Reti di calcolatori, parte 2

M3 U1 L1 Nomi e Indirizzi

IPv4 e v6 sono sequenze di numeri usate per individuare le risorse di rete. Gli utenti però ricordano più facilmente i nomi che le sequenze di numeri.
Il DNS (Domain Name Service) fornisce una corrispondenza (mapping) da nomi a risorse di molti tipi.

Caratteristiche del DNS

Distribuzione globale: I dati vengono gestiti localmente ma sono recuperabili globalmente, le ricerche nel DNS possono essere eseguite da qualsiasi dispositivo e I dati remoti del DNS sono memorizzabili localmente in una cache per migliorare le prestazioni.

Coerenza: Il database DNS è sempre internamente coerente (Ogni versione di una porzione del database (zona) ha un numero seriale, che viene incrementato ad ogni modifica del DB).
Le modifiche alla copia master del database relativa a una zona vengono duplicate con la frequenza impostata dall'amministratore di zona.
Le modifiche alla copia master del database relativa a una zona vengono duplicate con la frequenza impostata dall'amministratore di zona.
I dati memorizzati nelle cache dei client scadono dopo un intervallo di tempo impostato dall'amministratore di zona.

Scalabilità: Nessun limite alla dimensione del database DNS (oltre 20kk di nomi per servizio). Nessun limite di query (24k query gestite facilmente, e le query possono venir distribuite)

Affidabilità: I dati vengono duplicati (dal master di zona vengono copiati in più slave)
I client possono interrogare le loro cache locali, il server master di zona o una qualsiasi copia dei dati nei server slave.

Dinamicità: Il database può essere aggiornato dinamicamente. Aggiunta, eliminazione, modifica di qualsiasi record

Nomi piatti e gerarchici

Uno schema di denominazione piatto genera delle etichette diverse tra loro, senza alcuna relazione tra loro
Uno schema gerarchico invece comprende delle parti (il nome è composto da porzioni), così che i nomi possano avere in comune degli elementi che identificano la loro posizione nella gerarchia.

Il DNS gerarchico fu creato da Paul Mockapetris nell'83 (RFC 1034 e 1035), e ha subito una lunga evoluzione con successive RFC.
Il DNS è un meccanismo di ricerca per tradurre gli oggetti in altri oggetti (nomi in indirizzi e viceversa),
Un database DNS è composto da tre elementi:

- un ***namespace*** (definizione gerarchica della struttura dello spazio dei nomi)
- i ***server*** che rendono disponibile quel namespace (rendono disponibile il DNS)
- i ***resolver*** (client) che interrogano i server sul namespace (software sulle macchine che pongono interrogazioni ai server opportuni)

DNS server

Un DNS server è una macchina che tiene una tabella dei nomi e i corrispondenti indirizzi IP (è in grado di risolvere ALCUNI dei nomi).
Il resolver sul nostro host sa a quale macchina rivolgersi per tradurre un nome.
Il DNS server può già conoscere la traduzione, perché l'ha effettuata in precedenza, diversamente valuta l'ultima parte del nome (.it, .de, .com, .edu). Questo farà riferimento ad un root server, che saprà a quale

server far riferimento per la seguente parte gerarchica a partire da destra (es. unimi.it, il root server “it” saprà quale server potrà risolvere “unimi”, e così via fino ad arrivare alla parte più a sinistra dell’URL).

M3 U1 L2 Caratteristiche del DNS

DNS è un db distribuito GLOBALMENTE su più DNS server.
Il DNS dovrà essere coerente, scalabile (la crescita è continua, parallela alla crescita degli host), affidabile e dinamico

Distribuzione globale
I dati vengono gestiti localmente ma recuperati globalmente → nessun singolo computer ha tutti i dati DNS.
Ogni dispositivo può eseguire delle ricerche DNS, e i dati sono memorizzabili in cache.

Coerenza
La coerenza viene garantita dal “versioning”, facendo sì che ogni versione di una porzione del db abbia un suo numero siarel, che viene cambiato ad ogni cambiamento del DB.
Per mantenere coerenti i dati in cache, si terrà comunque la versione dei dati memorizzati, per verificarne la validità, e inoltre viene imposta una scadenza di essi dopo un intervallo di tempo

Scalabilità
Non c’è alcun limite alla dimensione del database DNS, e può contenere più di 20kk di nomi.
Inoltre, le query sono gestite in maniera molto efficiente e sono gestite tra master, slave e cache.

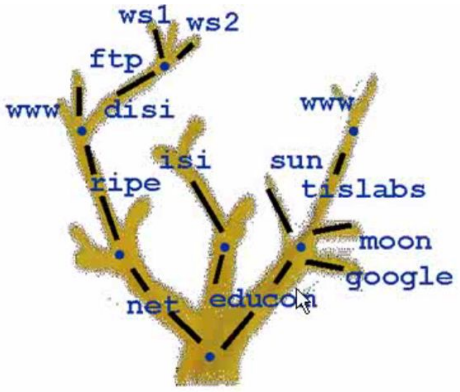
Affidabilità
I dati vengono duplicati (i dati del master di zona vengono copiati in più slave), così che se il DNS locale dovesse cadere, in un’altra parte siano reperibili.Quindi i client potranno interrogare le loro cache locali, il server master di zona, o un qualsiasi server slave che contiene la copia dei dati
Dinamicità
Il database può essere aggiornato dinamicamente (le varie zone possono evolvere a velocità diversa). Gli slave non possono essere aggiornati dinamicamente, in quanto sono solo copie del master, e ciò rende i master dei single point of failure (se vanno giù, non è garantito che gli slave abbiano dati aggiornati).

M3 U1 L3 Complementi sui nomi

Nomi DNS
Il namespace DNS deve essere gerarchico, che vanno in ordine di generalità a partire da sinistra:

- Il suffisso più a destra identifica il root server (“.it”)
- Quindi si ha un primo suffisso che rappresenta una “location” nella nazione o nella società (“unimi.it”)
- Poi si prosegue con le unità all’interno della location (“dti.unimi.it”)
- E infine si ha l’oggetto dell’unità (nome del computer nella società)

Il **FQDN (Fully Qualified Domain Name)** identifica il nome di dominio completo che termina con un punto, ad esempio:
→ WWW.RIPE.NET.
Il DNS andrà a prendere un FQDN comunicato da un resolver, e tramite un opportuno scambio di messaggi ritornerà l’indirizzo.
I nomi di dominio possono essere mappati su un albero:



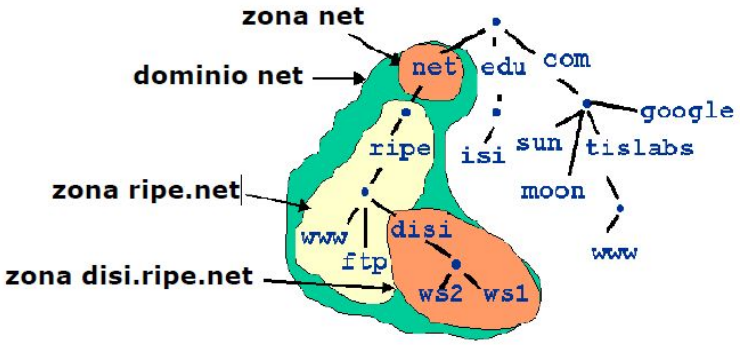
L'attivazione di nuovi rami di questo albero può essere delegata alle autorità interne delle organizzazioni che detengono un nome di secondo livello, senza far riferimento ad una autorità centrale.

Domini
Si tratta di un "namespace local".

- Tutto ciò che è sotto ".com" è nel dominio "com"
- Tutto ciò che è sotto ripe.net è nel dominio "ripe.net" e nel dominio "net".

In generale quindi funziona un meccanismo di ***delega***.
L'amministrazione di un dominio di alto livello può delegare qualcun altro per la gestione del sottodominio (di livello inferiore).

I database sono associati alle zone, non ai domini.



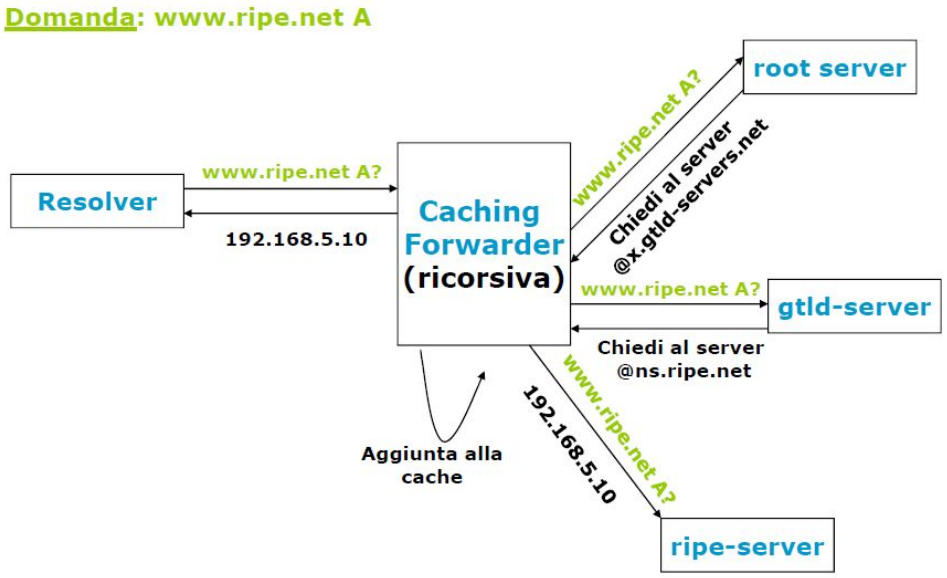
Per ws1 e ws2 sarà il server di zona "disi" che conterrà le copie più aggiornate di mapping (copie di autorità).

Name server
Rispondono alle interrogazioni DNS e ve ne sono diversi tipi:

- ***server autorevoli***: di tipo master o slave (contengono fisicamente una parte del DB di traduzione)
- ***server ricorsivi***: servono per sapere a quali altri server rivolgersi (caching o forwarder caching)
- ***server con funzionalità miste***: i server di zona possono fare da forwarder per certe query e da authoritatives per altre.

Periodicamente gli slave ricevono copie delle informazioni presenti nei master.
Un esempio di server ricorsivo è quello che abbiamo nella nostra rete locale, il quale si occuperà di risolvere le query DNS andando a chiedere ad altri name server. Il server ricorsivo salverà in cache la risposta, segnandola però come "non autorevole".

M3 U1 L4 Risoluzione dei nomi



TLD: Top level domain (.net, .it…)

Più è lunga la gerarchia nella query, più tappe si faranno.
Di solito i resolver sono implementati in una libreria di sistema, richiamabile come segue:

- `gethostbyname(char *name);`
- `gethostbyaddr(char *addr, int len, type);`

Il DNS ha un formato di record più che uno schema (come mongo)
Il tipo fondamentale di record nel DB è il **RR (Resource Record)**, costituiti da:

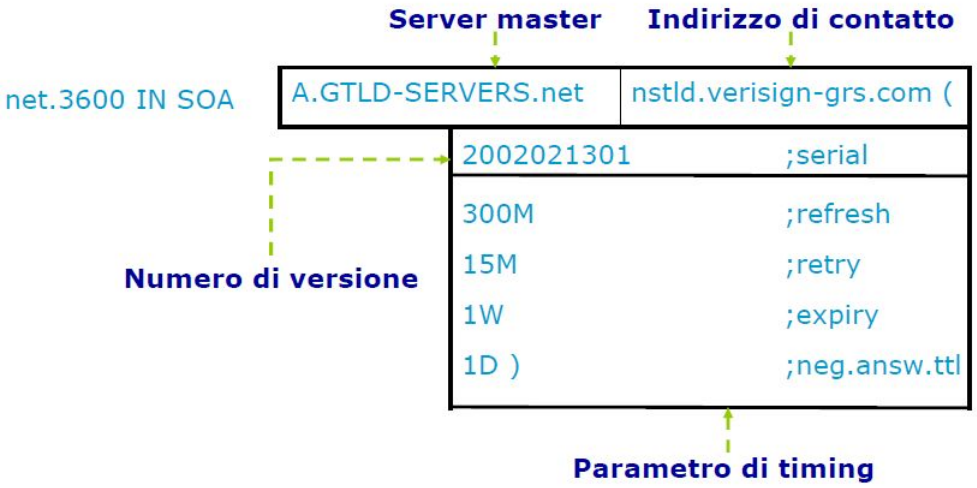
- etichetta
- TTL: parametro di timing
- classe: classe della risorsa (per identificare se la traduzione è finale o è un forwarding)
- tipo di risorsa
- RDATA: indirizzo della risorsa

Esempio di un RR di un file di zona

```
ripe.net.             7200 IN    SOA   ns.ripe.net.    olaf.ripe.net.
      (
            2001061501 ; Serial
            43200      ; Refresh 12 hours
            14400      ; Retry 4 hours
            345600     ; Expire 4 days
            7200       ; Negative cache 2 hours
      )
ripe.net.             7200  IN    NS    ns.ripe.net.
ripe.net.             7200  IN    NS    ns.eu.net.
pinkie.ripe.net.      3600  IN    A     193.0.1.162
```

RR NS
I record di *tipo* **NS** indicano il punto dove si possono trovare informazioni su una data zona (differiscono da A, che invece rappresentano la macchina finale vera e propria)

RR SOA
Forniscono informazioni sull'origine delle autorità (ossia chi ha delegato l'autorità), chiamata anche APEX



TTL vs Timer SOA
Il TTL è un timer usato nelle cache: indica per quanto tempo i dati possono essere riusati (i dati "stabili" possono avere TTL alti).
I timer SOA vengono *utilizzati per mantenere coerenza tra server primari e secondari.*

I cambiamenti al DNS non si propagano istantaneamente:
- Gli slave continuano a rispondere alle interrogazioni (secondo i tempi specificati nel SOA)
- I forwarder continuano a rispondere alle interrogazioni (secondo i tempi specificati nei TTL)
- In alcuni casi ci sono dei resolver locali che salvano in cache le corrispondenze per tempo semi-indefinito (quindi vai a capire quanto tempo ci si potrebbe mettere per la traduzione)

DNS e prestazioni
I DNS necessitano di server autoritativi multipli, e fanno largo uso di cache per ridurre il carico ai server autoritativi e diminuire i tempi di risposta.
I timer SOA e TTL devono essere ottimizzati in base alle necessità di zona.

La distribuzione dei nomi però non dovrebbe MAI corrispondere ad una struttura vera delle macchine nell'ambiente di rete privata.

M3 U2 L1 FTP
FTP (File Transfer Protocol) si basa su TCP e consente sia il trasferimento file sia l'accesso interattivo. Richiede l'autenticazione tramite nome utente e password.
Si basa su un'architettura C/S, e il server dovrà disporre in un servizio che rimane in ascolto delle richieste FTP su una porta nota FTP. È anche possibile definire una modalità di accesso anonima, che prevede l'utilizzo di una stringa convenzionale che permetta solamente di leggere file.

