PROTOCOLLI DEL LIVELLO APPLICATIVO

Mattia Dutto ~ February 2021

HTTP

HTTP

~ Hypertext Transfer Protocol ~

Definito dal RFC 1945 e in seguito dal 2616.

HTTP si basa sul protocolo **TCP**.

Il protocollo HTTP viene definito senza memoria di stato. Quando il client interroga il server lui genera la richiesta ma lui non memorizza le richieste del client.

Due tipologie di connessione a livello di trasporto:

- **Connessione persistente**
- **Connessione non persistente**

A livello applicativo ritroviamo le porte, HTTP utilizza la porta **80**

HTTP

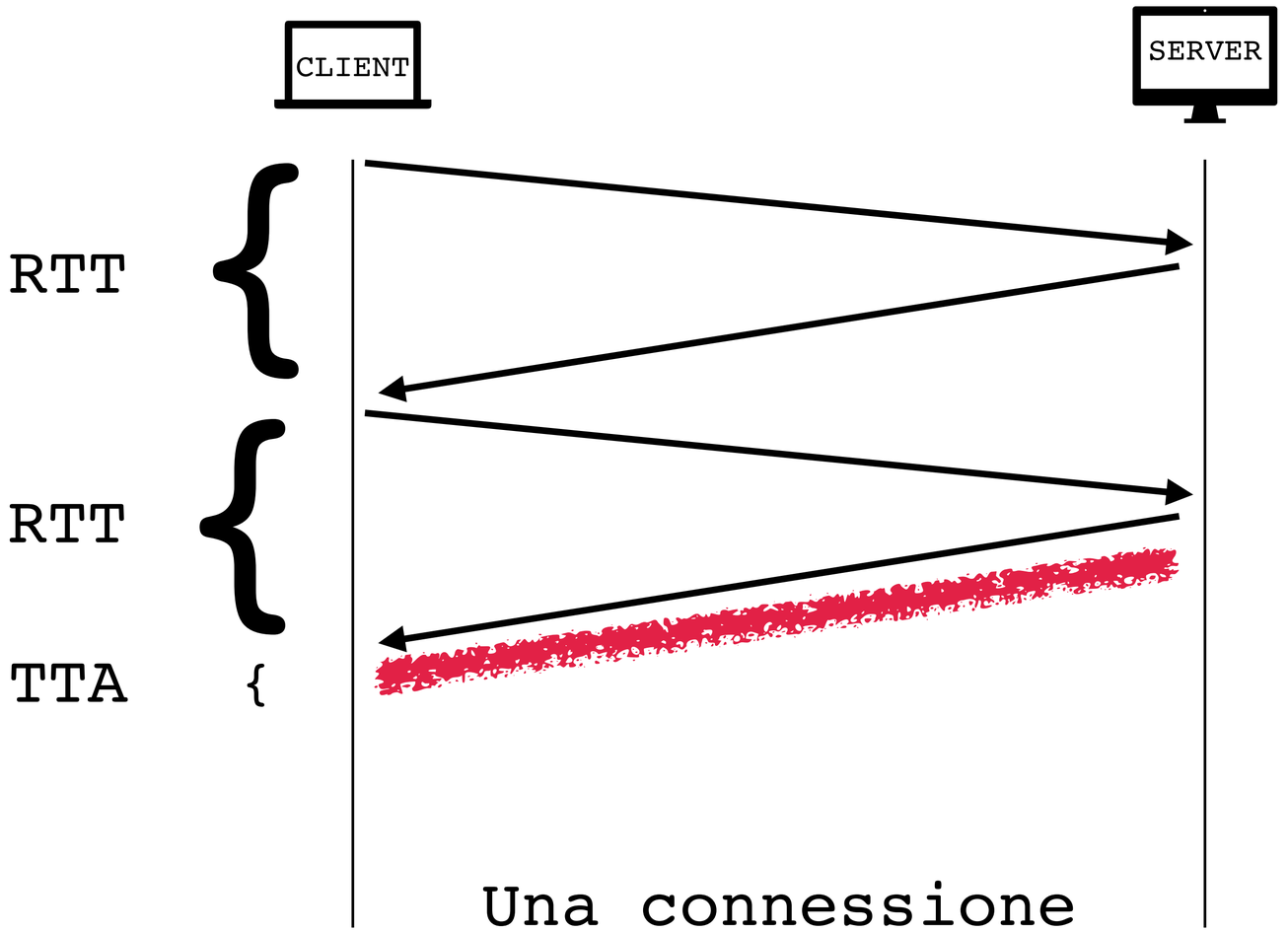
~ Connessione NON persistente ~

1. Ipotizziamo di fare una richiesta di una pagina HTML con 10 oggetti JPEG.
2. Tramite il socket il processo client invia la richiesta al server HTTP. Il protocollo TCP instaura una connessione.
3. Il server riceve il messaggio
4. Il server spedisce la risposta e interrompe la connessione
5. Il client riceve il messaggio e la connessione decade.
6. Caricata la pagina HTML, si ripete la procedura per tutti gli altri 10 oggetti.

Il problema del ritardo rimane comunque.

HTTP

~ Connessione NON persistente ~



HTTP

~ Connessione persistente ~

Una connessione persistente persiste nel tempo.

Il server non da il segnale di chiusura.

Se dall'ultima connessione passa un certo tempo e in questo tempo non avvengono richieste la connessione verrà chiusa.

Io posso decidere quanto è il massimo tempo di attesa.

~ Alcune informazioni ~

HTTP utilizza il TCP più la persistenza.

TCP: disponibilità della rete e del destinatario

orientato alla connessione

è affidabile

Persistenza: connessione stabile

HTTP

~ Definizioni ~

- **Throughput**: velocità di trasferimento del file bit/s. Il punto con la minor velocità andrà a definir il valore. Se invece ho un valor minimo richiesto dovrò garantire almeno quello in ogni mio collegamento. Se non l'ho definito il throughput può essere dinamico.
- **Timing**: tempo di trasmissione tra due punti (nodo) della rete.

Messaggi

Abbiamo due differenti messaggi inviati da HTTP:

- Messaggio di richiesta
- Messaggio di risposta

HTTP

~ Messaggio di richiesta ~

```
GET   /registro/homePage.html   HTPP/1.1

Host: www.itiscuneo.gov.it

Connection: close

User-Agent: Mozilla 5.0

Accept-Language: IT
```

HTTP

~ Messaggio di richiesta ~

```
GET /registro/homePage.html HTTP/1.1
```

Riga di richiesta:

- Campo del metodo: GET serve per effettuare una richiesta al server.
- Campo del URL: mi dice dov'è allocata la risorsa che richiedo
- Campo della versione: si intende la versione del protocollo HTTP.

HTTP

~ Messaggio di richiesta ~

Righe di intestazione: ti spiego il tipo e come eseguo la richiesta.

- HOST: dove si trova l'oggetto richiesto.
- CONNECTION: qui è il punto in cui vado a definire la persistenza della connessione:
  - CLOSE -> Connessione non persistente. Il server da l'impulso per chiudere.
  - OPEN -> Connessione persistente.
- USER-AGENT: fa riferimento al browser che utilizziamo
- ACCEPT-LANGUAGE: Mi restituisce la lingua della pagina.

