un servei de comunicacions no fiable i no orientat a connexió, és a dir, sobre UDP, i s'utilitza bàsicament per moure fitxers entre màquines de diferents xarxes que utilitzen UDP, mitjançant la lectura (operació get) o l'escriptura (operació put) per part d'un Client de/a un fitxer d'un Servidor.

6.2. Característiques fonamentals.

Tota transferència comença amb una petició de lectura o d'escriptura d'un fitxer per part del Client, que al mateix temps, també serveix per demanar una connexió. Si el Servidor accepta la sol·licitud, la connexió s'obre i s'envia el fitxer en blocs de longitud fixa de 512 bytes. Cada paquet de dades conté un d'aquest blocs, i té que haver un justificant de recepció mitjançant un paquet de reconeixement, de forma que en absència de fallida de les màquines, el fitxer acabarà sent transmès correctament. Per aconseguir la transferència fiable podem utilitzar el protocol de Stop & Wait o el protocol de Transmissió Continua. Un paquet de dades de menys de 512 bytes indica la finalització de la transferència. Cada paquet de dades porta un nombre de bloc, començant la transferència pel bloc 1. Cada paquet de reconeixement porta un nombre de bloc. En el cas d'utilitzar un protocol d'enviament continu el nombre de bloc indicarà l'últim paquet que és reconegut (els anteriors a aquest queden reconeguts implícitament) i en cas d'utilitzar un protocol Stop & Wait, el nombre de bloc identifica el paquet concret que s'està reconeixent.

Si un paquet es perd en la comunicació, produirà en el destinatari una interrupció per excés de temps (time-out), i llavors es pot demanar la retransmissió de l'últim paquet (que pot ser de dades o de reconeixement),causen al remitent del paquet perdut la retransmissió de dit paquet perdut. El remitent té que guardar únicament un paquet disponible en cas de ser necessària la retransmissió, degut a que ja s'han reconeguts tots els paquets anteriors, això garanteix la correcta transmissió fins aquest moment. Observeu que totes dues màquines involucrades en una

transferència es consideren remitents i destinataris. Un envia les dades i rep els reconeixements, l'altre envia els reconeixements i rep les dades.

La gran majoria dels errors causen la terminació de la connexió. Un error es senyalitza enviant un paquet d'error. Aquest paquet no és reconegut, i no es transmet (per exemple, un client o servidor de TFTP pot terminar la connexió enviant un missatge d'error), per tant l'altra extrem de la connexió no podrà recollir-lo. Per resoldré aquest problema cal utilitzar els temporitzadors (time_out), per tal d'esbrinar una terminació quan el paquet d'error s'ha perdut. Hi ha tres tipus de situacions que ocasionen error:

- quan no és possible acceptar una sol·licitud de transferència (fitxer inexistent, violació de permisos),
- quan es rep un paquet amb format incorrecte (degut a un retard o a duplicació a la xarxa),
- quan es perd l'accés a un recurs (s'omple el disc) durant la transmissió.

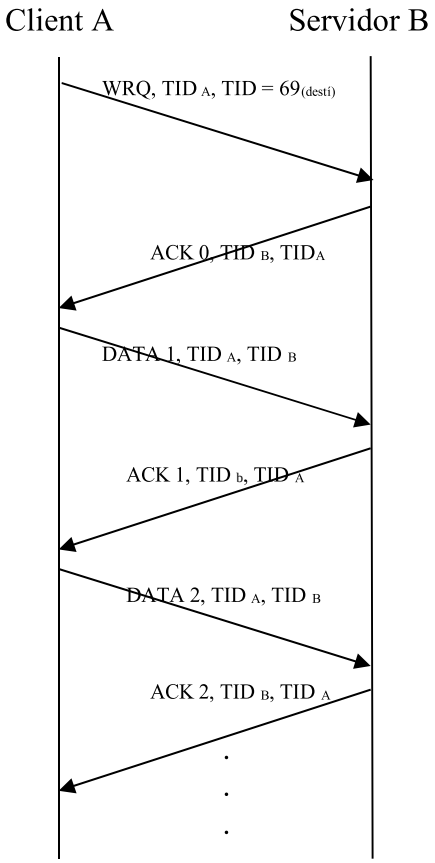
Aquest protocol s'ha dissenyat per que sigui molt restrictiu i poder simplificar la seva implementació. Per exemple, els blocs al ser de longitud fixa permeten l'assignació directa, i el paquet de reconeixement proporciona el control de flux i elimina la necessitat de demanar de nou els paquets de dades entrants.