Il server FTP è costituito da due processi:
- Il processo di controllo che comunica con il processo di controllo remoto (e riceve i comandi)
- Il processo di trasferimento dati che trasferisce veramente il file richiesto.

I due funzionano parallelamente (sarà possibile dare comandi mentre si effettua il trasferimento)

Il processo di controllo client si connette al processo di controllo server (tramite il comando `ftp media.dti.unimi.it`).
Inoltre, il client attiva un processo di trasferimento dati su una porta locale in grado di ricevere connessioni. Questo numero di porta viene comunicato al server, così che si possa instaurare la connessione, e che si possa evitare il problema del firewall.

GET e PUT permettono di attivare la procedura di trasferimento file. Alla ricezione, server inizierà una comunicazione con la porta comunicata dal client (usando la porta su server 29) in modo da iniziare il trasferimento.
I comandi sono un sottoinsieme del protocollo Telnet NVT e vengono inviati in formato simil ASCII (ASCII NVT)

Si useranno poi dei ***codici di controllo*** per rispondere ai comandi inviati e indicare il grado di successo del comando inviato:

- 1xx = OK, lo farò
- 2xx = OK, fatto
- 3xx = OK, finora (ad esempio “ok il nome utente, ma ora devi darmi la password”)
- 4xx = NO, temporaneamente
- 5xx = azione richiesta

M3 U2 L2 Da FTP a Telnet

Una tipica sessione FTP prevede che:

- Il server sia in ascolto sulla porta 21
- Il client si connetta alla porta 21
- Il client mandi un comando dalla sua porta (non well-known) alla porta 21, indicando poi la porta sulla quale accetterà la risposta del server
- Il server risponde al comando sulla porta di connessione del client, e poi fornisce al client sull’altra porta il risultato del comando

Questo funziona linearmente, ma le varie configurazioni di firewall possono cambiare le carte in tavola.
Nel caso non sia possibile la connessione da fuori (server) verso il dentro (client) non sia possibile si attiva la ***modalità passiva***.
In questa modalità, sarà il server a comunicare la porta al client, e sarà il client ad attivare la connessione TCP verso il server per ricevere i risultati del comando.

Telnet
Protocollo fratello di FTP, che consente di inviare comandi da una macchina all’altra. Trasmette i suoi comandi in ASCII NVT. Richiede anche lui l’autenticazione tramite user e password (che circolano in chiaro).
Richiede che sul sistema remoto ci sia in esecuzione un server telnet (*telnetd*) (questo sarà niente di più che un processo che passerà i comandi digitati da client ad una shell locale di sistema operativo).
Quindi telnet offre due servizi di base:

- Un terminale di rete virtuale
- Un meccanismo per gestire le opzioni di comunicazione (inviando comandi che non saranno inoltrati alla shell locale ma che andranno a configurare le opzioni relative a Telnet).

Chiaramente, *Telnet non è molto sicuro* (infatti di solito sta dietro firewall che impediscono la connessione da parte di host remoti).
La porta nota di Telnet è la 23, però ogni volta che si inizia una connessione via telnet si utilizza una connessione su una porta diversa.
Telnet, però, è in grado di gestire l’eterogeneità dei sistemi, permettendo di evitare di collegare fisicamente dei terminali alla macchina (così da usare la rete locale).
Il paradigma Telnet prevede comunque che venga generato molto traffico (***tinygram***), in quanto ogni comando dovrà essere inoltrato subito al destinatario

M3 U2 L3 SMTP: introduzione

Simple Mail Transfer Protocol è il principale protocollo per lo scambio di posta elettronica su Internet.

Si basa su TCP, e a differenza di FTP/TELNET non è un protocollo interattivo (i messaggi vengono accodati e inviati tramite lo spooler dell'agente SMTP).

Gli utenti utilizzano degli editor (client di posta elettronica) che generano dei file e li salvano nella directory di spooling, a cui accede un daemon chiamato "Mail Transfer Agent" (MTA).
Su unix, questo si chiama "sendmail" (che è critico per la sicurezza, perché è difficile da chiudere).
SMTP specifica come il MTA deve trasmettere le mail tramite internet, e utilizza i comandi NVT.

M3 U2 L4 Sistemi P2P

Si tratta di sistemi composti da attori dello stesso tipo (tutti uguali) che collaborano tra loro.
Questi sfruttano risorse che non c'entrano con internet, come Memoria e contenuti, cicli di CPU e il lavoro degli utenti stessi.
Potenzialmente si tratta di sistemi a basso costo, ma bisogna anche far fronte al fatto che il sistema deve funzionare anche a limitata availability.
P2P si divide in due approcci:

- ***Non strutturati***: in cui gli attori sono programmi tutti uguali e non c'è relazione tra il programma e il suo compito (comunicazione che avviene tramite flooding)
- ***DHT Strutturati***: in cui il compito, il ruolo, di un programma è determinato dalla sua posizione in una rete P2P rispetto agli altri (quindi in base ai "vicini" che ha)

Reti overlay

La rete P2P è un overlay della rete IP (situata al di sopra di essa), i PC che si conosceranno in una rete P2P si conosceranno attraverso dei nomi, dei nick ecc.
Ogni sistema P2P ha un suo sistema di indirizzamento gestito a livello di applicazione, e con una certa capacità (limitata o globale) di risolvere gli indirizzi di overlay in IP.

Ogni nodo conoscerà gli indirizzi overlay e IP dei suoi vicini. Ciò rende possibile la trasmissione dei messaggi, al peggio usando il flooding (che prima o poi porta a destinazione il messaggio).
L'overlay però va anche mantenuto:

- periodicamente viene eseguito un ping per assicurarsi che il vicino sia ancora raggiungbile
- Verifica che i vicini siano in vita prima dell'invio di messaggi
- Se il vicino cade, e c'è un numero minimo di vicini da avere, bisognerà reclutarne un altro.

Gli ***overlay non strutturati*** in cui ogni nodo sceglie i suoi vicini casualmente (ci si attribuisce un nick di overlay, e si cercano dei vicini (anche nella propria sottorete), che vanno tutti bene).
Negli ***overlay strutturati*** invece occorre trovare i vicini giusti in cui inserirsi, in base al ruolo che si vuole assumere. In questo caso si dovrà ricevere un overlay address in base alla posizione che si vuole ricoprire (cercando di mantenere contiguità con gli overlay address degli altri host coerenti), e anche l'operazione di adesione all'overlay è più complessa.
La posizione dipende da vari fattori, ad esempio i tipi di file che si vogliono condividere (ad esempio, se si hanno prevalentemente file immagine, si verrà collocati tra vicini che anche loro hanno prevalentemente file immagine), quindi non tenendo conto della prossimità IP → di contro, se non c'è coerenza con un possibile vicino, questo viene scartato, e si farà richiesta ai SUOI vicini.

Progettare una rete overlay garantisce una grande flessibilità, in quanto non si è vincolati alle scelte fatte per IP (sta tutto sullo strato di applicazione).
Quindi si potranno decidere topologia, mantenimento, tipi di messaggi, protocollo, uso di TCP o UDP, mantenendo trasparente la rete fisica alla base.

Esempi di overlay

- DNS (overlay in cui ogni server DNS ha un suo superiore (vicino) anche se non ha alcun rapporto basato su IP)

- Router BGP e i loro rapporti di peering
- CDN (Content Distribution Network)
- multicast applicativi, che non usano indirizzi di classe D
- Applicazioni P2P

Principalmente, le applicazioni P2P prevedono che si acceda al relativo client P2P, ci si connetta a internet per ottenere un nuovo IP per ogni connessione e si registri i proprio contenuto nel sistema P2P.
Si potranno inviare anche delle query a tutti gli indirizzi di overlay per sapere se qualcuno dispone di un certo file. La richieste viaggia per l'overlay, fino a che non si raggiunge qualcuno che dispone di ciò che si sta cercando.
Quando si trova, i due attori si scambiano l'IP, in modo da creare una "media session" al di fuori dell'overlay, tramite la quale trasferire i dati. Specularmente, altri si possono connettere all'host utilizzato in modo da usufruire dei file messi a disposizione.

Chiaramente, non tutti i file che si VOGLIONO condividere non sempre coincidono con i file di cui si ha il DIRITTO di condivisione. I sistemi P2P aprono scenari per la violazione di diritti di copyright, violazioni che possono essere:

- ***dirette***: Gli utenti finali che scaricano o caricano lavori protetti da copyright (contraffattore contributivo)
- ***indirette***: Si ritiene una singola persona responsabile di azioni di altri, in quanto essa contribuisce alla diffusione di materiale protetto (contraffattore indiretto, in quanto può avere in primo grado sviluppato o contribuito alla diffusione, oppure addirittura in secondo grado omesso controlli sui sottoposti per lo sviluppo di tali reti di overlay)

Instant Messaging
Anche questo sfrutta le reti di overlay, ma non ha il problema delle precedenti applicazioni. Si sfrutta la rete di overlay per creare rapporti e instaurare connessioni tra i vari utenti. Si pensi a Skype (sistema P2P VoIP).

Calcolo distribuito P2P
Le reti di overlay possono essere usate anche per la suddivisione di compiti di calcolo distribuito per conto della comunità. Qualcuno (un superpeer centrale) riceve la disponibilità e trasmette un compito da elaborare.
Es. "seti" (utilizzo distribuito della trasformata di Fourier veloce per la ricezione di trasmissioni da parte di specie aliene intelligenti).

M3 U3 L1 SMTP

Gli elementi della posta elettronica principali sono:

- user agent UA (programma per la scrittura della posta)
- mail server MS (invio dei messaggi via SMTP ad altri MS)
- mail box MB (directory che contiene i messaggi generati da un utente tramite il suo UA)

Se lo user agent non è sulla stessa macchina del mail server (client di posta locali vs web mail) invia tramite SMTP il messaggio al MS, diversamente salva direttamente nella directory di spooling.

Il MS analizza quindi chi è il destinatario e si mette in contatto via SMTP con questa macchina destinataria e gli invia il messaggio.
A livello di destinatario, anche qui lo UA può essere sulla stessa macchina del MS (e quindi si sfrutta lo smistamento a livello di directory per trasferire i messaggi nella sua MB), diversamente possono venir mantenuti nella directory in attesa che gli UA si colleghino e scarichino i messaggi tramite appositi protocolli (POP).

Quindi SMTP viene usato per inviare il messaggio dallo UA al MS (unidirezionale), e anche per la consegna tra MS (bidirezionale).
SMTP usa un socket TCP sulla porta 25 per trasferire la posta elettronica in modo affidabile da client a server.
La ritrasmissione intermedia (cioè l'utilizzo di terzi a cui appoggiarsi prima di recapitare il messaggio) è possibile ma è un caso non ben visto e non frequente.

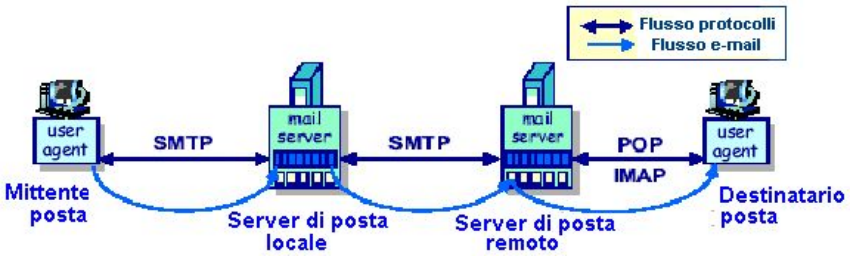
Le fasi del protocollo SMTP sono:

- handshaking (detto anche HELO o saluto) (non si tratta di un'autenticazione, non ci sono user e pwd)
- trasferimento di messaggi
- chiusura della connessione

Il tipo di interazione è client/server (uno manda dei comandi in formato ASCII NVT e l'altro risponde dei codici di stato e opzionalmente delle stringhe, e si ha come caratteri terminatori CR-LF (/r /n)).

Protocollo POP
Il POP (Post Office Protocol) consente l'accesso dell'utente alla posta elettronica consegnata alla sua mailbox.

Processo di consegna della posta elettronica



(esistono delle varianti, come IMAP, che permettono di leggere senza prelevare i messaggi dalla mailbox sul server di posta remoto).
L'unico tempo morto del protocollo può essere la coda di intasamento del server di posta locale.

Esempio di una interazione SMTP
Il client SMTP stabilisce la connessione TCP con il server "hamburger.edu" alla porta 25.

- SMTP non è standard in quanto il server "parla per primo", anche se la connessione è stata aperta da client.

```
Server: 220 hamburger .edu
Client: HELO crepes.fr (specificazione identità client)
S: 250 HELO crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok (accettazione del mittente)
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu... Recipient ok (accettazione del destinatario)
C: DATA (identificatore dell'inizio dei dati da trasmettere)
S: 354 Enter mail, end with "." on a line by itself (istruzioni dal server)
C: Do you like ketchup?
C: .
S: 221 hamburger.edu closing connection
```

M3 U3 L2 MIME
Formato dei messaggi di posta

- From:
- To:

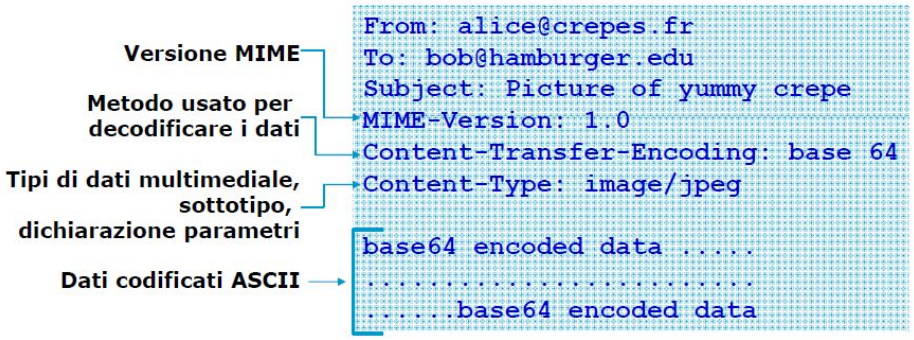
- Subject:

Poi altri come Cc, Ccn
E poi c'è il corpo, composto da caratteri ASCII.
Tra intestazione e corpo c'è una riga vuota.

Estensioni multimediali MIME

SMTP richiede che tutti i dati siano caratteri ASCII a 7 bit. Tutti i dati non ASCII devono essere codificati come stringhe ASCII e per far ciò è necessario aggiungere dei metadati nell'intestazione (ossia il tipo di contenuto).