HTTP

~ Messaggio di richiesta ~

POSSIBILI METODI:

- **GET**: il più utilizzato serve per una richiesta di un oggetto al server generica.
- **HEAD**: è una metodica che si utilizza per fare un controllo o debugging. Viene eseguita da un programmatore.
- **POST**: serve per andare a modificare / sovrascrivere / aggiornare una risorsa già presente.
- **PUT**: vado ad aggiungere una risorsa al mio server che prima non era presente.
- **DELETE**: vado ad eliminare / cancellare una risorsa dal server.

HTTP

~ Messaggio di risposta ~

1 HTTP/1.1 200 OK

2
```
Connection: close
Date: Thu, 18 Feb 2021 11:54:10 GMT
Server: Apache/2.2.3
Last-Modified: Thu, 18 Jan 2021 15:39:02 GMT
Content-Length: 1802
Content-Type: Text / HTML
```

3 ...

HTTP

~ Messaggio di risposta ~

Riga di stato:

HTTP/1.1 200 OK

1) **Campo della versione**: corrispettivo di quello del messaggio di richiesta.
2) **Codice di stato**: mi dice com'è andata a finire la mia richiesta.
3) **Campo del messaggio di stato**: mi da ulteriori informazioni sullo stato della richiesta.

HTTP

~ Messaggio di risposta ~

6 Righe di intestazione

1) Mi dice se la connessione è persistente o meno.
2) Specifica la data e l'ora in cui è andato a interpellare la risorsa ed è stata presa e messa nel pacchetto.
3) Indica il tipo di Web-Server che sono andato ad interpellare.
4) Per ultima modifica si fa riferimento solamente a una modifica strutturale della mia risorsa.
5) Quanto è grosso il contenuto della mia risposta in Byte.
6) Mi dice che tipo di oggetto è la mia risposta.

Dopo le righe di intestazione ritroviamo il dato.

HTTP

~ Messaggio di risposta ~

Codici di stato:

- 200 OK: Tutto è andato bene
- 301 MOVED PERMANENTLY: Non ricevo l'oggetto. Il server ha capito la mia richiesta ma non ha trovato l'oggetto. Aggiunge all'intestazione del messaggio una 7° riga con il nuovo URL:

  LOCATION: NEW_URL
- 400 BAD REQUEST: Non ha capito niente di quello che gli ho scritto.
- 404 NOT FOUND: Oggetto richiesto non trovato.
- 501 HTTP NOT SUPPORTED: il server non support la mia versione di HTTP.

HTTP

~ PROBLEMI ~

- Il nostro HTTP continua a non avere una memoria delle comunicazioni.

- Stiamo girando tutto il carico al server

COOKIE

~ Cosa sono? ~

Definiti dall'RFC 6265.

Sono astrazioni informatiche per sopperire al fatto che HTTP non ha una memoria di stato.

Sono nati per facilita la navigazione da parte dell'utente finale.

I cookie sono informazioni relative all'identità dell'utente. Non sono affatto facili, come ben sapete si portano dietro problemi legati alla privacy.

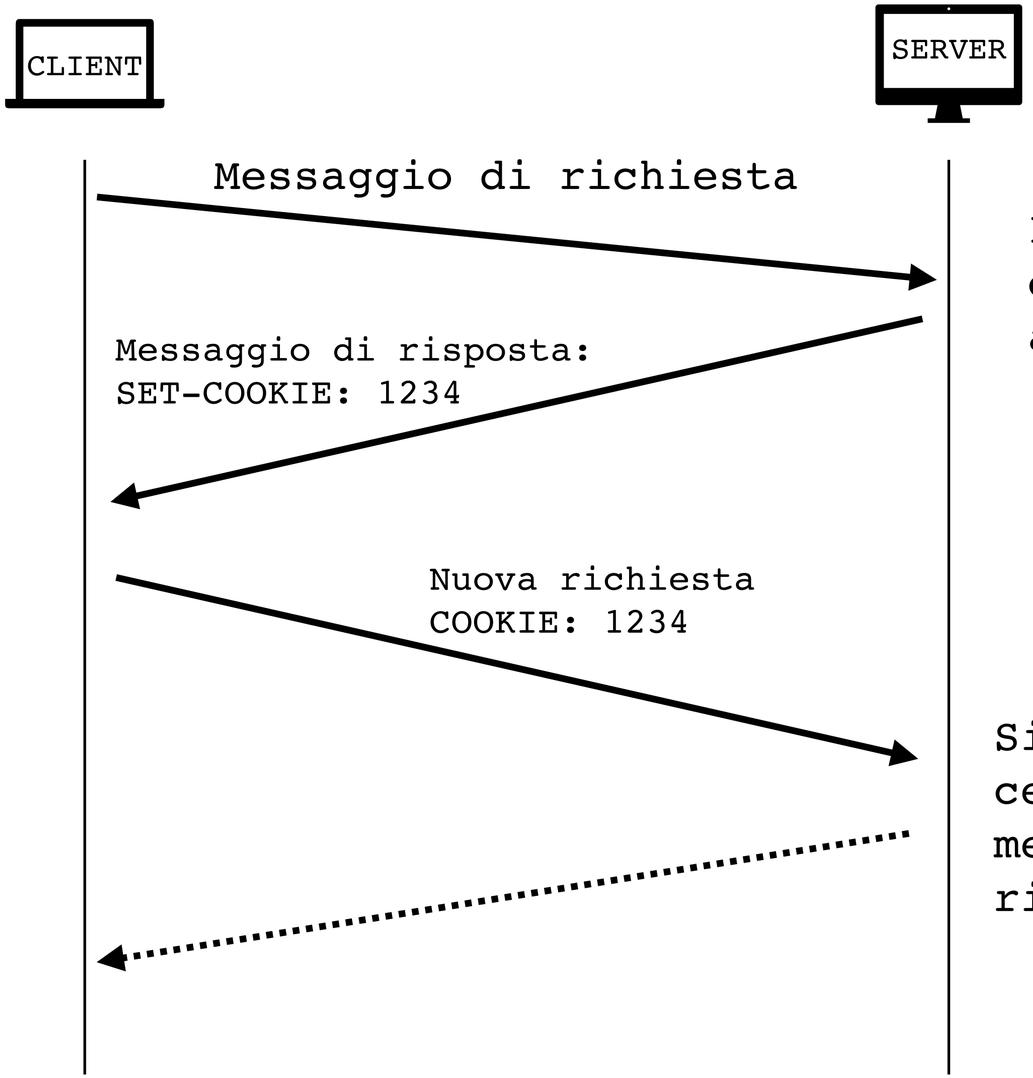
Possiamo parlare di cookie quando all'utente (client) viene assegnato un codice.

I cookie tengono traccia delle richieste effettuate da uno stesso utente lato server.

COOKIE

~ Funzionamento ~

Immaginiamo un client che effettua una richiesta a un server per la prima volta.



Il server si chiede se il client si è già collegato ad esso o meno.

Richiesta successive:

Si guarda cosa aveva già cercato l'utente e si memorizza la nuova richiesta.

WEB CACHING

~ Lato server ~

I proxy server sono degli intermediari tra il client e il server originale.

