

client-server

P2P

Compito dello sviluppatore : progettare l'architettura dell'applicazione e stabilire la sua organizzazione sui vari sistemi periferici.

ARCHITETTURA

CLIENT-SERVER : Vi e' sempre un host attivo (= **SERVER**)

che risponde alle richieste di servizio degli host
(= **CLIENT**)

SERVER :

- sempre attivo
- indirizzo IP fisso conosciuto
- ospitato nei **DATA CENTER**

CLIENT :

- comunicano attraverso i server
- puo' contattare il server in qualsiasi momento inviandogli un pacchetto
- puo' avere IP dinamico
- non comunicano direttamente tra loro

ARCHITETTURA

PEER-PEER (P2P) :

- l'infrastruttura di server e' minima o assente.
- comunicazione diretta tra coppie di host (peer) collegati in modo intermittente

PROCESSI COMUNICANTI

PROCESSO = programma in esecuzione su un sistema.

- processi in esecuzione sullo stesso sistema comunicano utilizzando un approccio interprocesso (**INTER-PROCESS COMMUNICATION**)
- processi su due sistemi terminali comunicano scambiandosi **messaggi**.

Per ogni coppia di processi comunicanti :

- **CLIENT PROCESS** = avvia la comunicazione
- **SERVER PROCESS** = attende di essere contattato per iniziare la sessione

SOCKET = interfaccia software con cui un processo invia/riceve messaggi.

API (application programming interface) = interfaccia tra applicazione e rete.

Per identificare il processo ricevente serve specificare:

1) indirizzo dell'host = **IP**

2) **identificatore** del processo ricevente (la socket che deve ricevere il dato)
= **NUMERO DI PORTA** di destinazione

QUALI SERVIZI IL PROTOCOLLO A LIVELLO DI TRASPORTO PUÒ OFFRIRE
A UN'APPLICAZIONE CHE LI INVOCA ?

- **TRASFERIMENTO DATI AFFIDABILE** (data integrity)

ci sono applicazioni loss-tolerant (es. audio, video)

e applicazioni che necessitano totale certezza (es. file transfer)

- **THROUGHPUT**

ci sono applicazioni sensibili alla banda che richiedono requisiti specifici di throughput (es. multimedia)

e applicazioni "elastiche" che possono far uso di tanto o poco throughput a seconda della disponibilità (es. email)

- **TEMPORIZZAZIONE**

alcune app vogliono la garanzia di basso delay
(es. teleconferenze, giochi multimediali)

- **SICUREZZA** (es. crittografia)

SERVIZI DI TCP :

- orientato alla connessione
- trasporto affidabile
- controllo della congestione
- controllo del flusso

NON PREVEDE temporizzazione, garanzia
di throughput minimo e sicurezza

SERVIZI DI UDP :

- senza connessione
- trasferimento non affidabile

NON PREVEDE affidabilità, controllo di flusso e congestione, temporizzazione, garanzia di throughput minimo, sicurezza e setup di connessione.

TCP puo' essere facilmente arricchito a livello applicativo con SSL per fornire servizi di sicurezza

PROTOCOLLO DI APPLICATION LAYER

definisce come i processi di un'applicazione, in esecuzione su sistemi periferici diversi, si scambiano i messaggi.

- tipi di messaggi scambiati
- sintassi dei vari tipi di messaggio
- semantica dei campi
- regole di invio e risposta

WEB : opera su richiesta (on demand)

HTTP (hypertext transfer protocol)

↳ implementato in due programmi 
↳ definisce la struttura dei messaggi e le modalita' di trasferimento

PAGINA WEB = documento = insieme di oggetti

OGGETTO = file indirizzabile tramite URL

La maggioranza delle pagine web consiste di un **file HTML principale** e diversi oggetti referenziati da esso.

Il file HTML **referenzia altri oggetti** nella pagina tramite il loro URL.

URL : due componenti

- nome dell'host del server che ospita l'oggetto
- percorso dell'oggetto

https://informatica.i-learn.unito.it/course/view.php

nome host percorso

BROWSER WEB (es. Firefox) : implementa il lato client.

WEB SERVER : implementa il lato server.

HTTP usa TCP (no UDP) come protocollo di trasporto:

- Il client inizia una connessione TCP col server
- client e server accedono a TCP attraverso le proprie socket
- i messaggi HTTP vengono scambiati tra browser (=client) e web server
- la connessione TCP viene chiusa.

VANTAGGIO: HTTP non si deve preoccupare dei dati smarriti o di come TCP recuperi le perdite o riordini i dati all'interno della rete (sono compiti di TCP)

N.B. HTTP è protocollo senza memoria di stato (stateless):

i server HTTP non mantengono informazioni sui client !