6.3. Protocol inicial de connexió

La transferència s'inicia enviant una petició (s'utilitzarà WRQ per una petició d'escriptura en un sistema de fitxers remot, o RRQ per llegir del mateix), i esperar la recepció d'una resposta afirmativa, que pot ser o bé un paquet de reconeixement per procedir a l'escriptura del fitxer, o bé, el primer paquet de dades del fitxer llegit. De forma general un paquet de reconeixement, contendrà el nombre de bloc de les dades a confirmar. Cada paquet de dades té associat un nombre de bloc; els nombres de blocs son consecutius i comencen amb el número u. Donat que la resposta correcta a una petició d'escriptura, és un paquet de reconeixement, en aquest cas especial, el paquet de reconeixement contendrà el número zero (normalment, quan es reconeix un paquet de dades, el paquet de reconeixement, contendrà el nombre de bloc de dit paquet de dades). Si la resposta és un paquet d'error això indicarà que la petició d'escriptura o lectura, ha estat denegada.

Per que es pugui crear una connexió, cada interlocutor, generarà per ells mateixos un nombre de TID i aquest, s'utilitzaran mentre duri la connexió. Els TID es generaran de forma aleatòria, per tant la probabilitat de que surti el mateix número en dos generacions consecutives, és molt baixa. Cada paquet s'associa amb els TID de cada interlocutor, i serà el TID origen i el TID destí. Aquest TID seran utilitzats pel protocol de nivell inferior UDP com el port origen i el port destí. Una petició del servidor origen, fa generar el seu TID origen, i envia la petició inicial al port número 69 del Servidor destí. La resposta a la petició, sota condicions normals, utilitza el TID generat per ell com el TID origen i el TID entregat en el missatge previ pel Servidor origen, com el TID de destí.

El següent exemple mostra les fases utilitzades per establir la connexió per escriure un fitxer. Observeu que el WRQ, ACK i les DADES son els noms de la petició d'escriptura, reconeixement i tipus de dades respectivament.



1. El client A envia un WRQ al servidor B amb el seu origen TID del client A, destí 69.

2. El servidor B envia un ACK (amb número de bloc = 0) al client A amb origen TID del servidor B, destí TID del client A.

En aquest punt, podem dir que la connexió s'ha establert correctament i el primer paquet de dades ja pot ser enviat pel client A amb el número de seqüència 1. En el pas següent, i en tots els passos successius, el servidor té que comprovar que el TID origen coincideixi amb el valor tal com s'ha vist en els passos 1 i 2.

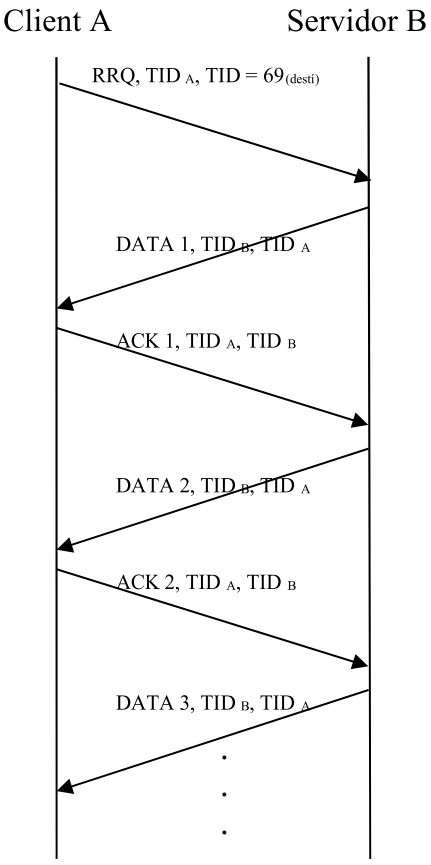
Si el TID origen no coincideix, llavors el paquet té que descartar-se al tractar-se d'un enviament erroni que ha vingut de qualsevol altra part de la xarxa. Per tant cal enviar un paquet d'error al client que ha generat aquest paquet incorrecte, sempre i quan no interfereixi la transferència. Aquest mecanisme només s'utilitzarà, si efectivament el TFTP rep un paquet amb un TID incorrecte.

