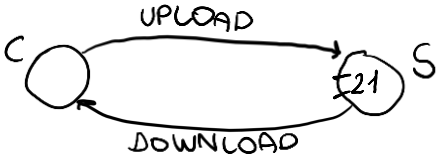
---------- seti 8/11 ----------

Livello applicativo -> **PROTOCOLLO FTP** (File Transfer Protocol, RFC 959)
Uno dei primi protocolli, attualmente in disuso
Protocollo di tipo client-server -> si effettua una connessione per trasferire file, due direzioni
Trasferimento di file da client a server -> upload
Trasferimento di file da server a client -> download



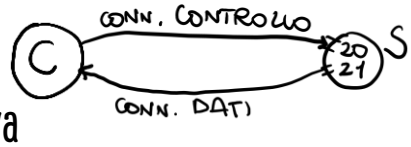
È coetaneo del protocollo di posta elettronica, condividono proprietà

Client si connette al server con una connessione di tipo stream (porta 21/TCP), e invia una serie di comandi. Necessita l'autenticazione, vanno garantite caratteristiche di sicurezza dei vari sistemi, bisogna che il server riconosca il client

Comandi:

- Comando "list", come "ls" sul server -> vedo file presenti sul server con permessi di accesso
- Comando "cwd", come "cd" -> è come aprire shell sul server

Su connessione possono viaggiare solo dati codificati ASCII 7 bit, se voglio altro file apro un'altra connessione solo per il trasferimento dati (porta 20/TCP) -> **connessione di controllo**, solo per trasferimento di file e non permanente



2 modi di funzionamento per connessione dati: **attiva e passiva**
Di default è modalità attiva, modalità passiva inserita in un secondo tempo, deve essere esplicitamente richiesta dal client

- **Attiva**: per connessione dati client e server si scambiano i ruoli, client deve ottenere socket, fare bind, listen e accept. Server deve aprire socket e fare connect. Problema: client di solito non ha privilegi di amministratore, quindi non può fare bind su porta nota ma solo su effimera; deve comunicarlo al server, che da porta well-known (20) fa partire 3-way-handshake a porta effimera
- **Passiva**: oggi sempre presenti firewall per sicurezza, ma difficile gestirli in modalità attiva, più facile in passiva, dove client fa connect e server accept.

canale dati serve sia in upload che in download.
in connessione comando non arriva l'ok fino a quando il trasferimento dati non è finito
in modalità attiva, cosa succede se client dà l'indirizzo di un'altra macchina per la connessione file? il server disturba un'altra macchina -> per questo firewall configurato passivo

ci sono varianti più nuove del protocollo: ftps e sftp (s = secure)
adottano sistemi di sicurezza, cifratura di password, certificati digitali al posto di autenticazione con username, ecc.

nella versione originale serve lo stesso account sia su client che su server per l'autenticazione, ad esempio posso usare server come estensione di memoria del client -> primo tentativo di cloud
è definito però anche un ftp anonimo, dove non mi autentico sul server, e sul server l'utente è definito come "anonymous", non ha password associata, ma accesso comunque abilitato, con modalità anonima, il server non può distinguere il client -> anonymous ftp

può servire in download per distribuire a chiunque un file
problematiche di sicurezza: file distribuito deve essere non pericoloso da distribuire a tutti
per upload: permetto a chiunque di mandarmi file -> uno degli scopi di questo protocollo era il trasferimento di grandi quantità di dati
problemi di sicurezza: se qualcuno fa upload di un virus sul mio server e qualcuno lo scarica -> soluzione: se permetto upload non posso consentire download e viceversa, così nessuno può scaricare virus. in anonimo è consentita solo una modalità.

protocollo http (prima versione rfc 1945)

il server usa porta 80/tcp, il client usa una porta effimera
una volta instaurata la connessione, il protocollo è principalmente usato per download di file
principale differenza: si usa la stessa connessione per trasferimento dati
connessione a 8 bit, si possono trasferire interi byte, versione semplificata con funzionalità analoghe a ftp

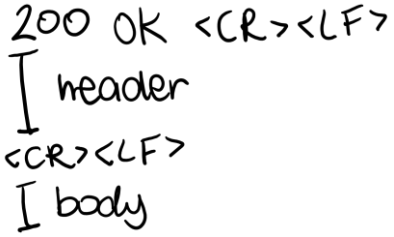
non prevista autenticazione, tipo anonimo in download. in RFC si parla di una prevista attivazione della modalità upload, ma non è mai avvenuto, per evitare problemi presenti anche in FTP

diverse versioni di protocollo, prima HTTP/1.0, ormai obsoleta e non in uso, ma più semplice. dopo un anno circa 1.1, attualmente utilizzabile, che offre più funzionalità; poi 2.0 e 3.0.

dialogo con comandi ascii a 7 bit, divisione in righe?
metodo, righe header, poi body (che può essere file binario)
righe terminate da <CR><LF>
dopo header riga vuota
vari tipi di metodo: get, head, post ...

metodo get: nome file + versione protocollo, client e server possono usare versioni diverse, client chiede specifica versione, se server può risponde con 'ok', se no ritorna errore con motivazione

possibile risposta del server:



HTTP/1.0

server in attesa di connessione con listen, client si connette -> connessione non permanente, dopo ogni messaggio viene chiusa (molto scomodo, cambiato già in 1.1 con una connessione permanente)
metodo principale usato get (= voglio fare download di un file)
file principalmente di tipo HTML -> HTML è un linguaggio di mark up e contiene link ad altri file, dentro ad un file può esserci riferimento ad altri file, posso scoprire contenuti di server (?) senza sapere com'è configurato
in tutti i server HTTP c'è file "/", detto homepage (come radice unix), file HTML contenente link ad altri file, così l'utente scopre file interessanti di cui richiederer download; non c'è comando list

dopo richiesta con get (nomefile versione) vado accapo e metto l'header per dissuadere il client, in modo che il server possa dare risposte significative al client; poi header termina

con riga vuota. server deve accettare connessione, stabilizzarsi su un socket diverso (sul socket originario arriveranno richieste da altri client), cerca su filesystem il file e ne manda il contenuto (se ci sono i permessi)
l'header di risposta può contenere metadati del file inviato e una descrizione del suo contenuto, data usando classificazione di tipo mime (come attachment di posta elettronica) in 1.0 dopo l'invio si chiude la connessione

un altro metodo è il **metodo head**: analogo al get, chiedo file, se c'è, il server prepara header come in get ma mi manda solo quello -> ottengo i metadati ma non il contenuto

metodo post: via di mezzo per upload di informazioni. faccio get di pagina html, guardo il contenuto, trovo form che può essere compilato (esempio. nome: _____, cognome: _____, password: _____). compilo il form e faccio submit, devo mandare queste informazioni al server, mando non file qualsiasi ma form compilato come body; server riconosce post, guarda body e ricava informazioni. nel form possono esserci username e password, ma non usati da http, lui gestisce solo il trasferimento, diverso da autenticazione di ftp

http è un protocollo privo di stato, se ripeto lo stesso comando succede sempre la stessa cosa -> server dimentica tutto dopo aver risposto, ma le informazioni trasferite possono essere memorizzate in database indipendentemente da http