Lo standard per far ciò prende il nome di MIME



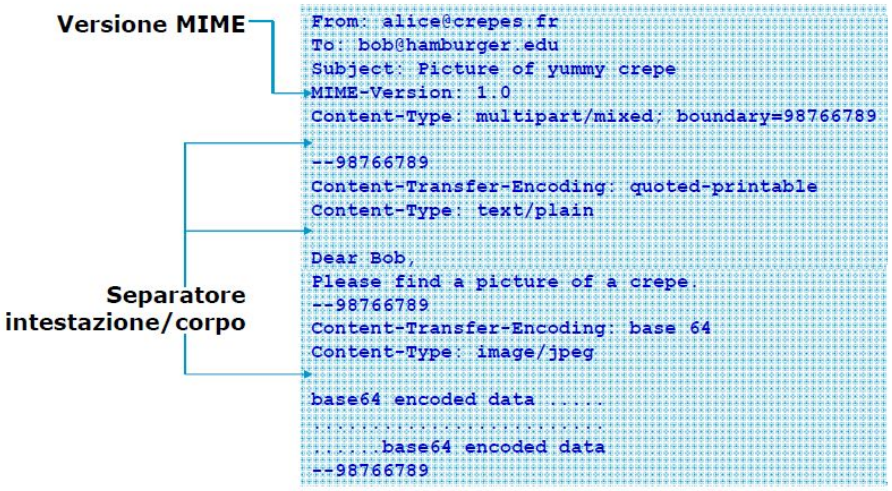
Tipi di MIME

• ***Testo*** → Sottotipi: plain, html
• ***Immagini*** → Sottotipi: jpeg, gif
• ***Audio*** → Sottotipi: basic (8-bit μ-law encoded), 32kadpcm (32 kbps ADPCM).
• ***Video*** → Sottotipi: mpeg, quicktime
• ***Dipendenti dall'applicazione*** → Altri dati che devono essere elaborati dal lettore prima che siano "visualizzabili" → Sottotipi: msword,octet-stream (formati binari specifici per determinate applicazioni, non visualizzabili).

Lo schema utilizzato sarà: `Content-Type: <type>/<subtype>[; <parameters>]`
Es.

```
Content-Type: text/plain; charset=us-ascii
Content-Type: application/pdf; filename=foo.pdf
```

Nel caso di messaggi composti da "testo + dati multimediali" si introducono dei separatori.



Un messaggio può contenere anche vari tipi di contenuti multimediali (un content-type per ciascuno)

M3 U3 L3 La posta elettronica

Esistono diversi modi non coperti dallo standard STMP per il recapito dei messaggi. Spesso e volentieri il mail server ricevente si limita a salvare sul suo disco i messaggi e ciascun destinatario utilizzerà un altro protocollo per trasferire i messaggi dal server smtp al computer (POP o IMAP).

SMTP ha un dualismo: il mittente annuncia chi sta inviando il messaggio sia come messaggio di protocollo che come nello header. La possibile differenza tra questi due (che dovrebbero essere uguali) può creare problemi.

Punti deboli di SMTP
Il protocollo è nato come semplice in modo che fosse facilmente parallelizzabile. Questo però si è tradotto nella presenza di vulnerabilità:
SMTP non prevede alcun controllo dell'identità del mittente rispetto al destinatario (si potrebbe fare una query DNS, ma ciò non è previsto dal protocollo).
In più c'è il discorso della duplicazione che permette di introdurre delle discrepanze.
Es.

```
from 159.149.70.1 by pollon (envelope-from <caio@crema.unimi.it>, uid 201) 08
Dec 2008 18:42:20 -0000
```

→ Questo campo dice che il messaggio è stato ricevuto dal server SMTP che si chiama 'pollon' (come dice la clausola 'by pollon') e proviene da un MTA di cui non è noto il nome, ma che ha l'indirizzo di rete 159.149.70.1 (dove quindi il mittente non si è qualificato nel protocollo ma solo nell'invio del messaggio).

Inoltre, la clausola `envelope-from <caio@crema.unimi.it>` specifica che nel messaggio, nel campo "from" ci dovrà essere caio@crema… ma ciò non è pragmatico, in quanto il campo From potrebbe comunque contenere qualcosa di diverso.
C'è poi da dire che gli eventuali mismatch possono essere dovuti sia alla cattiva programmazione del SMTP mittente, ma anche a volute confusioni create da hacker o spammer.

Il minimo che si può fare sarebbe una DNS query con il campo "envelope-from", così da confrontare l'IP ottenuto con quello ottenuto nel protocollo.
Questo però potrebbe dare risultati incoerenti non solo pe volute manipolazioni, ma anche perché l'editor di posta elettronica può aver inserito delle informazioni abbreviate nel envelope-from, rendendolo intraducibile via DNS.

Inoltre il campo "Received" si può ritenere affidabile il server mittente che l'ha creato.
Stesso discorso per il campo "from", che riteniamo affidabile perché l'ha controllato il server mittente (per scoprire la provenienza dell'IP si dovrà utilizzare un reverse DNS, o "whois".

Esempio di una query whois:

```
# ARIN WHOIS database, last updated 2008-12-08
19:10 % Information related to '159.149.0.0 -
159.149.255.255'
inetnum:   159.149.0.0 - 159.149.255.255
netname:   UNIMINET
descr:     Universita' degli Studi di Milano
country:   IT
remarks:   To notify abuse mailto: cert@garr.it
remarks:   Multiple-Lans of Milan University
```

Questa conoscenza trasmessa permetterà di configurare i server accettare (whitelist) o rifiutare (blacklist) messaggi in arrivo da determinati server.

M3 U3 L4 Lo spam

Secondo molti, la data d'inizio dello spamming commerciale è il 1994 con l'utilizzo di un programma per l'invio sistematico del messaggio a centinaia di gruppi Usenet per pubblicizzare i servizi di ottenimento della green card. Solo l'anno successivo, Jeff Slaton riuscì ad ottenere lo stesso risultato dissimulando l'identità del mittente così da non far risalire a chi stava inviando i messaggi.
Lo spam è tutt'ora un grande problema.

Lo spammer struttura la sua attività in due parti:

- reperimento degli indirizzi dei destinatari
- trovare un server SMTP disponibile a recapitare i messaggi spam agli indirizzi (facile, in quanto è possibile consultare i file di zona cercando i campi "MX".

Il server di SMTP potrebbe comunque adottare una forma di blacklisting per gli IP dei programmi mittenti (alla fine del 90 si corredava il server STMP di un "killfile" con gli IP delle macchine dalle quali non si desiderava ricevere posta da inviare)
Questo però è facilmente evitabile, basta utilizzare degli "***open relay***", ossia una funzionalità dei server SMTP che consente di usarli come intermediari per far recapitare il messaggio al server ricevente.
Sendmail di unix permette di utilizzare (e anzi, fino alla v5 era di default) la funzionalità open relay.
Quindi per lo spammer è sufficiente trovare dei server mail che ancora eseguano tale funzionalità. Questo non impedisce il blacklisting, ma lo rende molto difficile. Per evitare il problema si dovrebbero configurare i vari server SMTP in modo che non operino in open relay.

Tuttavia, nonostante l'applicazione di ciò, gli spammer hanno trovato nuove tecniche di recapito:

- ***Relay multi-hop***: le reti dei provider internet si affidano a più server SMTP, di cui alcuni usati per l'invio di posta tra utenti dello stesso dominio, e altri "di confine" usati per inoltrare la posta verso l'esterno (questi devono quindi cercare di attuare una sorta di relay). Lo spammer potrebbe usare un server di posta interno e conseguentemente usare un server di posta "di confine" che utilizza relay (il quale, potendo non essere sempre lo stesso, è più difficile da blacklistare).
- ***Spedizione con Dynamic Addressing***: se mi configuro un sendmail sulla mia stessa macchina si può contare sul fatto che il sendmail potrà essere non visto sempre dallo stesso IP, rendendo molto difficile il blacklisting tramite killfile
- ***Open proxy***: molte organizzazioni usano proxy per consentire ai loro computer di rete locale di condividere la connessione internet. I proxy sono spesso mal configurati e permettono ad host parassiti di attivare connessioni proxy.

M3 U3 L5 Le difficoltà nel combattere lo spam

Esistono diverse tecniche anti-spam applicabili:

- Cercare di bloccare gli IP dinamici (non esiste però un semplice test per stabilire se un IP è dinaimico o meno, e inoltre è vulnerabile all'utilizzo dell'open proxy)
- Attivazione degli "honeypot", costituiti da finti server SMTP poco scrupolosi e caselle di posta non vere, per cercare di catturare gli IP dei server SMTP usati dagli spammer. In pratica però la latenza per diffondere le segnalazioni degli honeypot li rende utili più per contromisure legali che per reazioni in tempo reale.

La tecnica però che ha preso più piede è quella del ***filtraggio orientato al contenuto***, che può essere effettuata ricalcando una impronta comune ai messaggi di spam della stessa "famiglia" (calcolando l'hash del messaggio e confrontando con l'hash di messaggi notoriamente spam).
Il software "SpamAssassin" combina le tecniche di blacklisting con quelle di filtraggio per assegnare ad ogni messaggio una probabilità di spam, ed agire di conseguenza. Questo software si basa sulla rete distribuita e collaborativa di identificazione dello spam "Vipul's Razon", il cui catalogo di esempi di spam viene costantemente aggiornato.
Teorema di Bayes

La soluzione di bayesiana si basa sullo studio statistico del contenuto dei messaggi.
Un filtro bayesiano decide se il messaggio è spam o no in base alle parole contenute in esso.
Questo funziona in base allo studio probabilistico, e infatti:

- si stima qual'è la probabilità che un msg di spam (H) contenga una determinata parola (O) → **P(O|H)**
- secondo il teorema di Bayes **P(H|O) = P(O|H) * P(H) / P(O)**
- **P(H)** si stima esaminando la cartella "Junk Mail" dove l'utente mette lo spam e contando quanti sono i messaggi di spam rispetto al totale dei messaggi
- **P(O)** si stima contando quanti messaggi contengono la parola incriminata sul totale dei messaggi (spam o no) ricevuti dall'utente

Il P(H|O) sarà la probabilità che cerchiamo.

Gli spammer hanno risposto a queste tecniche di blocco. Hanno applicato delle "list splitting" per personalizzare a seconda del destinatario il contenuto delle email, e hanno spostato la parte informativa dei loro messaggi all'interno di immagini → è molto difficile scrivere un anti-spam bayesiano basato sul riconoscimento di caratteri all'interno di immagini

Contromisure
Quello su cui si mira adesso è l'utilizzo di tecniche per evitare che gli "harvester" (o "spambot") riescano a reperire indirizzi email a cui spedire lo spam.
Per far questo si cerca di oscurare che meglio possibile gli indirizzi presenti sulle pagine web:

- utilizzando delle sintassi con placeholder
- utilizzando le codifiche HTML (tipo &#64 per la chiocciola)
- utilizzando delle immagini per mostrare gli indirizzi email
- utilizzando tecniche crittografiche (dove la mail viene decifrata solo nel momento di utilizzo)

M3 U4 L1 World Wide Web

Si tratta del sistema C/S più esteso al mondo, e si basa sullo scambio di messaggi secondo il paradigma request-response.
Si seleziona l'URL all'interno del browser → questo richiede al server la pagina → il server risponde con un messaggio che contiene un'intestazione e il contenuto → il browser in base ai contenuti dell'intestazione visualizzerà correttamente la pagina → se la pagina contiene altri elementi, il browser dovrà richiederli separatamente.

HTTP (HyperText Transfer Protocol)
Protocollo a livello applicativo per sistemi informativi ipermediali, collaborativi e distribuiti (orientato agli oggetti, generico e stateless - ogni richiesta è a sé).
Il protocollo si basa sulla porta 80, che è "firewall traversal" (ossia è sempre aperta su tutti i sistemi, escluse rarissime eccezioni).
> HTTP 1.0 mappava ogni singola richiesta su una diversa connessione TCP, così che anche ogni risorsa aggiuntiva per cui andavano eseguite altre richieste HTTP richiedeva più connessoni.
> HTTP 1.1 invece manteneva una connessione persistente di default, che rimane aperta finché le controparti non decidono di chiuderla (o si verifica il timeout). Questo ha permesso di ridurre le connessioni e quindi l'overhead.
HTTP è stateless (Due richieste successive con gli stessi contenuti ottengono la stessa risposta), ma ci sono tipi di applicazioni che richiedono sessioni stateful (che richiedono che la risposta sia dipendente dalle richieste precedenti)

M3 U4 L2 HTTP

Basato sulla porta TCP 80 è il protocollo usato per la comunicazione tra browser e server web. Introdotto con la RFC 1945 della IETF.

Per identificare le risorse da ottenere, HTTP usa degli identificativi:

- URI (Uniform Resource Identifier): Informazioni relative a una risorsa
    - URN (Universal Resource Name): → Il nome della risorsa nel namespace
    - URL (Uniform Resource Locator): → sono degli URN che contengono anche le informazioni per trovare la risorsa

URL è composto da:
Protocollo → Nome dell'host → Porta → Percorso di FS della risorsa → nome della risorsa

Metodi HTTP

• ***GET***: Recupera un URL dal server
- richiede una pagina
- esegue un programma CGI (eventualmente passandogli gli argomenti inviati insieme all'URL)

• ***POST***: Metodo preferito per elaborare le form
- Esegue un programma CGI
- Dati parametrizzati nel payload HTTP in sysin (dipende dal sistema operativo)
- Più sicuro e privato

• ***PUT***: Usato per trasferire un file dal client al server

• ***HEAD***: Richiede solo l'header dello stato degli URL (recupera l'URL solo se non è nella cache locale e se la data è più recente di quella della copia memorizzata nella cache)

Le richieste vengono inviate dal client al server e sono costituite dall'header HTTP e da un URL.
L'header conterrà il tipo di contenuto (o il tipo MIME), la lunghezza del contenuto, lo user agent e i tipi di contenuto che lo user agent può gestire.