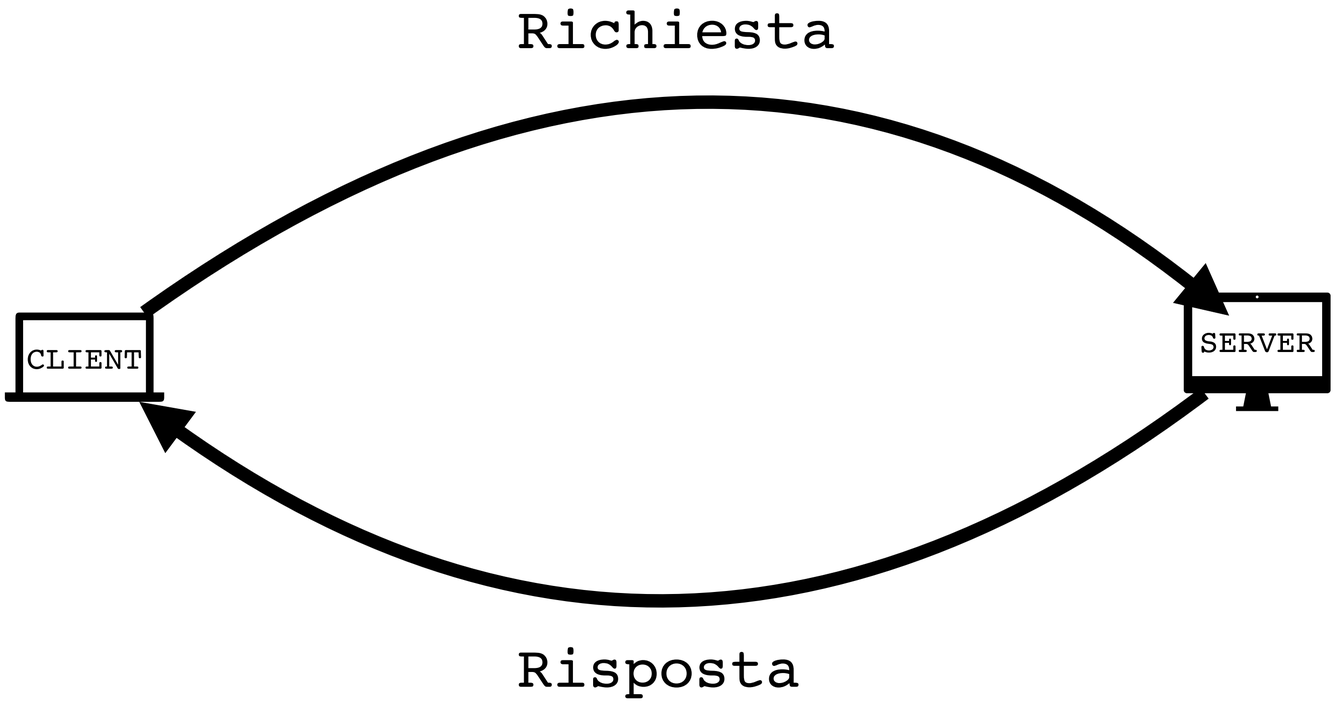
Questi server sono dotati di una memoria volatile e possono rispondere direttamente al client.

Si utilizzano per diminuire il carico di richieste sul server e quindi permettere la gestione di più richieste.

Un proxy server è una copia parziale o totale del server originario.