Les característiques de la petició de lectura son similars a la petició d'escriptura, però hi ha petites diferencies que podem observa en el següent exemple on es mostra les fases utilitzades per establir la connexió per llegir d'un fitxer. Observeu que el RRQ, ACK i les DADES son els noms de la petició de lectura, reconeixement i tipus de dades respectivament.

El client A envia un RRQ al servidor B amb el seu port origen=TID del client A, amb port destí 69.

El servidor B envia el primer bloc de DADES (amb número de bloc=1) al client A amb port origen TID del servidor B, i port destí TID del client A.

En aquest punt, podem dir que la connexió s'ha establert correctament i el client A ha d'emetre un reconeixement pel primer bloc de dades (ACK 1). En el pas següent, i en tots els passos successius, el servidor té que comprovar que el TID origen coincideixi amb el valor tal com s'ha vist en els passos 1 i 2.



6.4. Tipus de paquets del TFTP

El TFTP suporta cinc tipus de paquets, tots ells anomenats anteriorment:

Codi	Operació
1	Petició de lectura (RRQ)
2	Petició d'escriptura (WRQ)
3	Dades (DATA)
4	Reconeixement (ACK)
5	Error (ERROR)

La capçalera del protocol TFTP d'un paquet conté el codi d'operació associat amb aquest paquet.

6.4.1. Paquets RRQ/WRQ

Format del paquet RRQ/WRQ:

2 bytes	string	1 byte	string	1 byte
Codi d'operació	Nom del fitxer	0	Mode	0

Aquest son els paquets de lectura i d'escriptura, amb codis d'operació 1 i 2 respectivament. El nom del fitxer consta d'una seqüència de bytes codificats en netASCII finalitzada amb un byte zero. El camp mode conté la cadena "netASCII" o "octet", la primera indica la transmissió de text pla normalment de 7 bits, mentre que la segona utilitza 8 bits, per tant permet transferir fitxers executables. Després del camp Mode, hi ha un camp d'un byte amb valor zero, per determinar el final del paquet.

6.4.2. Paquets DADES

Format del paquet DADES:

2 bytes	2 bytes	N bytes
Codi d'operació	Número de bloc	Dades

Els paquets de dades porten el codi d'operació 3, en el primer camp, en el segon porten el número de bloc seguit pel camp de dades. El camp de número de bloc, en els paquets de dades, comença per 1 i es va incrementant en una unitat per cada nou bloc de dades. Aquesta forma de treballar tan senzilla permet al programa discriminar entre els paquets nous i els duplicats. Els números de blocs no s'esgoten ja que rang dels números de blocs és de 2^{16}-1 números, i quan s'arriba a 65.535 es torna a començar per 1, per tant els números de blocs són cíclics (per un fitxer de 700 Mbytes, necessitaríem 1400 M blocs, això vol dir que recorrerien 21.363 vegades tots els números de blocs) . El camp de dades té una longitud variable entre 0 y 512 bytes. Si el bloc conté 512 bytes, això indica que no és l'últim paquet de dades, i que el segueixen més paquets; en canvi si la longitud del paquet oscil·la entre 0 i 511 bytes, això indica que finalitza la transferència.

Tots els paquets excepte ACK duplicats i aquells que causen una terminació son reconeguts, a menys que ocorri una interrupció per excés de temps (time_out). Quan s'envia un paquet de DADES s'està reconeixent implícitament el paquet ACK del paquet de DADES anterior. Els paquets WRQ i DADES és reconeixen amb un paquet ACK o ERR, mentre que els paquets RRQ i ACK és reconeixen amb un paquet de DADES o ERR.

6.4.3. Paquets ACK

Format del paquet ACK:

2 bytes	2 bytes
Codi d'operació	Número de bloc

El codi d'operació d'aquest paquet és 4, el número de bloc d'un paquet ACK és exactament el mateix del paquet de DADES que esta reconeixent. El bloc WRQ es reconeix a través d'un paquet de ACK amb el número de bloc posat a zero.