A seguire l'analisi dei campi dell'header HTTP:

- ***From***: Indica l'indirizzo dell'utente (browser) che ha fatto la richiesta. Potrebbe non essere corrispondente al nome dell'host (ci potrebbe essere un proxy), e deve essere un indirizzo di posta elettronica valido (difficilmente controllato)
- ***Accept***: Elenco di schemi accettati dal client. Dispone di una sua grammatica:
  `<field> = Accept: <entry> * [,<entry>]`
  `<entry> = <content type> *[;<param>]`
  `<param> = <attr> = <float>`
  `<attr> = q / mxs / mxb`
  `<float> = <ANSI-C floating point >`
  Se non viene trovato il campo, il testo normale è di default
  es. `Accept: text/html,application/xhtml+xml,application/xml;`
- ***Accept-Encoding***: è come Accept, ma specifica quali tipi di compressione sono gestibili dal browser (es. `Accept-Encoding: x-compress;x-zip`)
- ***User-Agent***: consente al server di sapere con quale browser sta dialogando. Ha una grammatica:
  `<field> = User-Agent: <product>`
  `<product> = <word> [/<version>]`
  `<version> = <word>`
  Spesso e volentieri si usano però delle stringhe convenzionali (es. `User-Agent: IBM WebExplorer DLL /v960311`)
- ***Referer***: indica al server l'URL da quale ha ottenuto l'url dal quale sta proveniendo
- ***Authorization***: gestisce l'autenticazione tramite username e password, oppure tramite kerberos (es. `Authorization: kerberos kerberosparameters`)
- ***Charge-To***: informazioni di addebito (per capire a chi vanno attribuiti i costi di gestione del server sulla base delle chimate)

- ***Pragma***: ha lo stesso formato di Accept ed è rivolto al proxy, perché si vuole ottenere il file dal server originale e non dalla cache del proxy. Il solo Pragma definito al momento è <<no-cache>>. L'utilizzo può essere anche quello di forzare la connessione al server perché questo logghi la sua richiesta.
- ***If-Modified-Since***: tramite questo campo viene specificata una data, e se il server ha una versione più recente della risorsa richiesta la manda, altrimenti manda un header che dice che la risorsa è ancora quella e che quindi si può prelevare dalla cache del browser (rispondendo con un codice 304)

Pacchetti di risposta
In primo luogo il server specifica in risposta al client un ***header di stato***, composto da un ***codice***.
La prima cifra del codice identifica la famiglia del codice:

- 1xx: rimandato a un uso successivo
- 2xx: l'azione è stata ricevuta con successo, compresa ed accettata
- 3xx: per soddisfare la richiesta sono necessarie ulteriori azioni
- 4xx: la richiesta del client è sintatticamente scorretta
- 5xx: il server non può soddisfare la richiesta

Header delle risposte
Inviato dal server al browser e composto da:

- header di stato
- header di entità:
    - Content-Encoding:
    - Content-Length:
    - Content-Type:
    - Expires:
    - Last-Modified:
    - extension-header

E poi c'è il corpo, che può essere HTML o un qualunque tipo MIME.

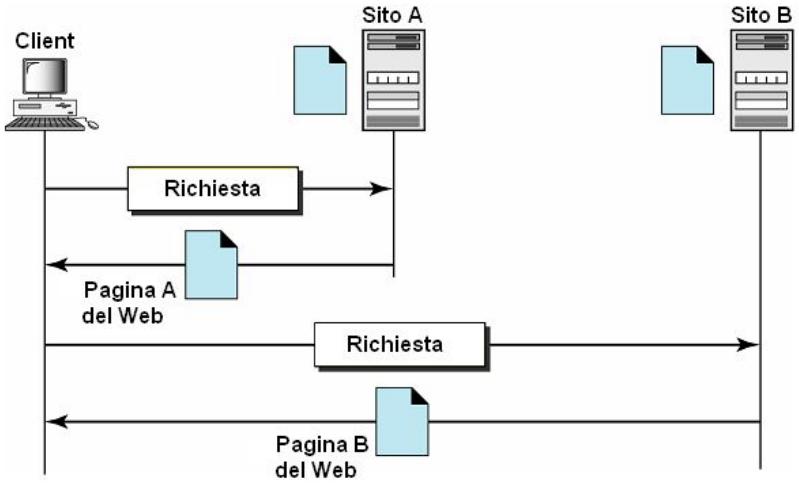
HTTP è evidentemente stateless, però tramite delle RFC è stato possibile pervenire a delle integrazioni a favore del mantenimento dello stato:

- ***cookies***, che vengono trasmessi ogni volta che si fa una richiesta all'url
- ***invio tramite richieste get il risultato di precedenti risposte get*** (salvando lo stato a lato client), opp
- ***codice di sessione***, che viene reinviato dal client ogni volta che si accede al server per quella sessione (salvando lo stato a lato server)).

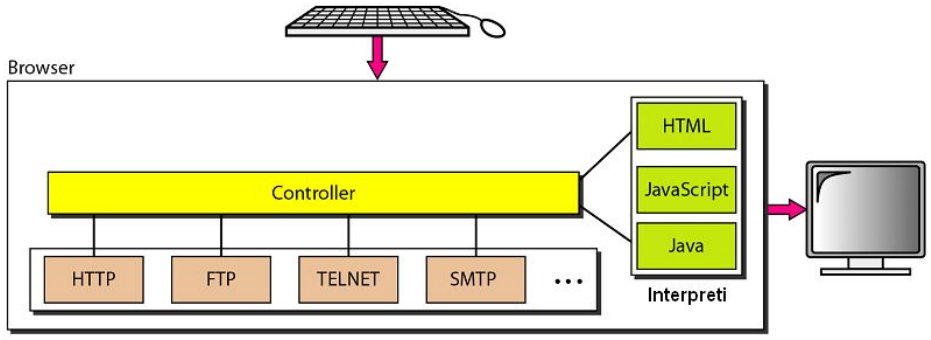
Queste tecniche consentono di rendere HTTP stateful.

M3 U4 L3 WWW e HTTP

Architettura del WWW

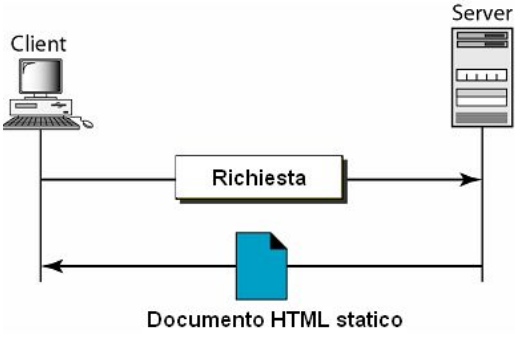


Il browser è diventato una vera e propria macchina, in grado di attivare stack di protocollo applicativo differenti (non solo HTTP, ma anche FTP, TELNET ecc.), attraverso un controller, il quale è in grado di passare le risposte in arrivo ad opportuni interpreti che si occuperanno di effettuare il rendering grafico o eseguire computazione.

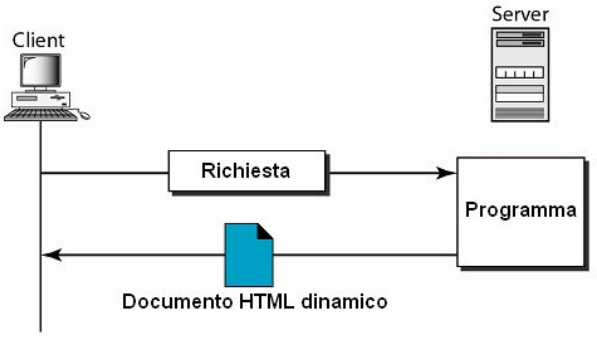


Nel WWW i documenti possono essere raggruppati in **statici**, **dinamici** e **attivi** (in base al momento in cui viene calcolato il contenuto del documento).

Documento statico

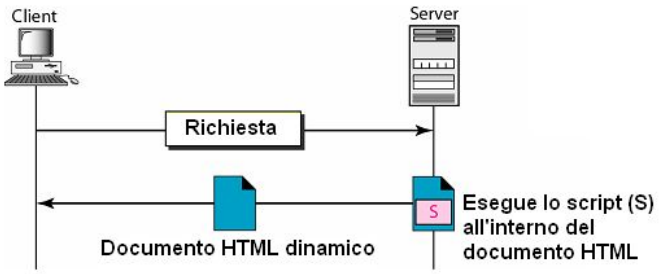


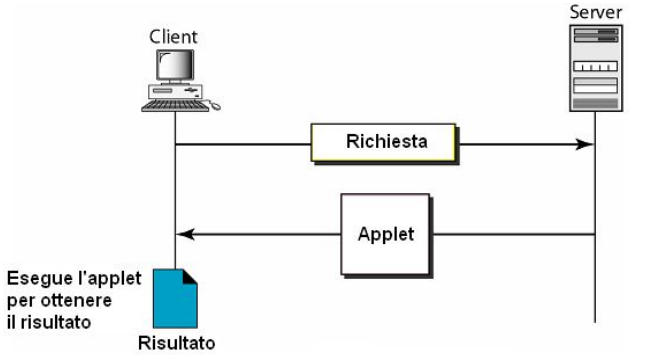
Documento dinamico usando CGI



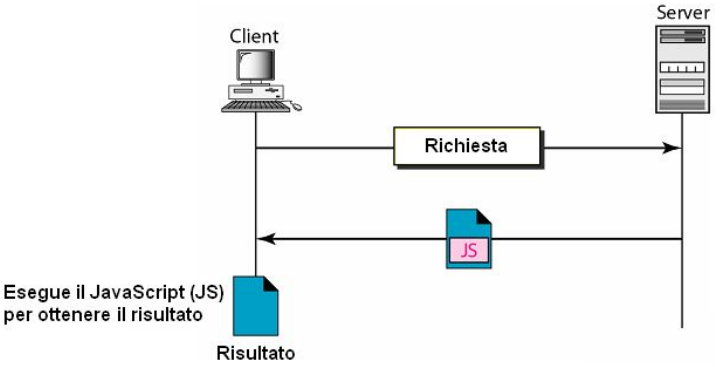
Documento dinamico usando uno script lato server

Documento attivo usando applet Java

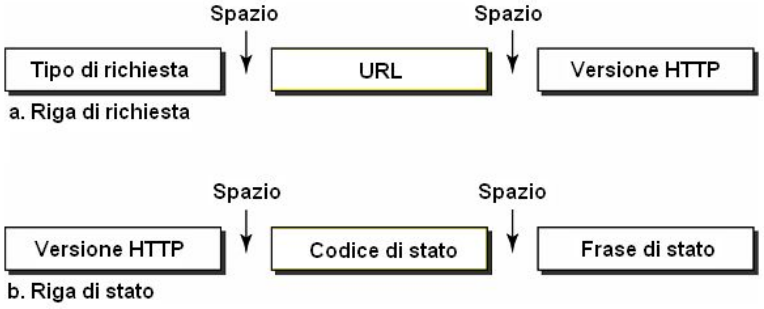




Documento attivo usando uno script lato client



Righe di richiesta e di stato HTTP



Metodi HTTP (completo)

Metodo	Azione
GET	Richiede un documento dal server
HEAD	Richiede informazioni su un documento, ma non il documento stesso
POST	Invia alcune informazioni dal client al server
PUT	Invia un documento dal server al client
TRACE	Riproduce la richiesta in ingresso
CONNECT	Riservato
OPTION	Chiede informazioni sulle opzioni disponibili

Codici di stato più usati

Codice	Frase	Descrizione
Informativo		
100	Continua	La parte iniziale della richiesta è stata ricevuta e il client può continuare con la sua richiesta
101	Passaggio da un protocollo all'altro	Il server adempie a una richiesta del client di passare ai protocolli definiti nell'header aggiornato
Esito positivo		
200	Esito positivo	La richiesta ha esito positivo
201	Creato	Viene creato un nuovo URL
202	Accettato	La richiesta è stata accettata, ma non è stata ancora completata
204	Nessun contenuto	Non c'è alcun contenuto nel corpo

Codice	Frase	Descrizione
Reindirizzamento		
301	Spostato permanentemente	L'URL richiesto non è più usato dal server
302	Spostato temporaneamente	L'URL richiesto è stato spostato temporaneamente
304	Non modificato	Il documento non è stato modificato
Errore del client		
400	Richiesta non valida	Nella richiesta c'è un errore di sintassi
401	Non autorizzata	La richiesta non ha l'autorizzazione idonea
403	Proibito	Il servizio è negato
404	Non trovato	Il documento non viene trovato
405	Metodo non consentito	Il metodo non è supportato in questo URL
406	Non accettabile	Il formato richiesto non è accettabile
Errore del server		
500	Errore interno del server	Nel server si è verificato un errore, ad esempio un crash
501	Non implementata	L'azione richiesta non può essere eseguita
503	Servizio non disponibile	Il servizio temporaneamente non è disponibile, ma può essere richiesto in futuro

Header http

- **Generali**

Header	Descrizione
Cache-control	Specifica informazioni sul caching
Connection	Indica se la connessione deve essere chiusa o meno
Date	Indica la data corrente
MIME-version	Indica la versione MIME usata
Upgrade	Specifica il protocollo di comunicazione preferito

- **Richiesta**

Header	Descrizione
Accept	Indica il formato medio che il client può accettare
Accept-charset	Indica l'insieme di caratteri che il client può gestire
Accept-encoding	Indica lo schema di codifica che il client può gestire
Accept-language	Indica il linguaggio che il client può accettare
Authorization	Mostra quali permessi ha il client
From	Mostra l'indirizzo di posta elettronica dell'utente
Host	Mostra l'host e il numero di porta del server
If-modified-since	Invia il documento se è più recente della data specificata
If-match	Invia il documento solo se corrisponde al tag dato
If-non-match	Invia il documento solo se non corrisponde al tag dato
If-range	Invia solo la parte del documento che manca
If-unmodified-since	Invia il documento se non è cambiato dalla data specificata
Referrer	Specifica l'URL del documento collegato
User-agent	Identifica il programma client

- **Risposta**

Header	Descrizione
Accept-range	Mostra se il server accetta il range richiesto dal client
Age	Mostra l'età del documento
Public	Mostra l'elenco di metodi supportati
Retry-after	Specifica la data dopo la quale il server è disponibile
Server	Mostra il nome del server e il numero della versione

Transfer-Encoding: chunked (la risposta viene spezzettata in caso sia lunga)
→ In caso, all'interno della response, si possono trovare dei codici esadecimali che identificano la lunghezza del pezzo)
Transfer-Encoding: compress (il messaggio è compresso)
Transfer-Encoding: deflate (il messaggio è compresso con l'algoritmo deflate)
Transfer-Encoding: gzip (il messaggio è compresso con l'algoritmo gzip)
Transfer-Encoding: identity (il messaggio non è compresso)

- **Entità (info a corredo della risposta per la gestione della stessa da parte del client)**

Header	Descrizione
Allow	Elenca i metodi validi che possono essere usati con un URL
Content-encoding	Specifica lo schema di codifica
Content-language	Specifica il linguaggio
Content-length	Mostra la lunghezza del documento
Content-range	Specifica il range del documento
Content-type	Specifica il tipo medio
Etag	Indica un tag di entità
Expires	Indica la data e l'ora in cui il contenuto può cambiare
Last-modified	Indica la data e l'ora dell'ultimo cambiamento
Location	Specifica la posizione del documento creato o spostato

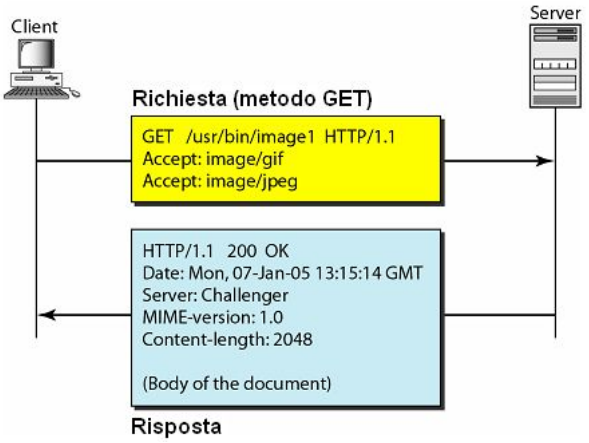
Esempio di esercizio esame

- Usiamo il metodo GET per recuperare un'immagine con il percorso /usr/bin/image1

- La riga di richiesta mostra il metodo (GET), l'URL e la versione HTTP (1.1)
- L'header ha due righe che mostrano che il client può accettare immagini nel formato GIF o JPEG

Riflessioni...