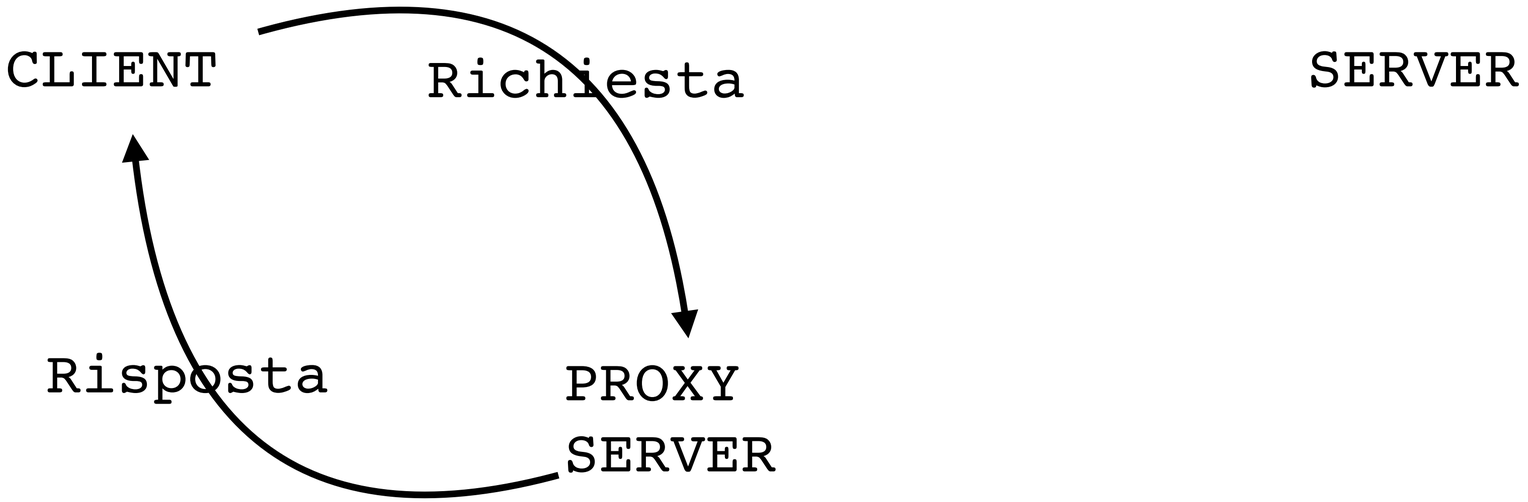
WEB CACHING

~ Caso client - server ~



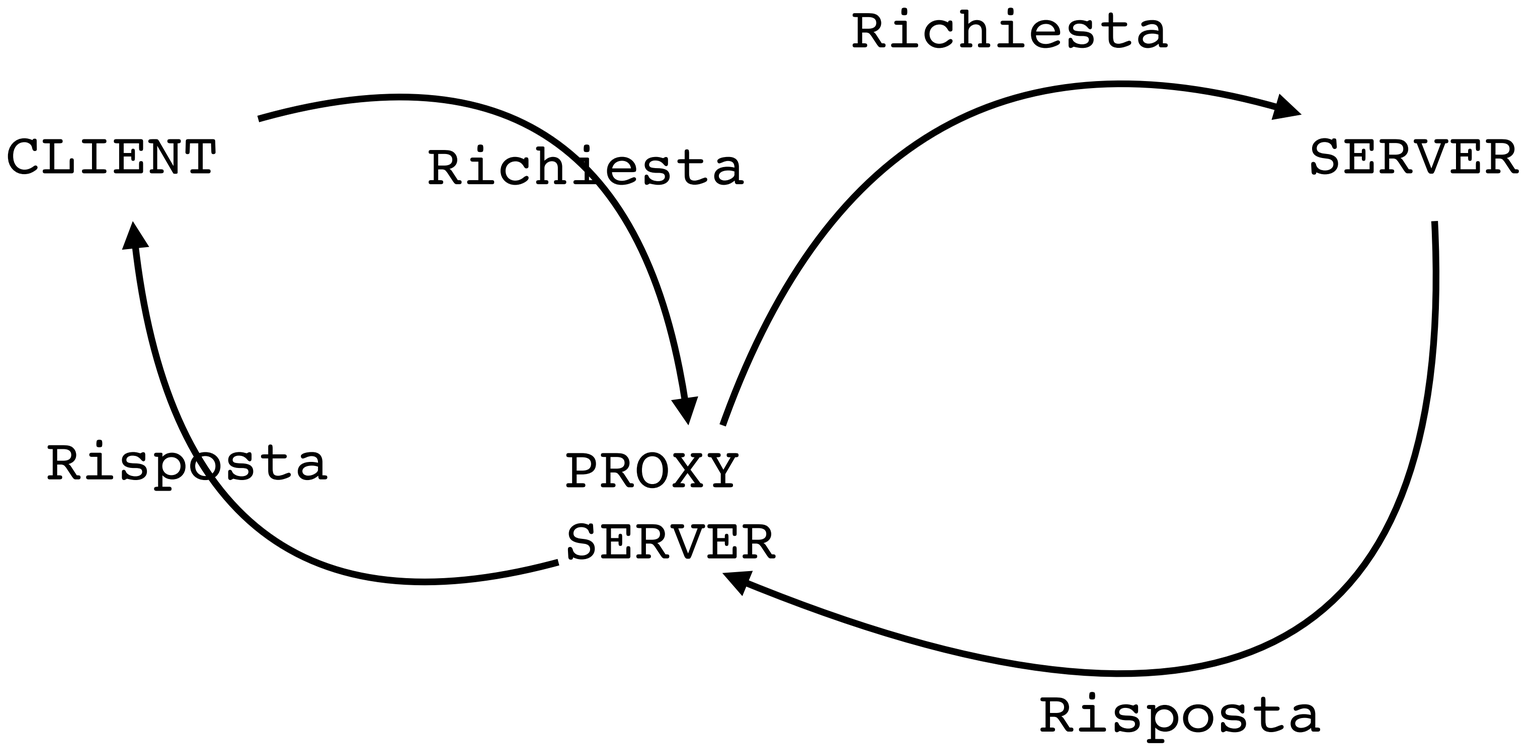
WEB CACHING

~ Caso A con PROXY SERVER ~



WEB CACHING

~ Caso B con PROXY SERVER ~



WEB CACHING

~ La risorsa evolve nel tempo ~

Può capitare che una risorsa cambi il suo contenuto nel tempo e quindi il valore presente nel proxy-server non è l'ultimo.

Il metodo GET CONDIZIONALE è un metodo riservato da HTTP per richieste da parte del proxy-server al server. Chiede al server se ha modificato l'oggetto o meno.

Due possibili risposte:

- Invia pure che non è variato.
- Ti invio il nuovo oggetto.

WEB CACHING

~ Messaggio di richiesta ~

Messaggio di richiesta dal proxy server verso il server:

GET /registro/homePage.html HTTP/1.1

Host: www.itiscuneo.gov.it

If-Modified-Since: WED 30 NOV 2016 09:20:20 GMT

CODICI DI STATO:

Se non è stato modificato: 304 NOT MODIFIED

Se fosse stato modificato e ha l'oggetto: 200 OK + OGGETTO

Se non è più presente: 404 NOT FOUND

FTP

FTP

~ File Transfer Protocol ~

FTP si appoggia a TCP, ha avuto un sacco di modifiche noi faremo riferimento all'RFC 959.

Come metodo di lavoro assomiglia ad HTTP.

A differenza di HTTP FTP lavora con due connessioni:

1) Per aprire il dialogo con il server

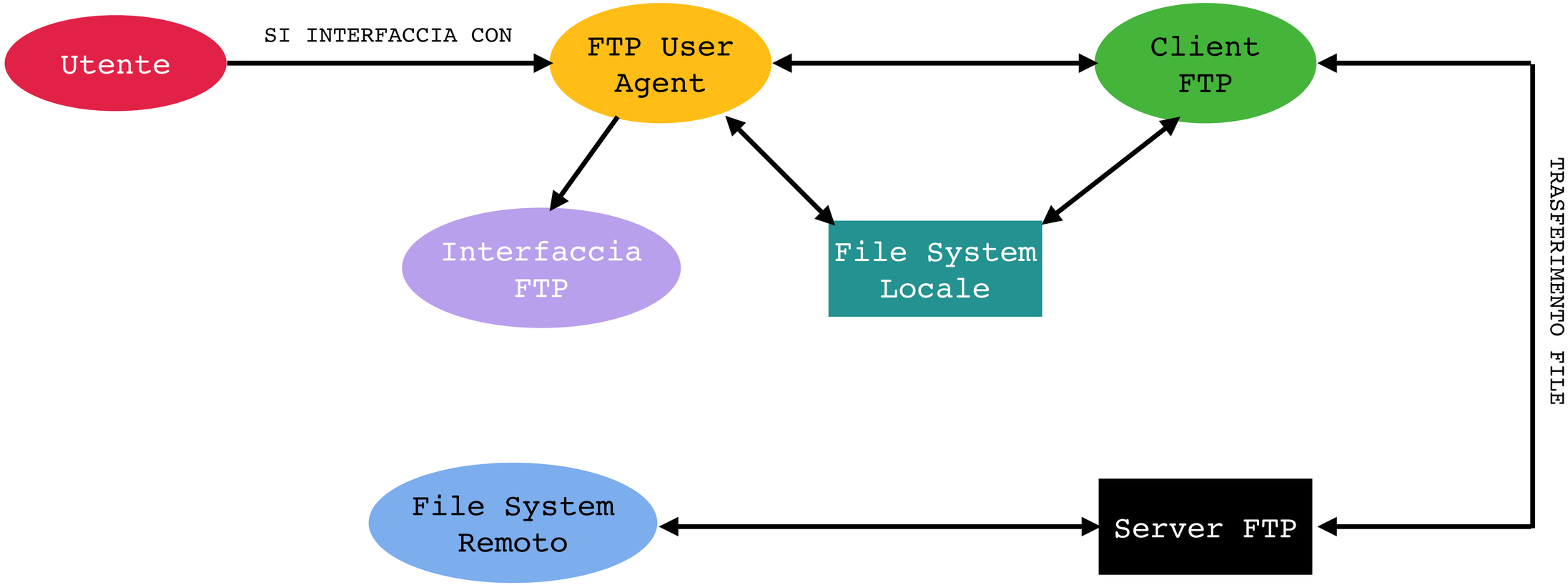
2) Per ricevere la risposta dal server

Un server FTP non è adattabile ad HTTP.

Si può sia inviare sia ricevere file dal server.

FTP

~ Funzionamento ~



FTP

~ Connessioni ~

Ha due connessione:

- **Connessione di controllo** vengono inviate al server le seguenti informazioni:
  - Nome utente
  - Password
  - “Modalità di trasferimento”
- **Connessione dati**

La connessione di controllo usa la porta **21** mentre quella dei dati la **20**.

FTP

~ Connessioni ~

La connessione di controllo è una connessione persistente.

La connessione dati non è persistente in quanto sarebbe troppo onerosa come richiesta di elaborazione.