6.4.4. Paquets ERROR

Format del paquet ERROR:

2 bytes	2 bytes	string	1 byte
Codi d'operació	Codi d'error	Missatge d'error	0

El codi d'operació del paquet d'error és 5. Un paquet d'error és pot reconèixer amb qualsevol altre tipus de paquet. El codi d'error és un número sencer que representa la naturalesa de l'error produït, la següent taula mostra els diferents codis d'errors:

Valor	Descripció
0	No identificat, vegeu el missatge d'error (si existeix)
1	Fitxer no trobat
2	Violació d'accés
3	Disc ple o s'ha excedit la capacitat
4	Operació TFTP il·legal
5	Identificador de transferència desconegut
6	El fitxer ja existeix
7	Usuari desconegut

Els missatges d'error estan pensats per que siguin llegits per una persona i per tant estaran formatejat en netASCII. De la mateixa manera que d'altres cadenes (string), aquesta també terminarà amb un byte zero.

6.5. Terminació Normal

El final de la transmissió s'indica amb un paquet de DADES de 0 a 511 bytes de longitud (per exemple, longitud del Datagrama < 516). Aquest paquet es reconeix amb un paquet ACK de la mateixa manera que tots els altres paquets de DADES. El servidor que reconeix últim paquet de DADES amb un ACK, ja ha finalitzat la seva tasca. Per una altra part, cal esperar una mica. Això significa que el servidor que envia l'últim ACK esperarà durant algun temps abans de terminar per reenviar l'últim ACK per si aquest s'hagués perdut. El servidor que esta transmetent l'últim paquet de DADES, el continuarà transmeten fins que sigui reconegut, o al final es produeixi un error per excés de temps (time_out). Si la resposta és un ACK, la transmissió ha finalitzat correctament. Si en el servidor que esta enviant les DADES ocurreix un error per excés de temps, i no està preparat per enviar res més, això també indica que la transferència pot haver finalitzat totalment, però és possible que s'hagi produït algun problema posteriorment, o bé en el servidor que estava reconeixent les DADES o en la xarxa. També és possible en aquest cas que la transmissió s'hagués realitzat correctament. En qualsevol cas la connexió s'ha tancat.

6.6. Terminació Prematura

Si una petició no és contestada, o passa qualsevol error mentre dura la transmissió, s'enviarà un paquet d'ERROR (codi d'operació 5). Això es realitza només per cortesia, ja que és possible que mai es rebi o reconegui cap paquet. Per tant és necessari executar accions per excés de temps (time_out) per detectar els errors.

6.7. Extensions a TFTP

Mitjançant un mecanisme de negociacions d'opcions, podem fer-se extensions del protocol TFTP mantenint la compatibilitat cap en darrera. La idea és permetre la negociació entre el client i el servidor d'algunes opcions de la transferència de dades.

6.7.1. Negociacions d'opcions

El mecanisme de negociació d'opcions és de propòsit general, de forma que pot negociar-se amb ell diferents tipus d'opcions.

Les opcions s'afegeixen als paquets de sol·licitats RRQ i WRQ. A més es crea un nou tipus de paquet, el paquet OACK, per reconèixer una sol·licitud amb negociació d'opcions. Es crea així mateix un nou codi d'error, el 8, per indicar que la transferència ha de avortar-se degut a la negociació d'opcions.

Les opcions s'afegeixen als RRQ o WRQ de la següent manera:

2 bytes	string	1byte	string	1byte	string	1byte	string	1byte	...	string	1byte	string	1byte
Codi d'operació	Nom de Fitxer	0	Mode	0	Opció_1	0	Valor_1	0	...	Opció_N	0	Valor_N	0

El camp de codi d'operació pot contenir un 1 per peticions de lectura o un 2 per peticions d'escriptura, tal com en definit abans. De la mateixa manera, els camps nom del fitxer i mode, es defineixen com en el format original, tots dos han d'acabar amb un byte zero.

La primera opció Opció_1 es un string ASCII sense tenir en compte les majúscules o minúscules (per exemple, "blocksize"), aquest camp ha d'acabar amb un byte zero.