- La richiesta non ha un corpo
- Il messaggio di risposta contiene la riga di stato e quattro righe di header
- Le righe di header definiscono la data, il server, la versione MIME e la lunghezza del documento
- Il corpo del documento segue l'header



Esempio risposta completa

```
HTTP/1.1 200 OK
Date: Sat, 19 May 2007 13:49:37 GMT
Server: IBM_HTTP_SERVER/1.3.26.2 Apache/1.3.26 (Unix)
Set-Cookie: tracking= sklk87979HHIhhkhJKNAH8768983
Pragma: no-cache
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Content-Type: text/html;charset=ISO-8859-1
Content-Language: en-US
Content-Length: 24246
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html lang="it">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
```

M3 U5 L1 Gestione delle reti

I sistemi autonomi comprendono centinaia o migliaia di componenti hardware e software che interagiscono tra loro. Il concetto di gestione della rete comprende:

- sviluppo, integrazione e coordinamento di hardware, software e personale umano;
- controllo, interrogazione, configurazione e valutazione di una rete IP e delle sue risorse.

Il tutto per consentire alla rete di funzionare a costi contenuti.

Quest'area prevede delle sub-aree funzionali di gestione della rete:

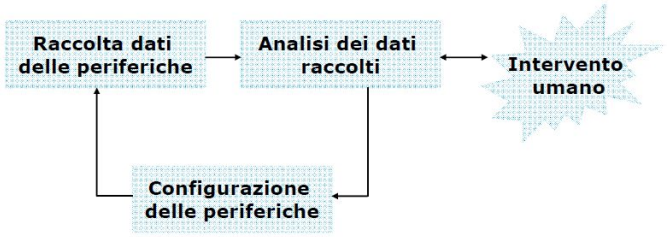
• Gestione delle risorse.
• Gestione della sicurezza.
• Gestione delle prestazioni.
• Gestione delle configurazioni.
• Gestione dei guasti.

La gestione delle reti è orientata agli oggetti (oggetti = dispositivi o parti di dispositivi nelle reti).

Questi saranno rappresentati esplicitamente con l'*insieme dei loro attributi di configurazione*, e le azioni che si svolgeranno su questi oggetti saranno dei ***cambiamenti di stato*** di questi oggetti.

Le rappresentazioni degli oggetti saranno quindi delle "entries" in un database distribuito (il Management Information Base, o MIB - quindi ogni dispositivo mantiene il proprio stato)

Il processo di gestione del MIB funziona come segue:



Se dall'analisi perviene la necessità di dover modificare la configurazione, si invieranno dei messaggi di SET, diversamente si rimarrà in questo loop di controllo dello stato degli oggetti.

Il CMIP (Common Management Information Protocol) fu creato negli anni '80 col proposito di eseguire questi GET e SET sulle configurazioni dei dispositivi, ma non ebbe lo stesso successo dei protocolli a seguire.

M3 U5 L2 SNMP (Simple Network Management Protocol)

Questo protocollo definisce un meccanismo per la gestione remota di dispositivi di rete (router, bridge ecc.).
Il principio fondamentale di questo protocollo è gestire i dispositivi manipolando i valori di opportune variabili.
Su questi attributi il protocollo può operare in "test" (per verificare il valore) e in "set" (per modificare il valore).

I messaggi usano il protocollo UDP e le porte 161 per le richieste/risposte, e 162 per le notifiche.
– ***GetRequest***: attraverso cui il manager recupera il valore di un oggetto;
– ***GetNextRequest***: con cui recupera l'oggetto successivo (nell'ordinamento lessicografico definito nella MIB);
– ***SetRequest***: attraverso cui il manager assegna un valore ad un oggetto.

– ***GetResponse***: formato di risposta univoco ai messaggi di richiesta

– ***Trap***: (dal dispositivo) avvisare il monitor/manager del cambiamento di valore

SNMP è attualmente usato in versione autenticata MA NON CRITTOGRAFATA (consentendo ad uno sniffer di avere info sullo stato dei dispositivi della rete).
Si possono però definire delle "***community***", specificando l'accesso a determinate serie di variabili (r/w, solo r, niente).

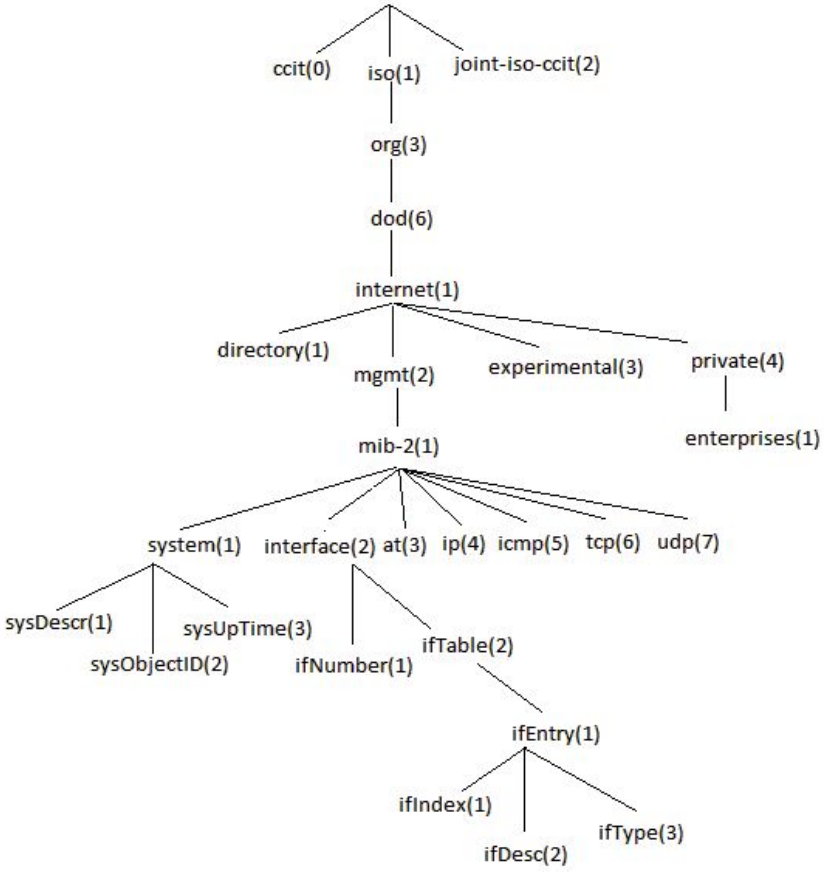
Esiste uno standard internazionale che definisce i valori degli attributi dei dispositivi di telecomunicazioni: ASN.1.
ASN.1 ha una sua BER (Basing Encoding Rule) utilizzata per esprimere i tipi di dati in SNMP.
I dati sono codificati come triple Type,Length,Value
I tipi di dato possono essere:

- *un sottoinsieme dei tipi di base ASN.1:* intero, stringa di ottetti, identificatore di oggetti ("nome della variabile"), sequenza
- *tipi definiti da SNMP*: strumento di controllo (gauge), contatore, indirizzo IP ecc.)

Tutti gli oggetti ASN.1 sono custoditi nella MIB, un DB in formato testo che riunisce tutte le info disponibili su tutti i dispositivi installati sulla rete.

Ogni dispositivo espone degli attributi standard codificati dalle RFC standard, ma espone anche delle variabili specifiche del dispositivo date dal produttore.

Esempio di una MIB:




```
-- the Interfaces group

        -- Implementation of the Interfaces group is mandatory for
        -- all systems.

        ifNumber OBJECT-TYPE
           SYNTAX  INTEGER
           ACCESS  read-only
           STATUS  mandatory
           DESCRIPTION
                "The number of network interfaces (regardless of
                their current state) present on this system."
           ::= { interfaces 1 }
```

Il gruppo di interfacce è un attributo obbligatorio per tutti i sistemi (e quindi tutti i dispositivi hanno questo attributo). Questo contiene il numero di interfacce presenti sul dispositivo, ed è (chiaramente) di sola lettura.

M3 U5 L3 Complementi di SNMP

La MIB di un dispositivo è un database in formato testuale mantenuto su ogni agent. Esso prevede un set di interrogazioni standard eseguibili su di essi, più delle informazioni inserite dai produttori.
La MIB si basa su due costrutti fondamentali:

- object type:
- module identity: consente di raggruppare definizioni di oggetti per definire un dispositivo più complesso

Es. di module identity

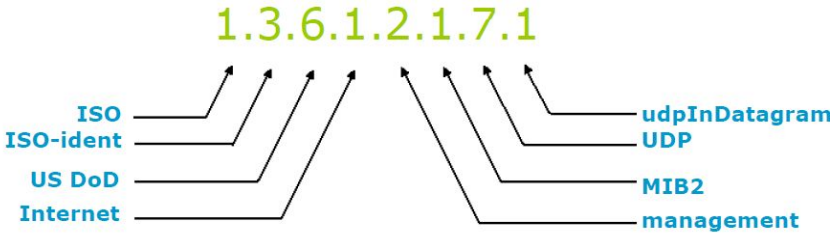
```
ipMIB MODULE-IDENTITY
      LAST-UPDATED “941101000Z”
```

```
      ORGANIZATION “IETF SNPv2 Working Group”
      CONTACT-INFO “Keith McCloghrie...”
      DESCRIPTION
            “Il modulo MIB per gestire implementazioni IP e ICMP escludendo però
            la gestione delle rotte IP.”
      REVISION “019331000Z”
.....
::={mib-2 48}
```

Modulo UDP

OID	Nome	Tipo	Commento
1.3.6.1.2.1.7.1	UDPInDatagr	Counter32	Numero di datagram consegnati a questo nodo
1.3.6.1.2.1.7.2	UDPNoPorte	Counter32	Numero di datagram non consegnabili per mancanza di applicazioni alla porta di destinazione
1.3.6.1.2.1.7.3	UDPInError	Counter32	Numero di datagram non consegnabili per altri motivi
1.3.6.1.2.1.7.4	UDPDatagrInv	Counter32	Numero di datagram inviati
1.3.6.1.2.1.7.5	UDPTabella	Sequence	Un elemento per ogni porta in uso dall’applicazione dà il numero di porta e l’indirizzo IP

I prodotti che acquistiamo contengono già il MIB definito, non sta a noi definire i moduli.
Gli OID vengono assegnati tramite una struttura gerarchica: alcuni moduli son composti da oggetti, i quali sono composti a loro volta da oggetti più semplici (come fosse un albero).
Es.



1 → ISO (siamo nell’albero di generazione degli OID di ISO)
3 → ISO-ident (si usa il particolare sottoalbero ISO)
6 → ISO definito dal dipartimento della difesa degli stati uniti
1 → Stiamo usando un tipo di oggetto usato nelle interreti (internet)
2 → Si fa riferimento agli oggetti di gestione
1 → Si intende che si sta utilizzando un oggetto che fa parte del core del MIB2
7 → Si tratta di un oggetto che ha a che fare con UDP
1 → Identificatore dello specifico oggetto

Per accedere ai dati utilizzando un client abilitato a SNMP si utilizzeranno diverse modalità:

- ***Modalità sincrona a richiesta/risposta***: si richiede esplicitamente ai vari dispositivi dal client di comunicarmi lo stato dei loro oggetti (che verranno mostrati a schermo)
- ***Modalità asincrona, o trap***: il client rimane in attesa, e quando uno stato cambia relativamente ad un certo attributo di un oggetto, questo viene comunicato e segnalato al client.

Le applicazioni attualmente mettono quasi sempre a disposizione delle interfacce grafiche per la gestione, rendendo trasparenti le specifiche degli oggetti SNMP.

M4 U1 L1 Interfaccia di programmazione di TCP/IP

La storica libreria messa a disposizione per l'accesso al livello di trasporto è la "***Socket Library***" (C, e Java)
I protocolli TCP e UDP usano le porte per mappare i dati in ingresso con un particolare processo attivo su un computer.

Le ***porte*** sono valori interi positivi di 16 bit, e ne esistono di conosciute (well-known port) come FTP (21), TELNET (23), SMTP(25), Login (513).
I servizi e i processi a livello utente in genere usano un numero di porta >= 1024.

L'idea del ***socket*** è quella di essere l'identificativo di un canale di comunicazione. *Forniscono un'interfaccia per la programmazione dell'accesso alla rete a livello trasporto*, ed è formato dalla quaterna
`IP Mittente, Porta Mittente, IP Destinatario, Porta Destinatario.`
Si basano sull'utilizzo di un descrittore di una comunicazione. Il fatto di avere un descrittore mira alla volontà di utilizzare le operazioni di I/O che già si utilizzano sui file per scrivere o leggere dalle connessioni di rete.

M4 U1 L2 Creazione di un socket

Il socket rappresenta un descrittore di una particolare connessione, però la connessione prevede due attori. Il socket rivestirà quindi un ruolo diverso per client e server: da un lato il server creerà un socket per accertare la richiesta di connessione dal client (chiamato "***socket di benvenuto***", mentre dall'altra il client creerà un socket per stabilire la connessione al socket del server.

Se tutto funziona, il server accetta la connessione e ottiene dal SO un nuovo socket legato ad una porta diversa (il socket di servizio, o "***di connessione***"), così da lasciare libero il socket libero per accettare connessioni da altri client.