Per ogni connessione dati si trasferisce un solo file.A fine trasferimento il server manda l’impulso per chiudere. Quando arriva al client il client la chiude.

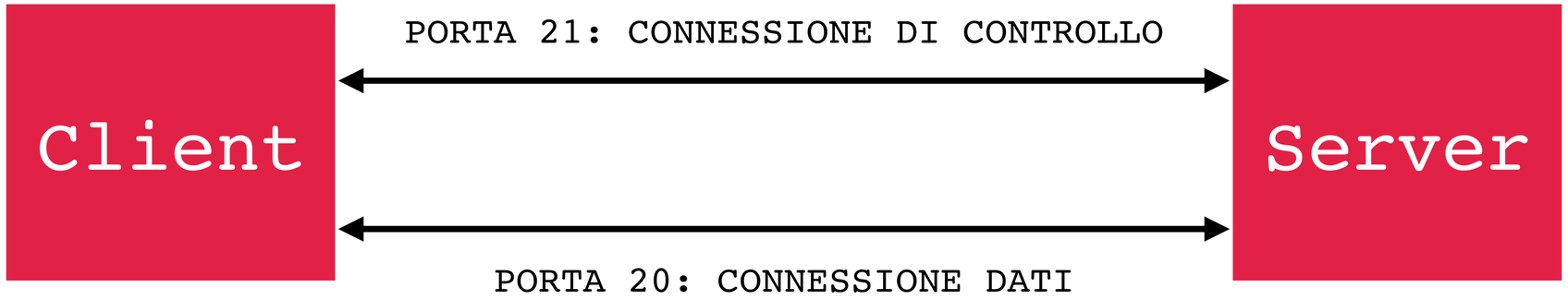
Per una sessione utente posso avere molteplici connessioni dati.

FTP ha una connessione di controllo fuori banda.

Mentre HTTP si dice che usa una connessione in banda.

FTP

~ Connessioni ~



FTP

~ Sintassi comunicazione ~

Inizialmente vi è uno scambio 1 a 1.

1. **USER, username**

Invio un comando per volta. Se lo username è accettato il client può procedere con il seguente comando:

2. **PASS, password**

Se la password viene accettata. Questi due passaggi sono da effettuare a ogni comunicazione.

FTP

~ Comandi ~

Dopo USER e PASS possiamo procedere con i seguenti comandi:

- **LIST** -> Richiesta della lista dei file contenuti nella directory remota. “Nascosta” vi è la richiesta di forzare l’apertura della connessione dati e su essa mi verrà restituita la lista dei file disponibile sul server.

- **RETR, filename** -> dove filename è il nome del file, serve per recuperare un file dalla directory remota. Mi verrà inviato il file se presente.

- **STOR, filename** -> permette di condividere il file con la directory remota.

FTP

~ Risposte ~

Possibili risposte lato SERVER:

- **331 User name okay, need password.** -> ci dice che l'utente è stato riconosciuto e possiamo procedere con l'invio della password.
- **125 Data connection already open; transfer starting.** -> possiamo procedere con l'invio del file.
- **425 Can't open data connection.**
- **452 Requested action not taken.**
  Insufficient storage space in system.

Quando ci dice che non riconosce l'username è perchè non è presente nella sua memoria di stato.

SMTP

SMTP

~ Simple Mail Transfer Protocol ~

Quando parliamo di posta elettronica facciamo riferimento a tre componenti:

- USER-AGENT
- MAIL SERVER
- SMTP

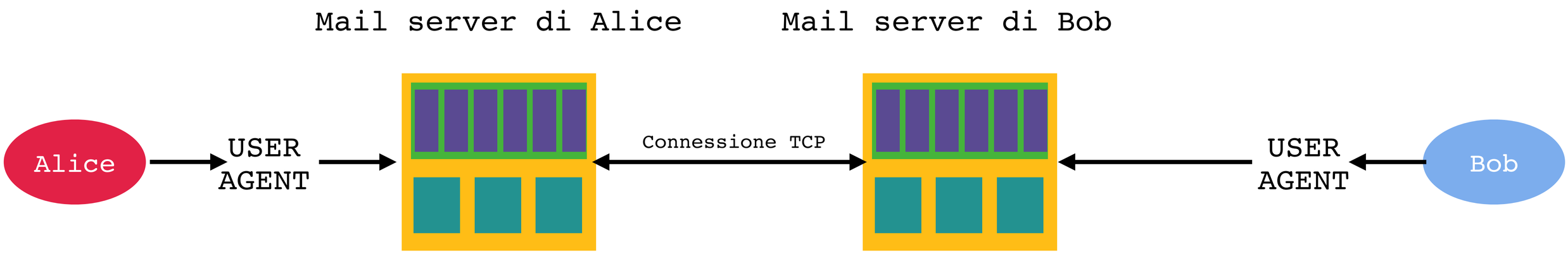
SMTP è standardizzato dal RFC 5321.

La posta elettronica viene detta asincrona.

Il mail server invia il messaggio quando ha la possibilità di instaurare una connessione TCP se non riesce mette il messaggio in coda e aspetta fino a quando non riesce ad aprire la connessione.

SMTP

~ Infografica ~



SMTP

~ Caratteristiche ~

SMTP ha due lati:

- Lato client: invia

- Lato server: riceve

SMTP lavora su una connessione TCP persistente, la connessione rimane aperta finche l’utente non ha finito di inviare tutti i messaggi.

SMTP utilizza la porta 25.

SMTP non ci permette di leggere ed organizzare la posta sul server per farlo dobbiamo appoggiarci a POP3 o IMAP: che sono i protocolli che permettono l’accesso alla posta.

SMTP

~ Invio del messaggio ~

Quando voglio inviare il messaggio:

- USER-AGENT

- Quando il messaggio è completo di indirizzo mail destinatario ed oggetto il messaggio arriva nella coda del mail server.

- Handshaking.: si stabiliscono le regole di invio.

- Si crea la connessione e viene inviato il messaggio in pacchetti.

- SMTP(S) se è acceso riceve il messaggio e lo rende leggibile.

- Il messaggio di QUIT chiude la connessione.

SMTP

~ Comunicazione tra client e server ~

BINARIO DI COLLEGAMENTO

S: 220 macchina.com -> il server è pronto a ricevere

C: HELO utente.it -> il client si presenta

S: 250 Hello utente.it, “Pleased to meet you”

CONTROLLO

C: MAIL FROM <alice@utente.it>

S: 250 alice@utente.it … SENDER OK

C: RCPT TO <bob@macchina.com>

S: 250 bob@macchina.com … RECIPIENT OK

SMTP

~ Comunicazione tra client e server ~

C: DATA

S: 354 ENTER MAIL, END WITH “.” ON A LINE BY ITSELF

MESSAGGIO

C: Ti piace la scuola?

C: Quale è la tua materia preferita?

C: .

CHIUSURA

S: 250 MESSAGE ACCEPT FOR DELIVERY

C: QUIT

S: 221 CONNECTION CLOSING

POP3

POP3

~ Protocollo di accesso alla posta ~

POP3 viene definita dal RFC 1939.

POP3 utilizza la porta 110.