El valor associat amb la primera opció Valor_1 en ASCII sense tenir en compte les majúscules o minúscules, aquest camp també finalitza amb un byte zero.

Opció_N/Valor_N, es l'últim par de opció/valor. Cada camp estarà en ASCII sense tenir en compte les majúscules o minúscules i finalitzarà amb un byte zero.

Les opcions i els seus valor sempre terminaran amb un byte zero, per mantenir el format de les peticions originals. Si cal negociar múltiples opcions, aquestes s'afegiran una a continuació de l'altra. L'ordre de col·locació de les opcions, és indiferent. La longitud màxima d'un paquet de petició serà de 512 bytes.

El paquet OACK tindrà el següent format:

2 bytes	string	1 1 byte	string	1 byte	...	string	1 byte	string	1 byte
Codi d'operació	Opció_1	0	Valor_1	0	...	Opció_N	0	Valor_N	0

El camp codi d'operació contindrà un 6, pel reconeixement de l'opció. El camp Opció_1 és el reconeixement de la primera opció, que és una copia de la petició original, i el camp Valor_1 és el valor reconegut associat amb la primera opció. La forma en la qual aquest valor pot diferir del de la petició original, dependrà de l'opció concreta. El par Opció_N/Valor_N és el reconeixement per l'últim par opció valor.

Si el servidor suporta la negociació d'opcions, i reconeix una o més de les opcions presents en un paquet de sol·licitud rebut, respon amb un paquet de OACK. Cada opció que reconeix i el valor acceptat per dita opció, son inclosos en el paquet OACK. El servidor no pot incloure en el OACK cap opció que no li hagués estat presentada en la sol·licitud . Les opcions que no accepti el servidor son omeses del OACK, sense que es generi un paquet d'error. Si el valor especificat per una opció és invàlid, el servidor té l'opció d'omitir dita opció del OACK, de inclourala en el OACK amb un valor vàlid, o d'enviar un paquet ERR amb codi d'error 8 per avortar la transferència.

Una opció sol·licitada per un client i no reconeguda per un servidor ha de ser ignorada pel client, com si mai l'hagués presentat. Si un client rep al OACK una opció que no va sol·licitat, ha d'enviar un paquet ERR amb codi d'error 8 i finalitzar la transferència.

Quan el client afegeix opcions a una sol·licitud RRQ, pot donar-se tres possibles respostes del servidor:

- OACK: reconeixement del RRQ i d'una o més de les opcions. En aquest cas el client ha de reconèixer el OACK amb un ACK amb el número de bloc 0.

- DATA: reconeixement del RRQ, però de cap de les opcions.
- ERR: sol·licitud denegada.

Quan el client afegeix opcions a una sol·licitud WRQ, pot donar-se tres possibles respostes del servidor:

- OACK: reconeixement del WRQ i d'una o més de les opcions. Aquest OACK és reconegut directament amb el primer paquet DADES pel client.
- ACK: reconeixement del WRQ, però de cap de les opcions.
- ERR: sol·licitud denegada.

Aquest mecanisme permet que un client que presenti opcions pugui ser atès per un servidor que no tingui implementada la negociació d'opcions.

6.7.2. Opció de mida de bloc

El protocol TFTP estableix la mida de cada bloc de fitxer transmès en un paquet de DADES com de 512 bytes. Sin embargo, amb el mecanisme de negociació d'opcions descrit, el client pot proposar un altre valor.

En aquest cas, el client afegirà a la seva sol·licitud de WRQ o de RRQ una opció de nom "blocksize", i de valor el número de bytes proposat com mida de bloc, entre 8 i 4096. Si el servidor accepta aquesta opció, enviarà un OACK incloent-la, i amb un valor igual, major o menor proposat. El client llavors donarà aquest valor per bo, o avortarà la transferència enviant un paquet ERR amb el codi d'opció 8.

Les regles per acabar la transferència son les mateixes que en el cas de la tamany de 512 bytes: el paquet DADES amb menys bytes dels negociats com a tamany de bloc es considera l'últim.