Creazione di un socket

```
int socket(int domain, int type, int protocol)
```

- domain: può essere
  - AF_UNIX: socket sulla stessa macchina, gestiti dal sistema operativo (pipe per comunicazioni tra processi)
  - AF_INET: socket per connessione di rete
- type:
  - SOCK_DGRAM: per pacchetti UDP (socket di datagram)
  - SOCK_STREAM: per pacchetti TCP (socket di flusso)
  - RAW: accesso agli strati inferiori del protocollo
- protocol:
  - TCP
  - UDP
  - Può essere omesso, il sistema lo deduce dal tipo

Funzione bind

Associa e può riservare (in modo esclusivo) ad un socket una porta TCP o UDP a uso di un socket

```
int status = bind(sockid, &addrport, size);
```

- status: stato di errore, -1 se bind non ha avuto successo
- sockid: descrittore del socket
- addrport: di tipo struct sockaddr, contiene IP e porta della macchina (è possibile utilizzare dei caratteri di wildcard "*", per far sì che sia possibile la connessione da "any")
- size: la dimensione in byte della structure addrport

• SOCK_DGRAM
→ Se si spedisce solamente, bind non serve. Il sistema operativo trova lui una porta ogni volta che il socket invia un pacchetto.
→ Se si riceve, bind è indispensabile (serve per stabilire la porta destinataria da far usare al mittente)

• SOCK_STREAM
Qua il bind è necessario.
→ Destinazione determinata durante il setup della connessione (three-way handshake).
→ Non è necessario conoscere la porta da cui avviene l'invio (si può lasciare "*")
→ Durante il setup della connessione, il lato ricevente viene informato del numero di porta dal SO

Per entrambi questi tipi di socket sarà necessario specificare (con i dovuti "*") la quaterna di cui prima.

M4 U1 L3 Set up di connessione

Per SOCK_DGRAM non è necessario alcun setup di connessione, tutto ciò che serve è attribuire l'indirizzo del destinatario e spedire dati.
Per quanto riguarda TCP si hanno invece due tipi di partecipanti:

- passivo: aspetta un partecipante attivo per richiedere la connessione;
- attivo: inizia la richiesta di connessione al lato passivo.

La distinzione tra mittente e destinatario avviene solo in fase di set up della connessione, dopo di che la connessione sarà in full-duplex.

Il set up della connessione prevede 4 fasi:

1) Listen - attesa di richieste in ingresso (creazione socket, attribuzione parametri tra cui *)
2) Connect - Richiesta di connessione dal partecipante attivo (attribuzione di tutti i parametri)
3) Accept - accettazione di una richiesta (che comporterà lo spostamento sul nuovo socket di servizio)
4) Invio dati - fase di scambio dei dati condivisa tra attivo e passivo

Listen
Chiamata dal partecipante passivo, con i parametri dovuti ("*" su IP e porta destinatari)

```
int status = listen(sock, queueLen);
```

- status (valore di ritorno): 0 se ascolta, -1 in caso di errore;
- sock: descrittore del socket, intero;
- queueLen: numero massimo (intero) di partecipanti attivi che possono "attendere" di connettersi (num massimo di three-way handshake in attesa sul socket)

Listen non è bloccante.

Accept
Chiamata dal partecipante passivo

```
int s = accept(sock, &name, nameLen);
```

- s (valore di ritorno): intero, il nuovo socket (usato per trasferire i dati);
- sock: intero, il socket originale (su cui si è ascoltato);
- name: di tipo struct sockaddr, indirizzo del partecipante attivo (il SO riempirà i parametri mancanti con IP e porta del client che si sta connettendo)
- nameLen: sizeof(name): parametro valore/risultato, contiene la dimensione in byte di name.

Accept è bloccante → Aspetta che un client si connetta prima di tornare

M4 U1 L4 Funzioni di comunicazione via socket

Connessione da parte di un client

```
int status = connect(sock, &name, nameLen);
```

- status: 0 se connect ha successo, altrimenti -1;
- sock: intero, il descrittore del socket che deve essere usato nella connessione;
- name: di tipo struct sockaddr, indirizzo del partecipante passivo;
- nameLen:intero, valorizzata con sizeof(name).

Se fatto con UDP, la sua semantica sarà memorizzare localmente le informazioni del destinatario per l'invio dei messaggi

Connect è bloccante

Invio e ricezione di dati con TCP (dietro connessione)

```
int count = send(socket, &buf, len, flags);
```

- count (valore di ritorno): numero di byte trasmessi (-1 in caso di errore);
- socket: intero, descrittore del socket che gestisce la trasmissione - socket di servizio -;
- buf: char[], buffer che deve essere trasmesso;
- len: intero, lunghezza del buffer (in byte) da trasmettere;
- flags: intero, opzioni speciali, di solito sempre 0.

```
int count = recv(socket, &buf, len, flags);
```

- count (valore di ritorno): numero di byte ricevuti (-1 in caso di errore);
- socket: intero, descrittore del socket che gestisce la trasmissione - socket di servizio -;
- buf: void[], memorizza i byte ricevuti;
- len: intero, lunghezza del buffer (in byte) di ricezione;
- flags: intero, opzioni speciali, di solito sempre 0

Entrambe le chiamate sono bloccanti: la chiamata ritorna solo dopo che i dati sono stati inviati (al socket buffer) o ricevuti

Invio e ricezione di dati con UDP (senza connessione)

```
int count = sendto(sock, &buf, len, flags, &addr, addrLen);
```

- count (valore di ritorno), sock, buf, len, flags: uguale al parametro della funzione send() per TCP;
- addr: di tipo struct sockaddr, indirizzo della destinazione; (se non si è fatta la connect)
- addrLen: intero, contiene la dimensione in byte dell'indirizzo (sizeof(addr)).

```
int count = recvfrom (sock, &buf, len, flags, &name, nameLen);
```

- count, sock, buf, len, flags: uguale alla funzione recv();
- name: struct sockaddr, indirizzo della sorgente; (vuota nel momento della chiamata, ma viene valorizzata dal sistema operativo con indirizzo e porta del mittente)
- nameLen: sizeof(name), dimensione in byte del parametro name.

Entrambe le chiamate sono bloccanti, anche se non c'è la semantica di TCP

Chiusura della connessione

Importante perché il SO ha un numero limitato di socket gestibili (ogni socket richiede un buffer).

```
status = close(s)
```

- status: 0 se l'operazione ha avuto successo, -1 in caso di errore;
- s: il descrittore del socket che viene chiuso.

M4 U1 L5 Codifica degli indirizzi nella Socket Library

Nell'atto del "Bind" bisogna passare una struttura necessaria per definire la quaterna IP_m, Port_m, IP_d, Port_d.

Formato della struct

```
struct sockaddr_in {
      short sin_family;
      u_short sin_port;
      struct in_addr sin_addr;
      char sin_zero[8];
};
```

- sin_family: AF_INET; (identifica che stiamo trattando un indirizzamento di tipo IP)
- sin_port: numero porta (0-65.535);
- sin_address: indirizzo IP; (occorre inizializzare la struttura inerente all'IP, non è stringa)
- sin_zero: inutilizzato.

Macchine diverse utilizzano diversi ordinamenti di byte per memorizzare i numeri.
Ci possono essere due modi:

- ***Little-endian***: prima i byte meno significativi (es. 1.1.168.192)
- ***Big-endian***: prima i byte più significativi (es. 192.168.1.1)

Passando da una macchina big-endian ad una macchina little-endian l'interpretazione potrebbe essere sbagliata. Per questo occorre che il numero di porte e gli indirizzi siano gestiti indipendentemente dall'architettura della macchina.

Bisogna quindi distinguere l'ordinamento dei byte dell'host (che può essere little o big endian) dall'ordinamento dei byte di rete, che dovrà essere SEMPRE big-endian.
Qualunque parola inviata via rete sarà sempre posta in una codifica Big-endian. Se la macchina destinataria poi usa little-endian, bisognerà eseguire una conversione.
Sarà nostra responsabilità scrivere all'interno della struttura gli indirizzi di rete in formato big-endian.
A tal proposito esistono delle funzioni standard che sono fornite dalla socket library:

- u_long htonl(u_long x);
- u_short htons(u_short x);
- u_long ntohl(u_long x);
- u_short htohs(u_short x);

(l'acronimo sta per "network/host to host/netowork long/short)
Sulle macchine big-endian, queste routine non fanno nulla.
Sulle macchine little-endian, invertono l'ordinamento dei byte

M4 U1 L6 Select

Un programma strutturato in più processi o thread indipendenti potrebbe voler abbandonare l'approccio bloccante tipico delle funzioni della Socket Library.
Per far ciò si può:

- usare chiamate bloccanti su codice multi-thread o multiprocesso
- disattivare la funzione di blocco, per esempio usando la funzione di controllo del descrittore di file `fcntl`
- usare la chiamata *select* che consente di avere più socket in attesa e gestire sistematicamente i dati in arrivo su uno di essi

Chiamata select

Select può essere bloccante, bloccante per un certo tempo (avere un timeout alla quale scadenza la chiamata scade) o non bloccante (inizializzando una serie di socket pronti a ricevere ma passando poi ad altro).

L'input delle chiamate di select è la serie di descrittori di file (socket) sui quali si vuole stare in ascolto.
L'output sono le informazioni sullo stato, ossia il primo socket su cui ci sono delle informazioni utili.
Quindi questa chiamata identifica i socket pronti per l'uso, su cui sono disponibili dei dati.

Per controllare le operazioni da svolgere sui vari socket, select utilizza una struttura: fd_set
Questa è composta da un vettore di bit.
Se il bit *i* è impostato a [`readfds`, `writefds`, `exceptfds`], select controllerà se il descrittore di file (cioè il socket) *i-esimo* è sta arrivando dell'informazione, e in caso tornare quello (vengono quindi definiti gli eventi che il socket aspetterà).

Prima di chiamare select

- `FD_ZERO(&fdvar)`: inizializza la struttura;
- `FD_SET(i,&fdvar)`: per controllare il descrittore di file i.

Dopo aver chiamato select

- `int FD_ISSET(i, &fdvar)`: booleano, restituisce TRUE se i è "pronto"

Chiamata Select

```
int status = select(nfds, &readfds, &writefds, &exceptfds, &timeout);
```

- status (valore di ritorno): numero di oggetti pronti, altrimenti - 1;
- nfds: 1 + numero massimo di descrittori da controllare;
- readfds: elenco di descrittori da controllare perchè pronti alla lettura;
- writefds: elenco di descrittori da controllare perchè pronti alla scrittura;
- exceptfds: elenco di descrittori da controllare perché si è verificata un'eccezione;
- timeout: tempo dopo il quale select ritorna, anche se nessun socket è pronto, (può essere 0, e in questo caso non è bloccante, o un puntatore nullo, in questo caso il parametro di timeout punta a NULL).

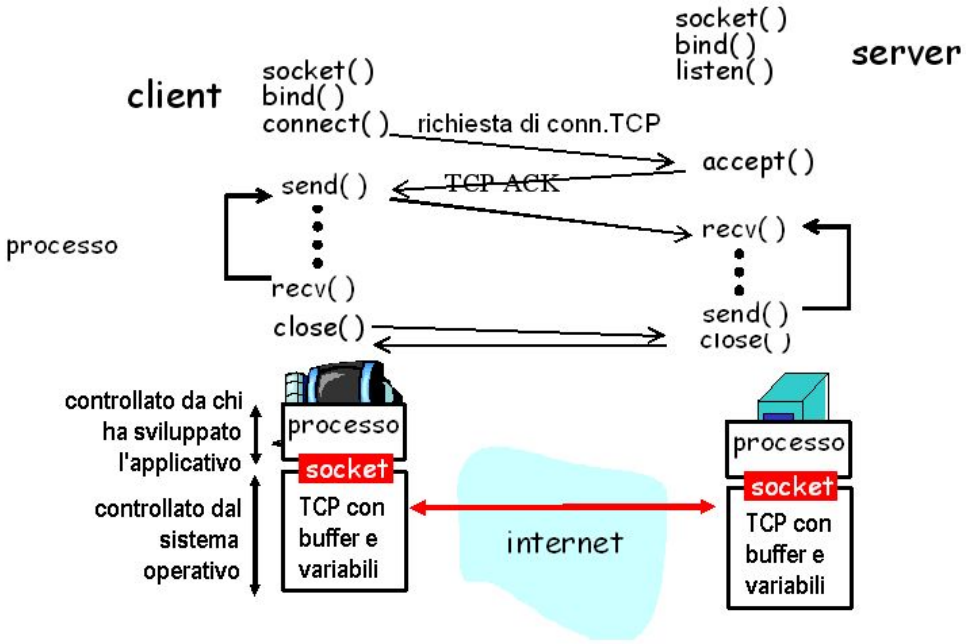
Altre funzioni utili

- `bzero(char *c, int n)`: copia *n* byte iniziali nulli (0) in c
- `gethostname(char *name, int len)`: restituisce il nome dell'host corrente
- `gethostbyadd(char *add, int len, int type)`: restituisce l'indirizzo, formattato rispetto al tipo "type" (AF_INET), dell'host specificato in "addr"
- `inet_addr(const char *cp)`: converte stringhe di caratteri decimali puntati rappresentati un IP in interi lunghi
- `inet_ntoa(const struct in_addr in)`: converte un long in notazione decimale puntata

Nota 1) in caso di una uscita rude da programma (Ctrl-C) il sistema non lascia libera immediatamente la porta.
Per ridurre il problema, si utilizza la libreria di di "signal.h", che permette di intercettare dei codici di chiusura forzata.

Nota 2) Chi sviluppa l'applicativo è in grado di fissare alcuni parametri TCP, ad esempio le dimensioni massime del buffer e dei segmenti

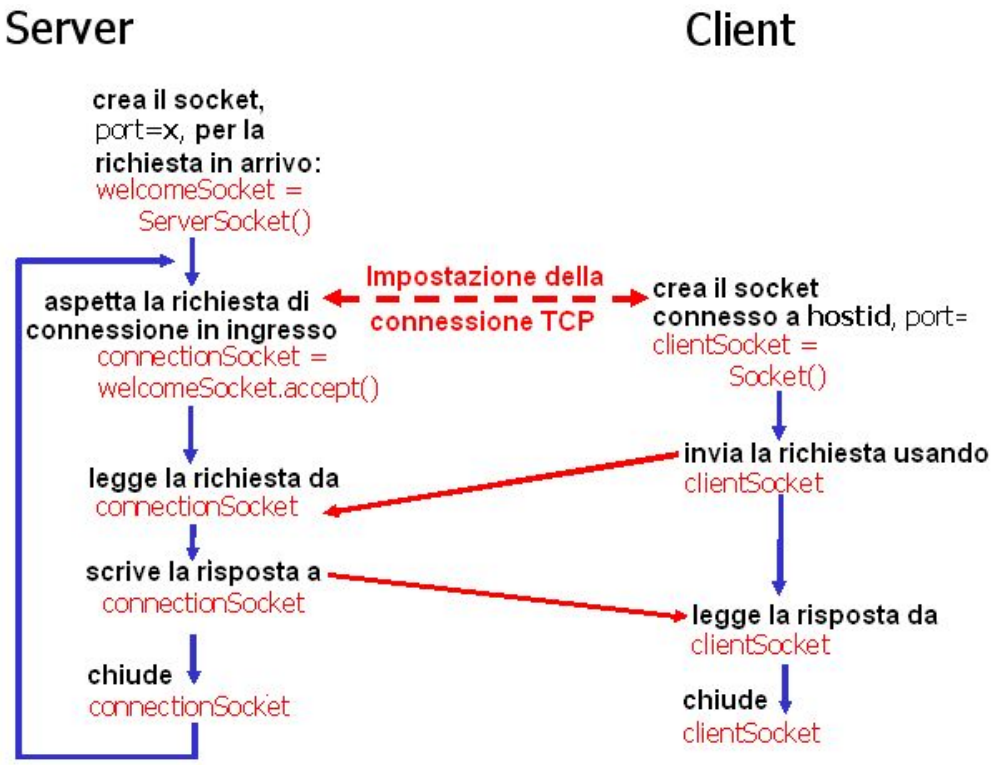
Nota 3) Pseudocodice di programmazione socket con TCP



Servizio TCP: trasferimento affidabile di flussi di byte

M4 U1 L7 Socket library in Java

Esempio di interazione dei socket TCP client/server



Nel package `java.net` sono presenti le classi per gestire i socket.