POP3 è un protocollo semplice ed intuitivo. Ci permette di fare:

- Scaricare i messaggi
- Leggere
- Cancellare

POP3 lavora in tre fasi:

1. Autorizzazione (Autenticazione)
2. Transazione
3. Aggiornamento

POP3

~ Funzionamento ~

Abbiamo due processi: un processo client POP3 -> user-agent

un processo server POP3 -> mail server

Il server può rispondere al client in due modi:

- +OK -> se il comando precedente è andato a buon fine
- -ERR -> se il comando precedente ha dato dei problemi

POP3

~ Fase 1 ~

- - - FASE 1 - - -
Esito positivo

C telnet mailserver 110

C USER Bob

S +OK, USER OK

C PASS 1234

S +OK, PASSWORD LOGGED SUCCESFULLY

- - - FASE 1 - - -
Esito negativo

C telnet mailserver 110

C USER Bob

S +OK, USER OK

C PASS 12345

S -ERR, PASSWORD NOT RECOGNISED

POP3

~ Fase 2 ~

Nella fase di transazione POP3 mi chiede la modalità di transazione:

- Modalità iniziale di transazione di POP3 definita: “Scarica e cancella”.

- Modalità “Scarica e mantieni”. Scarico la posta sul PC locale e la mantengo nel server è la più utilizzata.

Possibili comandi

- LIST -> Mi restituisce la lista dei file con la grandezza.

- RETR -> Comando per scaricare il messaggio richiesto.

- DELE -> Comando per cancellare il messaggio richiesto

- QUIT -> Comando per chiudere la sessione in atto.

POP3

~ Fase 2 ~

C LIST

S 1 498

S 2 912

S .

Lista dei messaggi presenti sul server

C RETR 1

S Ti piace la scuola?

S Quale è la tua materia preferita?

S .

Viene richiesto il messaggio 1

C DELE 1

Viene "eliminato" il messaggio 1

POP3

~ Modalità 2 ~

Viene richiesto il messaggio 2

C RETR 2

S In allegato la nuova scansione oraria

S Cordiali saluti.

S .

Chiusura della connessione

C QUIT

S +OK POP3 SERVER SIGNING OFF

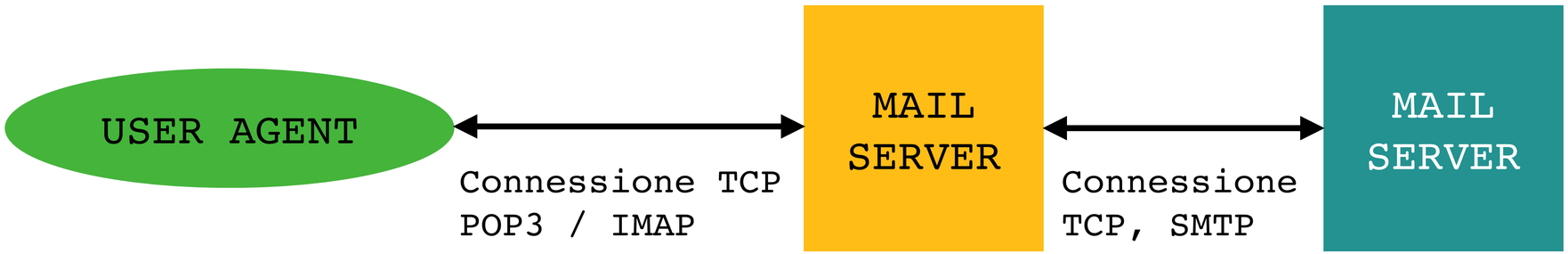
L’utente può decidere la modalità di funzionamento del protocollo. Durante la configurazione dell’user-agent.

POP3

~ Aggiornamento ~

Durante la fase di transizione tra l'ultimo comando RETR e il comando QUIT io posso vedere se ci sono dei nuovi messaggi o eseguire le cancellazioni dei messaggi marchiati.

~ Struttura ~



POP3

~ Confronto con IMAP ~

La differenza tra POP3 ed IMAP sta nella gestione delle cartelle create a livello di user-agent.

POP3 non ha insito in lui la gestione delle cartelle se non quelle di default create dallo user-agent stesso.

Vantaggi e svantaggi di POP3

+ Facile

+ Leggero

- Limitato

IMAP

IMAP

~ Internet Message Access Protocol ~

Definito dal RFC 3501.

Utilizza la porta 143. La versione sicura, IMAPS, è configurabile sulla porta 993.Il protocollo è stato creato per accedere alla casella di posta e non solo per leggere il contenuto dei messaggi.

IMAP in breve:

- Permette la gestione delle cartelle.
- IMAP richiede che la connessione sia costantemente attiva.
- IMAP è più complesso.
- Più client connessi contemporaneamente.

IMAP

~ Vantaggi rispetto a POP3 ~

- Permette più client contemporaneamente.

- Permette la gestione delle cartelle.

- Permette una connessione di durata maggiore rispetto ad POP3.

- Gestione smart dei messaggi in formato MIME, possiamo scaricare dal server il testo del messaggio e non gli allegati.
- Grazie ad IMAP4 abbiamo la gestione dello stato di un messaggio.
- Ricerca dei messaggi lato server e download da parte dell'user-agent di soli i messaggi interessati e non l'intero archivio.

IMAP

~ Svantaggi rispetto a POP3 ~

- Molto più complesso.

- Richiede una connessione persistente.

- IMAP per l’invio del messaggio richiede un doppio invio.

- Il singolo utente può utilizzare buona parte delle risorse di un server.

- Gli sviluppatori hanno creato un protocollo in parte libero, non è super serrato permette alcune personalizzazioni.

IMAP

~ Comandi ~

-**LOGOUT**: il client vuole chiudere la connessione.

-**AUTHENTICATE**: serve per specificare il metodo di autenticazione.

-**LOGIN**: serve per effettuare l’accesso al server.

-**CREATE**: serve per creare una cartella.

-**DELETE**: serve per eliminare una cartella.

-**RENAME**: serve per rinominare una cartella.

-**LIST**: serve per ottenere la lista della cartelle.

IMAP

~ Comandi ~

-**STATUS**: ci restituisce lo stato della cartella.

-**APPEND**: serve per andare ad aggiungere un nuovo messaggio.

-**CLOSE**: elimina tutti i messaggi che sono marcati.

-**SEARCH**: permette di ricercare i messaggi.

-**FETCH**: permette di ottenere i dati correlati a un messaggio.

-**STORE**: permette di modificare i dati di un messaggio presente.

-**COPY**: permette di copiare i messaggi in una cartella.

IMAP

~ Risposte ~

Sintassi uniforme alla maggior parte dei comandi:

-**OK** [messaggio] -> indica un successo.

-**NO** [messaggio] -> indica che vi è stato un problema.

-**BAD** [messaggio] -> indica che non ha capito il comando o che la sintassi è errata.

Quando la connessione viene chiusa: **BYE** [messaggio]

Casi particolari:

LIST -> la lista di tutte le cartelle che combaciano con i parametri passati.

SEARCH -> ci restituisce un elenco di numeri.