6.7.3. Opció de temps d'espera

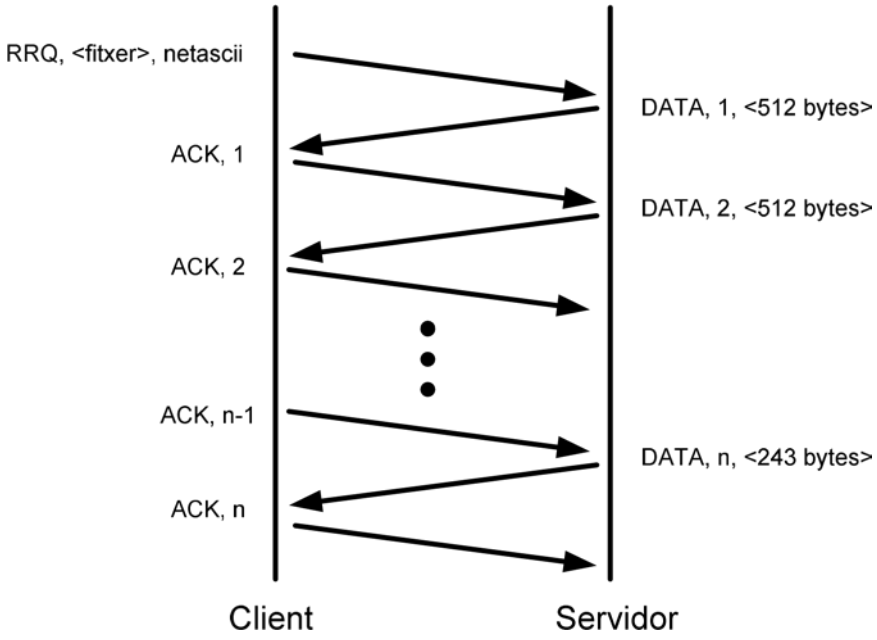
El protocol TFTP no estableix temps d'espera (time_out) que cada computador ha d'esperar per rebre el reconeixement d'un paquet. Sin embargo, mitjançant el mecanisme de negociació d'opcions descrit anteriorment, el client pot proposar un valor a emplear durant la subsegüent transferència.

En aquest cas, el client afegirà a la seva sol·licitud de WRQ o de RRQ una opció de nom "timeout", i de valor el número de milisegons proposat com a temps d'espera, entre 100 i 255000.

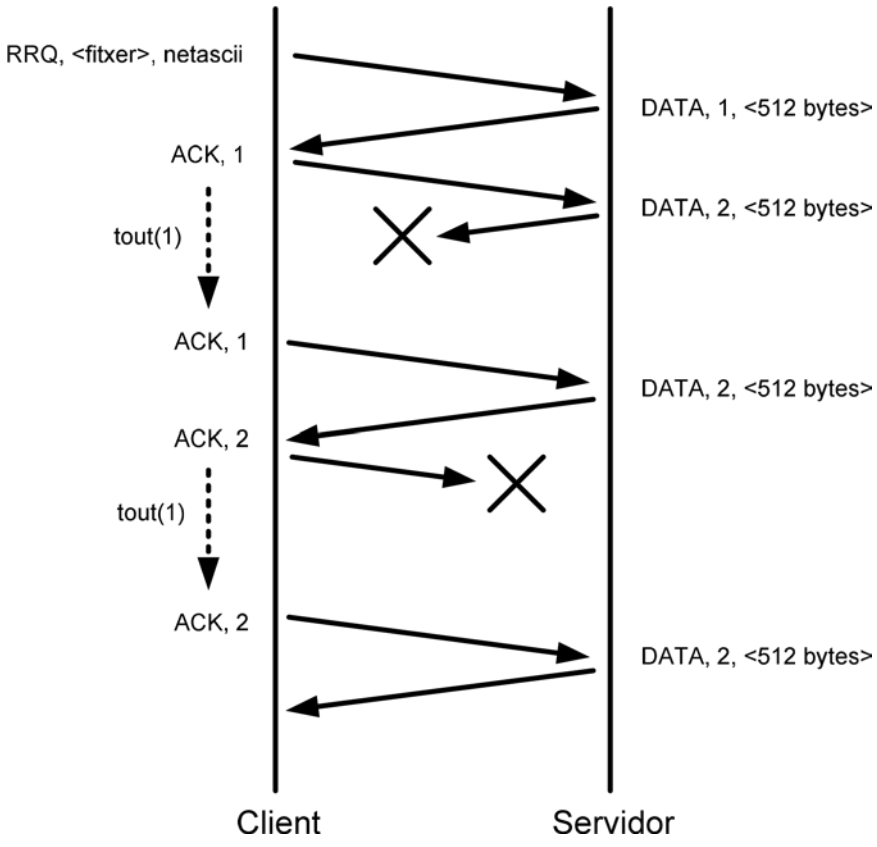
Si el servidor accepta dita opció, enviarà un OACK incloent-la, i amb un valor igual al proposat.