`java.net.Socket`

Implementa il socket lato client.

→ Costruttore e metodi

- `Socket(String host, int port)`: Crea un socket di flussi e lo connette al numero di porta specificato sull'host con nome
- `InputStream getInputStream()` usato per leggere dal socket creato
- `OutputStream getOutputStream()` usato per scrivere nel socket creato

- `close()`

`java.net.ServerSocket`

Implementa il socket lato server

→ Alla creazione aspetta automaticamente che arrivino richieste sulla rete

→ Esegue le operazioni in base alle richieste

→ Costruttore e metodi:

- `ServerSocket(int port)`
- `Socket Accept()`: controlla che avvenga la connessione a questo socket e la accetta (Questo metodo è bloccato finché non viene eseguita una connessione)

Esempio Client TCP

```
import java.io.*;
import java.net.*;

class TCPClient {
    public static void main(String argv[]) throws Exception
    {
        String sentence;
        String modifiedSentence;

        BufferedReader inFromUser =
            new BufferedReader(new InputStreamReader(System.in));

        Socket clientSocket = new Socket("hostname", 6789);

        DataOutputStream outToServer =
                    new DataOutputStream(clientSocket.getOutputStream());

          BufferedReader inFromServer =
            new BufferedReader(new
    InputStreamReader(clientSocket.getInputStream()));

    sentence = inFromUser.readLine();

    outToServer.writeBytes(sentence + '\n');

    modifiedSentence = inFromServer.readLine();

    System.out.println("FROM SERVER: " + modifiedSentence);

    clientSocket.close();

    }
}
```

Esempio Server TCP

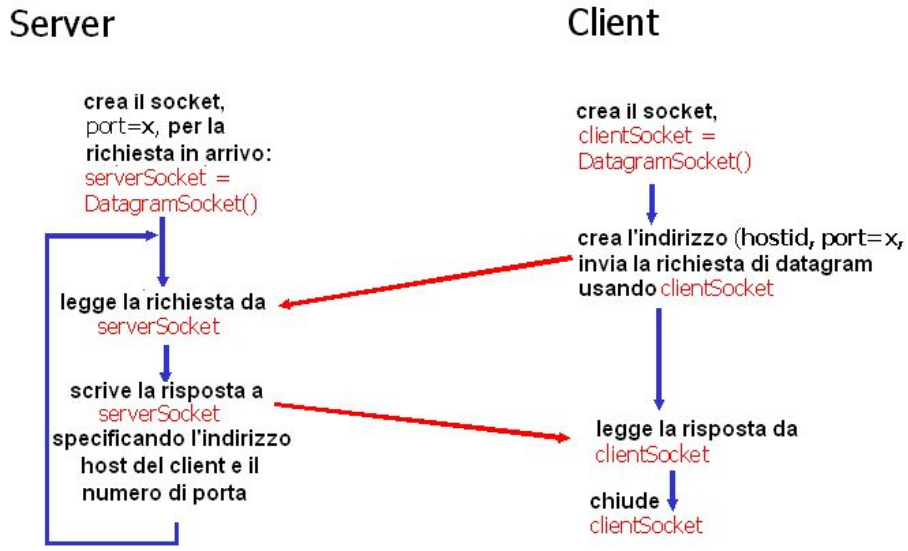
```
import java.io.*;
import java.net.*;
class TCPServer {
   public static void main(String argv[]) throws Exception
      {
        String clientSentence;
        String capitalizedSentence;

        ServerSocket welcomeSocket = new ServerSocket(6789);

        while(true) {

          Socket connectionSocket = welcomeSocket.accept();

         BufferedReader inFromClient = new BufferedReader(new
              InputStreamReader(connectionSocket.getInputStream()));

         DataOutputStream  outToClient =
            new DataOutputStream(connectionSocket.getOutputStream());

        clientSentence = inFromClient.readLine();

        capitalizedSentence = clientSentence.toUpperCase() + '\n';

        outToClient.writeBytes(capitalizedSentence);

      }
    }
}
```

UDP

- Servizio non affidabile e senza connessioni
- Non c'è una fase handshaking iniziale
- Non ha un pipe
- I dati trasmessi possono perdersi o arrivare a destinazione in un ordine diverso da quello d'invio
- Overhead minimo

Dal punto di vista della programmazione socket, UDP fa sì che non serva un socket di benvenuto e che non ci sia alcun flusso legato ai socket.
L'host mittente crea "pacchetti" allegando l'indirizzo IP di destinazione e il numero di porta a ogni batch di byte.
Il processo destinatario deve estrarre dal pacchetto ricevuto i byte di informazione.

Pseudocodice di C/S UDP



Ancora, nel package `java.net` sono presenti le classi per gestire i socket UDP:

java.net.DatagramSocket
Crea un socket per inviare e ricevere i pacchetti di datagram
Costruttore e metodi

- `DatagramSocket(int port)`: Costruisce un socket di datagram e lo lega alla porta specificata sulla macchina host locale
- `void receive( DatagramPacket p)` (eventualmente specificando hostid e porta del destinatario)
- `void send( DatagramPacket p)` (eventualmente specificando hostid e porta del mittente)
- `void close()`

Esempio socket client UDP

```
import java.io.*;
import java.net.*;

    class UDPClient {
       public static void main(String args[]) throws Exception
       {

         BufferedReader inFromUser =
           new BufferedReader(new InputStreamReader(System.in));

         DatagramSocket clientSocket = new DatagramSocket();

         InetAddress IPAddress =
    InetAddress.getByName("hostname");

         byte[] sendData = new byte[1024];
         byte[] receiveData = new byte[1024];

         String sentence = inFromUser.readLine();

         sendData = sentence.getBytes();
```

- NB InetAddress.getBytName(“nome”) è una chiamata DNS -

```
    DatagramPacket sendPacket =
          new DatagramPacket(sendData, sendData.length,
  IPAddress, 9876);

    clientSocket.send(sendPacket);

    DatagramPacket receivePacket =
          new DatagramPacket(receiveData, receiveData.length);

    clientSocket.receive(receivePacket);

    String modifiedSentence =
          new String(receivePacket.getData());

    System.out.println("FROM SERVER:" + modifiedSentence);

    clientSocket.close();

       }
}
```

Esempio socket server UDP

```
import java.io.*;
import java.net.*;

class UDPServer {
      public static void main(String args[]) throws Exception
        {

          DatagramSocket serverSocket = new
    DatagramSocket(9876);

          byte[] receiveData = new byte[1024];
          byte[] sendData  = new byte[1024];

          while(true)
            {

              DatagramPacket receivePacket =
                 new DatagramPacket(receiveData, receiveData.length);

              serverSocket.receive(receivePacket);

              String sentence = new String(receivePacket.getData());

        InetAddress IPAddress = receivePacket.getAddress();

        int port = receivePacket.getPort();

        String capitalizedSentence = sentence.toUpperCase();
         sendData = capitalizedSentence.getBytes();

        DatagramPacket sendPacket =
           new DatagramPacket(sendData, sendData.length, IPAddress, port);

         serverSocket.send(sendPacket);

      }
   }
}
```

getAddress e getPort permettono di ottenere i dati del mittente.

APPENDICE 1

COMANDI E CODICI FTP

Abort	ABOR		Interrompe trasferimento dati.
Account	ACCT	<account-information>	Informazioni account (raramente usato).
Allocate	ALLO	<decimal-integer>	Alloca spazio sufficiente per ricevere un file (raramente usato).
Append (with create)	APPE	<pathname>	Appende dati ad un file esistente.
Change to parent directory	CDUP		Va alla parent directory.
Change working directory	CWD	<pathname>	Cambia directory corrente.
Delete	DELE	<pathname>	Cancella file.
Help	HELP	<command>	Ritorna la lista dei comandi accettati dal server. Con argomento fornisce spiegazioni riguardo al comando specificato.
List	LIST	<pathname>	Lista il contenuto di una directory o le proprietà di un singolo file.
Trasfer mode	MODE	<mode-type>	Imposta la modalità di trasferimento (S=stream, B=block, C=compressed).
Make directory	MKD	<pathname>	Crea directory.
Name list	NLST	<pathname>	Ritorna il nome dei file della directory specificata.
Noop	NOOP		Non fa nulla (usato prevalentemente per prevenire disconnessioni per inattività prolungata).
Password	PASS	<password>	Specifica la password dell'utente.
Passive	PASV		Inizializza connessione dati passiva.
Data port	PORT	<host-port>	Inizializza connessione dati attiva (ossia crea la connessione utilizzata per lo scambio dati su una porta non well-known per la modalità attiva)
Print working directory	PWD		Ritorna nome della directory corrente.
Logout	QUIT		Disconnette. Se un trasferimento è ancora in corso attende che termini prima di chiudere la sessione.
Reinitialize	REIN		Effettua il log-off dell'utente loggato.
Restart	REST	<marker>	Riprende il trasferimento dall'offset indicato.
Retrieve	RETR	<pathname>	Preleva file (da server a client).
Remove directory	RMD	<pathname>	Rimuove directory.
Rename from	RNFR	<pathname>	Rinomina (sorgente).

Rename to	`RNTO`	`<pathname>`	Rinomina (destinazione).
Site parameters	`SITE`	`<command>`	Manda comando specifico per il server (non standardizzato; varia tra implementazioni).
Structure mount	`SMNT`	`<pathname>`	Monta struttura (raramente usato).
Status	`STAT`	`<pathname>`	Ritorna statistiche riguardo al server. Con argomento lista il contenuto di una directory utilizzando il canale comandi.
Store	`STOR`	`<pathname>`	Spedisce un file (da client a server).
Store unique	`STOU`	`<pathname>`	Spedisce un file (da client a server) utilizzando un nome univoco.
File structure	`STRU`	`<structure-code>`	Imposta la struttura dati (`F=file`, `R=record`, `P=page`). Praticamente inutilizzato. Il valore di default è `F`.
System	`SYST`		Ritorna tipo di sistema operativo.
Representation type	`TYPE`	`<type>`	Imposta la modalità di trasferimento (`A=ASCII`, `E=EBCDIC`, `I=Binary`, `L=Local`). Il valore di default è `A`. EBCDIC e Local sono raramente usati (esempio: unicamente su sistemi mainframe).
User Name	`USER`	`<username>`	Specifica nome utente.

Codici

1xx - Risposta preliminare positiva: Questi codici di stato indicano che un'azione è stata avviata correttamente, ma che il client rimane in attesa di un'altra risposta prima di continuare con un nuovo comando.

110 - Risposta dell'indicatore di riavvio.
120 - Servizio pronto tra nnn minuti.
125 - Connessione dati già aperta. Inizio del trasferimento in corso.
150 - Stato del file corretto. Apertura della connessione dati in corso. Per FTP vengono utilizzate due porte: la porta 21 per l'invio di comandi e la porta 20 per l'invio di dati. Il codice di stato 150 indica che dal server sta per essere aperta una nuova connessione sulla porta 20 per l'invio di dati.

2xx - Risposta di completamento positiva: Un'azione è stata completata correttamente. Il client può eseguire un nuovo comando.

200 - Comando corretto.
202 - Comando non implementato. Superfluo in questo sito.
211 - Stato del sistema o risposta della Guida di sistema.
212 - Stato della directory.
213 - Stato del file.
214 - Messaggio della Guida in linea.
215 - Tipo di sistema NAME, dove NAME è un nome di sistema ufficiale dall'elenco contenuto nel documento dei numeri assegnati.
220 - Servizio pronto per un nuovo utente.
221 - Chiusura della connessione di controllo da parte del servizio. Disconnessa se necessario.
225 - Connessione dati aperta. Nessun trasferimento in corso.

226 - Chiusura della connessione dati. Azione di file richiesta riuscita. Ad esempio, trasferimento file o interruzione file. Mediante il comando viene aperta una connessione dati sulla porta 20 per eseguire un'azione, come il trasferimento di un file. L'azione viene completata e la connessione dati viene chiusa.
227 - Inizio modalità passiva (h1,h2,h3,h4,p1,p2).
230 - Utente connesso. Continuare. Questo codice di stato viene visualizzato dopo l'invio della password corretta da parte del client. Indica che l'utente è regolarmente connesso.
250 - Azione di file richiesta corretta. Completata..
257 - Creato "PATHNAME".

3xx - Risposta intermedia positiva: Il comando è stato eseguito correttamente, tuttavia per completare l'elaborazione della richiesta è necessario che il client invii ulteriori informazioni al server.

331 - Nome utente corretto. Richiesta password. Viene visualizzato dopo l'invio del nome utente da parte del client. Lo stesso codice di stato viene visualizzato anche se il nome utente specificato corrisponde a un account valido nel sistema.
332 - Richiesto account per l'accesso.
350 - Azione di file richiesta in attesa di ulteriori informazioni.

4xx - Risposta di completamento negativa temporanea: Il comando non è stato eseguito correttamente, ma l'errore è temporaneo. Se il comando viene ripetuto dal client, potrebbe essere completato correttamente.

421 - Servizio non disponibile. Chiusura della connessione di controllo. Potrebbe trattarsi di una risposta a qualsiasi comando se il servizio deve essere chiuso.
425 - Impossibile aprire la connessione dati.
426 - Connessione chiusa. Trasferimento interrotto. Mediante il comando viene aperta una connessione dati per eseguire un'azione, ma l'azione viene annullata e la connessione dati chiusa.
450 - Azione di file richiesta non eseguita. Il file non è disponibile, ad esempio è occupato.
451 - Azione richiesta interrotta: errore locale durante l'elaborazione.
452 - Azione richiesta non eseguita. Spazio di archiviazione insufficiente nel sistema.

5xx - Risposta di completamento negativa permanente: Il comando non è stato eseguito correttamente e l'errore è permanente. Se il comando viene ripetuto dal client, verrà visualizzato lo stesso errore.