IMAP

~ Comunicazione ~

```
S:   * OK IMAP4rev1 Service Ready
C:   a001 login mrc secret
S:   a001 OK LOGIN completed

C:   a002 select inbox
S:   * 18 EXISTS
S:   * FLAGS (\Answered \Flagged \Deleted \Seen \Draft)
S:   * 2 RECENT
S:   * OK [UNSEEN 17] Message 17 is the first unseen message
S:   * OK [UIDVALIDITY 3857529045] UIDs valid

S:   a002 OK [READ-WRITE] SELECT completed

C:   a003 fetch 12 full
```

FASE DI LOGIN

SELEZIONE CARTELLE ESISTENTI

FETCH DI UN MESSAGGIO

IMAP

~ Comunicazione ~

```
S:    * 12 FETCH (FLAGS (\Seen) INTERNALDATE "17-Jul-1996 02:44:25 -0700"
       RFC822.SIZE 4286 ENVELOPE ("Wed, 17 Jul 1996 02:23:25 -0700 (PDT)"
       "IMAP4rev1 WG mtg summary and minutes"
       (("Terry Gray" NIL "gray" "cac.washington.edu"))
       (("Terry Gray" NIL "gray" "cac.washington.edu"))
       (("Terry Gray" NIL "gray" "cac.washington.edu"))
       ((NIL NIL "imap" "cac.washington.edu"))
       ((NIL NIL "minutes" "CNRI.Reston.VA.US")
       ("John Klensin" NIL "KLENSIN" "MIT.EDU")) NIL NIL
       "<B27397-0100000@cac.washington.edu>")
        BODY ("TEXT" "PLAIN" ("CHARSET" "US-ASCII") NIL NIL "7BIT" 3028
        92))

S:     a003 OK FETCH completed

C:     a004 fetch 12 body[header]
```

IMAP

~ Comunicazione ~

```
S:    * 12 FETCH (BODY[HEADER] {342}
S:    Date: Wed, 17 Jul 1996 02:23:25 -0700 (PDT)
S:    From: Terry Gray <gray@cac.washington.edu>
S:    Subject: IMAP4rev1 WG mtg summary and minutes
S:    To: imap@cac.washington.edu
S:    cc: minutes@CNRI.Reston.VA.US, John Klensin <KLENSIN@MIT.EDU>
S:    Message-Id: <B27397-0100000@cac.washington.edu>
S:    MIME-Version: 1.0
S:    Content-Type: TEXT/PLAIN; CHARSET=US-ASCII)

S:    a004 OK FETCH completed

C:    a005 store 12 +flags \deleted
S:    * 12 FETCH (FLAGS (\Seen \Deleted))
S:    a005 OK +FLAGS completed
C:    a006 logout
S:    * BYE IMAP4rev1 server terminating connection
S:    a006 OK LOGOUT completed
```

AGGIUNTA DEL FLAG AL MESSAGGIO

CHIUSURA DELLA CONNESSIONE

DNS

DNS

~ Domain Name System ~

Permette di tradurre un hostname in un indirizzo IP.
Si basa sugli RFC 882 e 883. Utilizza la porta 53.

Il DNS offre i seguenti servizi:
- Traduzione di un hostname in un indirizzo IP.
- L'utilizzo degli alias
- L'utilizzo degli alias per i mail server
- La possibilità di avere più indirizzi IP per uno stesso server.

I server hanno una gerarchia:
- DNS root server
- Server di primo livello
- Server di secondo livello

DNS

~ Gerarchia ~

I server di secondo livello vengono anche detti server autoritari.

I server di primo livello devono essere sempre allineati, ovvero se il sistema inserisce un nuovo server di secondo livello devo comunicarlo a tutti i server di primo livello.

I server di primo livello gestiscono i TLD: ovvero .com / .eu / .it / .fr / .edu / .jp

Nei server autoritari invece ritroviamo l'indirizzo IP e l'hostname di ogni server gestito dall'organizzazione.

Con il protocollo DNS è possibile avere dei server locali per velocizzare la ricerca.

DNS

~ Formato del record ~

Ogni record del database distribuito ha la seguente forma:

NAME VALUE TYPE TTL

Dove nome è la chiave e il valore è ciò che voglio cercare.

TIPO può essere:

- A: per indirizzi IPv4
- AAAA: per indirizzi IPv6
- NS: tipo puntatore
- CNAME: per gli alias
- MX: utilizzato per recuperare i nomi dei mailserver
- PTR: utilizzato per la ricerca inversa

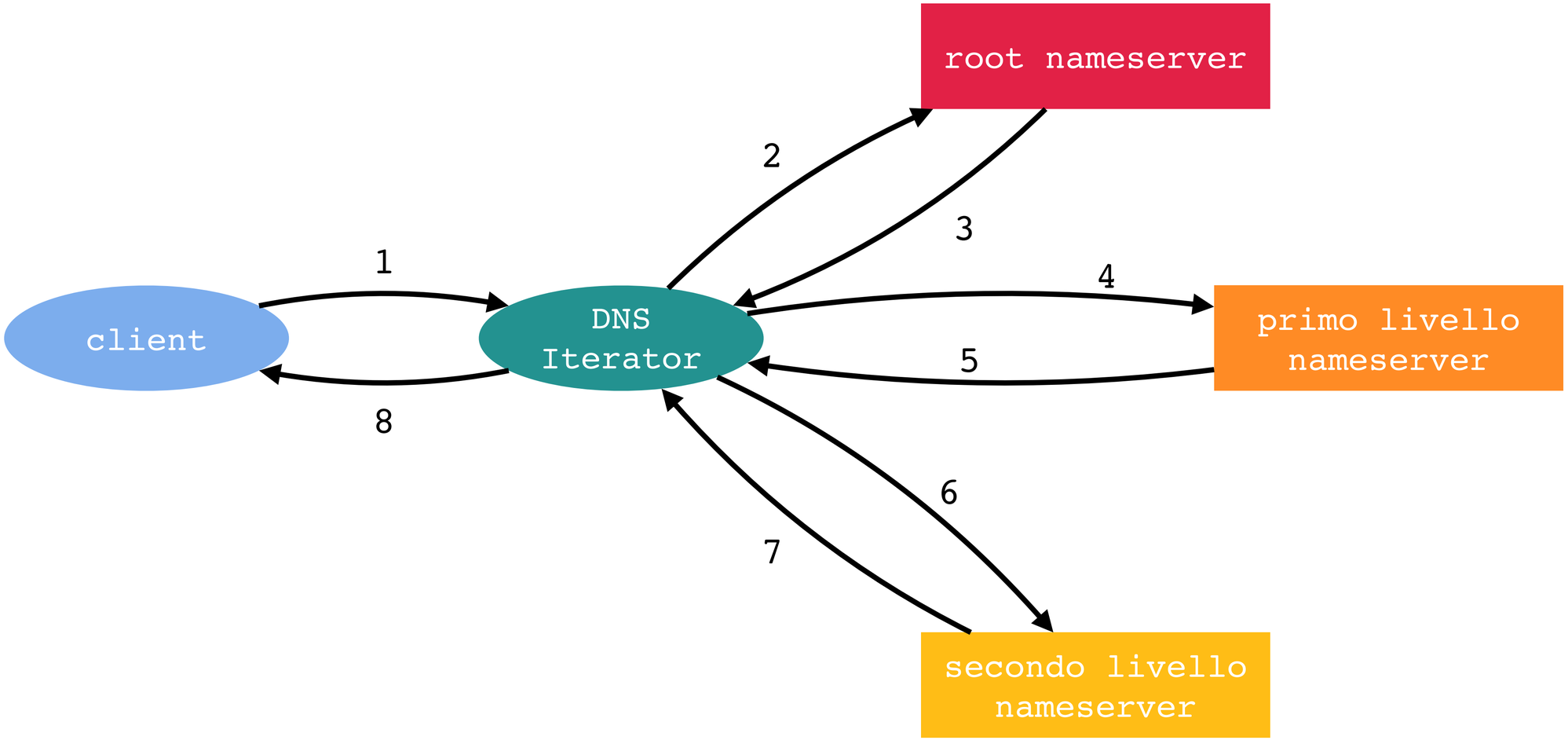
DNS

~ Messaggio di richiesta ~

IDENTIFICATION	FLAGS
#QUESTION	#ANSWER RRs
#AUTHORITY RRs	#ADDITIONAL RRs
QUESTIONS	
ANSWERS	
AUTHORITY	
ADDITIONAL INFORMATION	