7. Exemples del funcionament del protocol TFTP

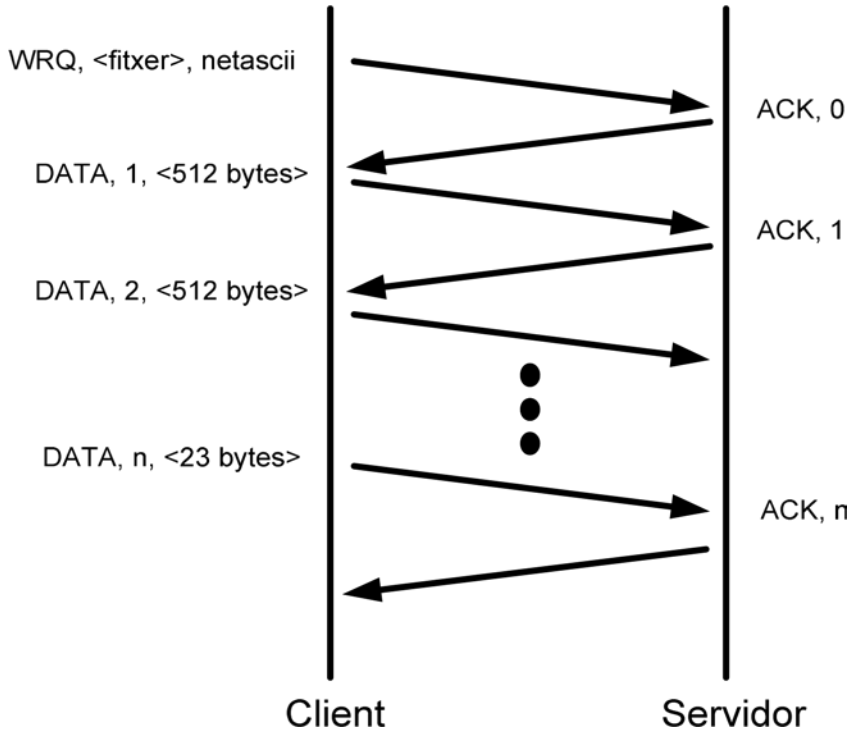
Lectura remota d'un fitxer



Lectura remota d'un fitxer (errors):



Escriptura remota d'un fitxer:



Proveu el protocol TFTP al vostre computador:

C:\>tftp

Transfiere los archivos a y desde un equipo remoto con el servicio TFTP en funcionamiento.

TFTP [-i] host [GET | PUT] origen [destino]

-i	Especifica el modo de trasferencia binario (llamado también octeto). En modo binario el archivo se transfiere literalmente byte a byte. Use este modo cuando transfiera archivos binarios.
host	Especifica el host remoto o local.
GET	Transfiere el archivo destino en el host remoto al archivo origen en el host local.
PUT	Transfiere el archivo origen en el host local al archivo destino en el host remoto.
origen	Especifica el archivo a transferir.
destino	Especifica dónde transferir el archivo.

8. Bibliografia

RFC 1350 K. Sollins. The TFTP Protocol (Revision 2). Juliol de 1992.
http://www-rfc-editor.org/rfc/rfc1350.txt

RFC 1782 G. Malkin, A. Harkin. TFTP Options Extensions. Març 1995.
http://www-rfc-editor.org/rfc/rfc1782.txt

RFC 1783 G. Malkin, A. Harkin. TFTP Bloksize Option. Març 1995.
http://www-rfc-editor.org/rfc/rfc1783.txt

RFC 1784 G. Malkin, A. Harkin. TFTP Timeout Interval and Transfer Size Options. Març 1995.
http://www-rfc-editor.org/rfc/rfc1784.txt