500 - Errore di sintassi. Comando sconosciuto. Può includere errori come una riga di comando troppo lunga.
501 - Errore di sintassi in parametri o argomenti.
502 - Comando non implementato.
503 - Sequenza di comandi errata.
504 - Comando non implementato per il parametro specificato.
530 - Non connesso. Questo codice di stato indica che all'utente non è consentito l'accesso poiché la combinazione di nome utente e password non è valida. Se per effettuare l'accesso viene utilizzato un account utente, è possibile che il nome utente o la password non sia stata digitata correttamente o che sia stato impostato solo l'accesso anonimo. Se si effettua l'accesso con un account anonimo, è possibile che IIS sia stato configurato per negare l'accesso anonimo.
532 - Richiesto account per l'archiviazione dei file.
550 - Azione richiesta non eseguita. Il file non è disponibile, ad esempio non è stato trovato o non è possibile accedervi. Il comando non viene eseguito in quanto il file specificato non è valido. Ad esempio, questo codice di stato viene visualizzato quando si tenta di utilizzare un verbo GET per accedere a un file

inesistente oppure un verbo PUT per memorizzare un file in una directory per la quale non si dispone dell'accesso in scrittura.
551 - Azione richiesta interrotta: tipo di pagina sconosciuto.
552 - Azione di file richiesta interrotta. Allocazione spazio di archiviazione superata per la directory o il set di dati corrente.
553 - Azione richiesta non eseguita. Nome file non consentito.
554 - Denied. Errore dovuto all'errata configurazione del server SMTP, al fine di evitare problemi con lo SPAM i server SMTP accettano email solo dal medesimo dominio del server SMTP che le ha generate.

```
ESEMPIO COMUNICAZIONE  FTP
%   ftp cs.colorado.edu
220 bruno FTP server (SunOS 4.1) ready.
USER nome_utente
331 Guest login ok, send ident as password.
PASS password
230 Messaggio di benvenuto
CWD \nuova\dir\lavoro\
250 CWD command successful
RETR README
200 PORT command successful.
150 ASCII data connection for README (128.138.242.10,3134) (2881 bytes).
226 ASCII Transfer complete.
RLST \dit\
200 PORT command successful
150 Connecting to port 62884
drwxr-xr-x    3 0          0               4096 Apr 28 11:18 .
drwxr-xr-x    3 0          0               4096 Apr 28 11:18 ..
-r--r--r--    1 0          0                 90 Apr 28 11:18 .htaccess
-rw-r--r--    1 0          2                  0 Apr 28 11:18 .override
-rw-r--r--    1 0          2                  0 Apr 28 11:18 DO NOT UPLOAD
FILES HERE
drwxr-xr-x    9 25648679   25648679         4096 Apr 28 12:03 htdocs
226-Options: -a -l
226 6 matches total
```

APPENDICE 2

TELNET

L'obiettivo principale del protocollo TELNET è quello di disporre di un interfacciamento standard dei dispositivi a terminale e dei processi terminal-oriented attraverso la rete.
TELNET definisce una rappresentazione standard per cinque funzioni. Queste funzioni hanno un significato ben specifico, e all'infuori della funzione IP sono tutte opzionali: se una funzione non viene fornita, allora il suo significato è quello di un'operazione nulla.

Go Ahead (Cod. 249)
Quando un processo ha finito di spedire i dati e nelle sue code in entrata non ce ne sono altri, viene spedito il comando GA che trasferisce il controllo dall'altra parte della rete.
Questo comando serve come "gettone" per il controllo del canale trasmissivo (il quale viene visto come half-duplex) per i terminali che trattano in maniera diversa i segnali di tipo end-of-line (un esempio e' dato dall'IBM 2741).

Interrupt Process (IP) (Cod. 244)
Molti sistemi forniscono una funzione che sospende, interrompe, abortisce o termina l'operazione di un processo. Questa funzione torna utile nel caso in cui un processo sia entrato in un ciclo infinito o nel caso in cui un processo non voluto sia stato inavvertitamente messo in funzione. IP è la rappresentazione standard di questa funzione.
Poiché questa funzione può venire usata da protocolli che si basano su TELNET, IP deve essere implementata.

Abort Output (AO) (Cod. 245)
Molti sistemi forniscono una funzione per bloccare la generazione dell'output permettendo al processo di giungere al termine senza spedire l'output al terminale. In più, questa funzione cancella tutta la coda di output prodotta ma non ancora stampata (residente nel buffer). AO è la rappresentazione standard di questa funzione.
Per esempio, una shell accetta un comando dall'utente, spedisce come risposta una serie di stringhe di caratteri al terminale ed infine spedisce un "prompt" al terminale per segnalare la disponibilità ad accettare un nuovo comando. Se l'AO fosse stato ricevuto durante la trasmissione del testo, la spedizione delle stringhe sarebbe terminata, ma il prompt verrebbe trasmesso ugualmente.

Are You There (AYT) (Cod. 246)
Molti sistemi forniscono all'utente con un messaggio visibile che il sistema funziona e sta processando. Questa funzione viene invocata dall'utente quando il sistema rimane "silenzioso" per un lungo periodo, di solito per colpa di una grossa computazione o per l'eccessivo carico. AYT è la rappresentazione standard di questa funzione.

Erase Character (EC) (Cod. 247)
Molti sistemi forniscono una funzione per cancellare l'ultimo carattere digitato dall'utente (il backspace in questo caso viene anch'esso visto come un carattere). Questa funzione è tipicamente usata per correggere eventuali errori di battitura. EC è la rappresentazione standard di questa funzione.

Erase Line (EL) (Cod. 248)
Molti sistemi forniscono una funzione per cancellare tutti i dati in una "linea" di input. Questa funzione è tipicamente usata per editare input da tastiera. EL è la rappresentazione standard di questa funzione

Opzioni negoziabili

Il principio della negoziazione delle opzioni viene attuata per il fatto che molti hosts forniscono dei servizi aggiuntivi rispetto a quelli forniti dal terminale NVT standard, e per il fatto che molti client dispongono di

terminali sofisticati, e vorrebbero avere una visione dello schermo più "elegante" rispetto a quella spartana dell'NVT. Il protocollo TELNET definisce il concetto di opzioni aggiuntive per permettere ai due partner di una connessione TELNET di modificare o elaborare un insieme di convenzioni riguardanti la connessione. Le opzioni possono comprendere il cambiamento del set di caratteri, la modalità di output, ecc.

La semplice strategia per utilizzare le opzioni è la seguente: un partner (o entrambi) indica all'altro che intende usare una certa opzione; l'altro partner può sia accettare che rifiutare la richiesta.

WILL (Cod. 251) e WON'T (Cod. 252)

Il comando WILL XXX indica che un partner ***desidera*** usare l'opzione XXX; DO XXX e DON'T XXX saranno gli acknowledgments positivo e negativo. Simmetricamente, il comando WON'T XXX indica che un partner non desidera più usare l'opzione XXX.

DO (Cod. 253) e DON'T (Cod. 254)

Il comando DO XXX indica che un partner ***richiede*** all'altro partner di usare l'opzione XXX; WILL XXX e WON'T XXX saranno gli acknowledgments positivo e negativo.
Simmetricamente, il comando DON'T XXX indica che un partner richiede all'altro partner di togliere l'opzione XXX.

SB (Cod 250) e SE (Cod 240)

Subnegoziazione (per le opzioni che richiedono parametri). Si effettua una negoziazione spedendo un DO o WILL per l'opzione da negoziare. Se l'altro host da un acknowledgement, si procederà con
`SB XXX <parametri> SE`
Questo attiva la subnegoziazione per parametri. Se il destinatario rifiuta, bisognerà effettuare la negozioazione dall'inizio. SE è il carattere di fine parametri

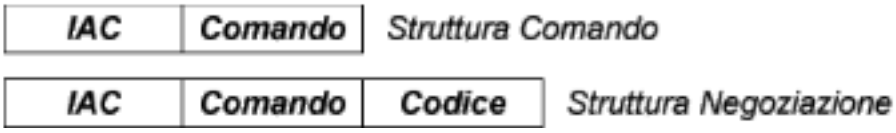
nr.	Acronimo	RFC	Dichiarazione	Significato
0	TRANSMIT-BINARY	856	Binary transmission	Trasmissione binaria dei dati senza gestione dei CR-LF
1	ECHO	857	Echo	Gestione degli "echo" attraverso la rete - usato assieme a SUPPRESS-GO-AHEAD per il char at time
2			Reconnection	(non documentata)
3	SUPPRESS-GO-AHEAD	858	Suppress go ahead	Disabilita l'uso del comando GA - usato assieme a ECHO per il char at time
4			Approximate message size negotiation	(non documentata)
5	STATUS	859	Status	Trasmette lo stato locale delle opzioni al partner
6	TIMING-MARK	860	Timing mark	Assicura che i dati trasmessi siano stati completamente

				processati, stampati o cancellati
7	RCTE	726	Remote controlled transmission and echoing	Gestione della trasmissione degli "echo" per reti lente
8			Output line width	(non documentata)
9			Output page size	(non documentata)
10	NAOCRD	652	Output carriage-return disposition	Gestione dei Carriage Return
11	NAOHTS	653	Output horizontal tab stops	Gestione dei tabstop orizzontali
12	NAOHTD	654	Output horizontal tab disposition	Gestione dei tabulatori orizzontali
13	NAOFFD	655	Output formfeed disposition	Gestione dei Form Feed
14	NAOVTS	656	Output vertical tabstops	Gestione dei tabstops verticali
15	NAOVTD	657	Output vertical tab disposition	Gestione dei tabulatori verticali
16	NAOLFD	658	Output linefeed disposition	Gestione dei Line Feed
17	EXTEND-ASCII	698	Extended ASCII	Gestione dei caratteri nazionali non-ASCII
18	LOGOUT	727	Logout	Chiusura della connessione TELNET
19	BM	735	Byte macro	Abbreviazioni di stringhe ad un byte
20	DET	1043	Data entry terminal	Terminale full screen (DODIIS)
21	SUPDUP	736	SUPDUP	Emulazione del protocollo SUPDUP
22	SUPDUP-OUTPUT	749	SUPDUP output	Emulazione del protocollo SUPDUP
23	SEND-LOCATION	779	Send location	Spedisce una stringa contenente la locazione fisica dell'utente
24	TERMINAL-TYPE	1091	Terminal type	Tipo di terminale (vt100, vt220

				etc.)
25	END-OF-RECORD	885	End of record	Segnala la fine di un'unità di dati multibyte
26	TUID	927	TACACS user identification	Autenticazione di utenti TAC
27	OUTMRK	933	Output marking	Permette ad un server di spedire ad un client un banner
28	TTYLOC	946	Terminal location number	Spedisce il '32-bit host address' e il '32-bit local terminal address' del terminale
29	3270-REGIME	1041	3270 Regime	Simulazione del terminale 3270
30	X.3-PAD	1053	X.3 PAD	Simulazione delle funzioni di X.3 PAD per X.25
31	NAWS	1073	Window size	Dimensioni della finestra di output (nr. di righe e colonne)
32	TERMINAL-SPEED	1079	Terminal speed	Velocità del terminale in trasmissione e ricezione (baud)
33	TOGGLE-FLOW-CONTROL	1372	Remote flow control	Gestione flusso di controllo
34	LINEMODE	1184	Linemode	Comandi di spostamento del cursore ed editing di caratteri
35	X-DISPLAY-LOCATION	1096	X display location	Gestione finestre multiple in X-Window
36	ENVIRON	1408, 1571	Environment communications	Scambio di variabili d'ambiente
37	AUTHENTICATION	1416	Authentication	Autenticazione accessi (Kerberos, SPX, RSA, Loki)
38			Encryption	(non documentata)
39	NEW-ENVIRON	1572	Environment	Scambio di variabili d'ambiente
40	TN3270E	1647	TN 3270 Enhancements	Nuova opzione per il terminale 3270

		1205	5250 telnet interface	Interfaccia per il terminale a "maschera" IBM 5250
255	EXOPL	861	Extended options list	Gestione delle opzioni superiori a 255
256	RANDOMLY-LOSE	748	Randomly-lose	Disabilita i crash di sistema, la perdita di dati, il malfunzionamento dei programmi (*)
257	SUBLIMINAL-MESSAGE	1097	Subliminal message	Spedisce messaggi subliminali ad utenti sprovveduti

Tutti i comandi TELNET consistono di una sequenza di almeno due byte: il carattere di escape Interpret As Command (IAC) seguito dal codice del comando.
I comandi riguardanti la negoziazione delle opzioni sono una sequenza di tre byte; il terzo byte sarà il codice dell'opzione referenziata.



Ad esempio, per spedire il comando IP, TELNET manda la sequenza di caratteri 255 (IAC) e 244 (IP).
Per spedire il comando DO TRANSMIT-BINARY, si manda la sequenza di caratteri 255 (IAC), 253 (DO) e 0 (opzione TRANSMIT-BINARY)

APPENDICE 3

COMANDI POP3 E SMTP

Pop3 usa la porta 110.

Comando	**Spiegazione**
USER	Your user name for this mail server
PASS	Your password.
QUIT	End your session.
STAT	Number and total size of all messages
LIST	Message# and size of message
RETR message#	Retrieve selected message
DELE message#	Delete selected message
NOOP	No-op. Keeps you connection open.
RSET	Reset the mailbox. Undelete deleted messages.

Esempio comunicazione POP3

```
S:+OK <22593.1129980067@example.com>
C:USER pippo
S:+OK
C:PASS pluto
S:+OK
C:LIST
S:+OK
```

```
  1 817
  2 124
  .
C:RETR 1
S:+OK
  Return-Path: <pippo@example.org>
  Delivered-To: pippo@example.org
  Date: Sat, 22 Oct 2005 13:24:54 +0200
  From: Mario Rossi <mario@rossi.org>
  Subject: xxxx
  Content-Type: text/plain; charset=ISO-8859-1

  testo messaggio
  .
C:DELE 1
S:+OK
C:QUIT
S:+OK
```

SMTP usa la porta 25

Comando	**Spiegazione**
ARTN	Authenticated TURN
AUTH	Authentication
BDAT	Binary data
BURL	Remote content
DATA	The actual email message to be sent This command is terminated with a line that contains only a \r\n
EHLO	Extended HELO
ETRN	Extended turn
EXPN	Expand
HELO	Identify yourself to the SMTP server.
HELP	Show available commands
MAIL	Send mail from email account "MAIL FROM: me@mydomain.com"
NOOP	No-op. Keeps you connection open.
ONEX	One message transaction only
QUIT	End session
RCPT	Send email to recipient "RCPT TO: you@yourdomain.com"
RSET	Reset
SAML	Send and mail
SEND	Send
SOML	Send or mail
STARTTLS	
SUBMITTER	SMTP responsible submitter
TURN	Turn
VERB	Verbose
VRFY	Verify

Esempio nel capitolo SMTP