DNS

~ Comunicazione ~



RIP

RIP

~ Routing Information Protocol ~

Si utilizza questo protocollo per permettere lo scambio di informazioni da parte dei routers.

Definito dal RFC 1058. Il protocollo utilizza UDP e la porta 520.

Il protocollo si basa sugli algoritmi di distance vector. Viene utilizzato l'algoritmo di Bellman Ford che si contrappone come metodo a Dijkstra. E' un algoritmo che converge velocemente ma ahimè è utilizzabile solo con pochi hop.

Il router che implementa RIP provvede ad inviare in broadcast a tutti i router connessi la propria tabella di routing come segmento. In qualche minuto otterremo la convergenza dell'algoritmo e quindi le strade più veloci.

RIP

~ Versioni ~

Nella seconda versione di RIP oltre a passare all'utilizzo di un gruppo multicast invece del broadcast. Inoltre sempre in questa versione si supporta la suddivisione degli indirizzi: CIDR.

RIP [versione 1] prevede due messaggi:

- Messaggio di richiesta: richiede a un router vicino di condividere con lui la sua routing table.
- Messaggio di risposta: contiene la routing table.

RIP ha il problema che ci mette molto a convergere verso la versione finale e in caso di loop non si arriva mai a una soluzione.

TELNET

Telnet

~ Generalità ~

E’ un protocollo che si utilizza per la comunicazione tra due host. In seguito con 854. Utilizza la porta 23.

Telnet a livello di trasporto utilizza TCP.

La versione iniziale di Telnet non era sicura, ma questo non era un problema in quanto veniva utilizzato principalmente da organizzazioni private.

In seguito negli anni ’90 con l’arrivo di SSH Telnet è stato rimpiazzato. Al giorno d’oggi usato per scopi di debug.

La comunicazione telnet è bidirezionale.

Il testo utilizza la codifica a 8bit. Ma non tutti i caratteri sono stampabili.

Telnet

~ Scopi ~

Telnet è stato creato con i seguenti scopi:

- per aver la possibilità di usare un terminal virtuale: il terminale prende il nome di NVT, che oltre a uno schermo prevede una tastiera. Il funzionamento è half-duplex e si lavora con un buffer.

- per poter negoziare delle opzioni per l'utilizzo di servizi aggiuntivi tramite l'NVT.

- per poter garantire la gestione di ACK/comandi durante la negoziazione delle opzioni.

SSH

SSH

~ Secure shell ~

Sfrutta la porta 22. Sfrutta TCP.

Nato come evoluzione di Telnet.

Utilizzato per:

- Effettuare il login su altre macchine remote
- Eseguire comandi su macchine remote
- Per garantire sicuri trasferimenti di file

Vi sono diverse versioni di SSH. Si basa su un’architettura client-server, un client di SSH è presente in quasi tutti i sistemi operativi moderni.

SSH utilizza la crittografia chiave pubblica per autenticare gli utenti.

SSH

~ Secure shell ~

SSH crea un tunnel di comunicazione client-server che è sicuro e fuori dalla portata degli attacchi, in quanto crittografato.

L'architettura prevista dal protocollo si suddivide in 4 ed è stata definita da:

- RFC 4251: definisce l'architettura interna di SSH
- RFC 4252: definisce come l'utente deve autenticarsi
- RFC 4253: definisce alcuni dettagli del livello di trasporto
- RFC 4254: definisce alcuni dettagli sul livello di connessione

SSH non è perfetto sono state infatti scovate alcune problematiche legate a overflow e un metodo di criptatura del messaggio.

SSH

~ Architettura ~

Ritroviamo un'architettura a 3 livelli che si appoggia all'architettura TCP/IP:

-Livello di trasporto: permette lo scambio delle chiavi per l'autenticazione del server, si può anche definire ogni quanto aggiornare le chiavi.

-Livello di autenticazione: è il modo in cui si può procedere all'identificazione del client.

-Livello di collegamento: viene instaurato il binario di collegamento, dove viaggeranno i dati, si deve poter garantire una minima banda passante, un controllo del flusso viene effettuato dal protocollo SSH internamente. Tutti i collegamenti sono full-duplex.

SNMP

SNMP

~ Simple Network Management Protocol ~

Definito da diversi RFC. Il protocollo utilizza le porte **161** e **162**, una versione sicura è disponibile sulle porte 10161 e 10162. A livello di trasporto troviamo UDP.

Viene utilizzato per gestire le apparecchiature di rete e la rete stessa. Anche se il nome può trarre in inganno la gestione della rete è tutto altro che semplice.

All'interno di SNMP il nostro server prende il nome di MANAGER e il client prende il nome di AGENT.

SNMP

~ SNMP versioni ~

Vi sono tre versioni, più quella sicura, che possiamo riassumere così:

- La prima versione di SNMP è stata criticata per via della bassa sicurezza.

- La seconda versione ha introdotto la sicurezza nella comunicazione tra due manager. Inoltre questa versione aumenta la dimensione dei counters. La versione 2 introduce anche la possibilità di far diventar un agent come proxy.

- La terza versione introduce la crittografia e un cambiamento a livello sintattico.

SNMP

~ PDU ~

VERSION	COMMUNITY	PDU-TYPE	REQUEST-ID	ERROR-STATUS	ERROR-INDEX	VARIABLE-BINDINGS

Possibili valori di PDU-TYPE:

-**GetRequest**: Il manager richiede il valore di un parametro all'agent.

-**SetRequest**: Il manager chiede di impostare uno o più parametri.

-**GetNextRequest**: Il manager chiede a un agent i valori delle variabili disponibili.

-**GetBulkRequest**: richieste multiple di tipo GetNextRequest.

-**Response**: contiene la risposta alle precedenti richieste.

-**Trap**: vengono inviati da parte di un agent dei dati al manager.

-**InformRequest**: Serve per avvertire l'agent che il manager ha ricevuto la sua richiesta

SNMP

~ Management Information Base ~

La comunicazione tra due nodi prevede lo scambio di MIB.

Le MIB sono definite da un RFC.

All’interno di queste strutture dati possiamo trovare diverse informazioni come ad esempio:

-Il numero di pacchetti ricevuti da un dispositivo di rete

-Lo stato di un dispositivo

-Il path per raggiungere un nodo della rete