Due tipi di connessione HTTP:

NON PERSISTENTE

Ciascuna coppia richiesta/risposta deve essere inviata su una connessione TCP separata

PERSISTENTE

coppie richiesta/risposta inviate tutte sulla stessa connessione TCP.



RTT (round-trip time) = tempo impiegato da un piccolo pacchetto per viaggiare dal client al server e poi tornare al client.

↳ include ritardi di propagazione, di accodamento e di elaborazione

Per ogni oggetto, ci vuole:

- 1 RTT per iniziare la connessione TCP
- 1 RTT per far sì che il server HTTP riceva il messaggio di richiesta e i primi byte di risposta tornino al client
- tempi di trasmissione

⇒ Il tempo totale di risposta in HTTP non persistente è di 2 RTT per oggetto

Con connessioni persistenti, invece:

- il server lascia la connessione TCP aperta dopo l'invio di una risposta
- le richieste/risposte successive tra gli stessi client/server possono essere trasmesse sulla stessa connessione
- il server può spedire allo stesso client più oggetti di seguito senza aspettare le risposte delle richieste precedenti
- quando il server riceve richieste in sequenza, invia gli oggetti con la stessa modalità

Leggere 2.2.3 dal libro

COOKIES : consentono ai server di tener traccia degli utenti.

WEB CACHE = entità di rete che soddisfa richieste HTTP al posto del web server.
(proxy server)

- ↳ il browser di un utente può essere configurato in modo che tutte le richieste HTTP dell'utente vengano dirette alla web cache
- ↳ ogni richiesta di oggetto da parte del browser viene inizialmente diretta alla cache :
 - SE l'oggetto è in cache, questa manda l'oggetto al client ;
 - ALTRIMENTI la cache apre una TCP verso il server d'origine, poi riceve l'oggetto e lo invia al client
- può ridurre i tempi di risposta alle richieste dei client ;
- riduce il traffico sul collegamento di accesso a Internet, senza aumentare l'ampiezza di banda frequentemente e ridurre i costi
- riduzione del traffico totale del Web

PROBLEMA : la copia di un oggetto che risiede in cache potrebbe essere scaduta

⇒ **CONDITIONAL GET** = meccanismo di HTTP che permette alla cache di verificare se i suoi oggetti sono aggiornati

- ↓ |
 - usa il metodo GET
 - intestazione : If-modified-since: <date>

comunica al server di inviare l'oggetto solo se è stato modificato rispetto alla data specificata
→ se non è stato modificato, il server risponde 304 Not Modified

HTTP/1.0 = prima versione di HTTP

↳ gestisce le richieste di un server tramite BLOCKING :

il client deve attendere la risposta alla richiesta inviata al server prima di effettuarne altre.

→ una richiesta alla volta → ritardi

HTTP/1.1 ha introdotto la possibilità di inviare molteplici richieste GET in sequenza su una singola TCP

- il server risponde in-order (FCFS = first come first served)
- con FCFS, piccoli oggetti possono dover aspettare dietro ad oggetti più grandi per essere trasmessi (**head-of-line blocking**)
⇒ ritardi
- ulteriori ritardi per la ritrasmissione di segmenti TCP persi.

HTTP/2 → diminuzione dei ritardi nelle richieste multioggetto

↳ maggior flessibilità del server nell'invio degli oggetti al client :

- ordine di trasmissione basato sulla priorità specificata dal client;
- inviati al client anche oggetti non richiesti (PUSH) → + efficienza ;
- oggetti divisi in frame che vengono programmati per mitigare il HOL blocking.

↓
Su una connessione TCP singola :

può ancora essere soggetto a ritardi dovuti al recupero delle perdite .

HTTP/3 migliora la sicurezza e introduce un controllo più raffinato sugli errori e sulla congestione tramite UDP

POSTA ELETTRONICA

Tre componenti principali

- **USER AGENT** : (es. Outlook, Apple Mail) consentono agli utenti di leggere, rispondere, inoltrare, salvare e comporre messaggi
- **MAIL SERVER** : parte centrale dell'infrastruttura del servizio.
Ciascun destinatario ha una casella di posta (**MAILBOX**) collocata in un mail server.
Se il server del mittente non può consegnare la posta al server destinatario, allora questo lo trattiene in una **CODA DI MESSAGGI** (MESSAGE QUEUE) per trasferirla in un secondo momento.

- **SMTP** (Simple Mail Transfer Protocol) :

- trasferisce i messaggi dal mail server del mittente a quello del destinatario.
- molto più vecchio di HTTP:
tratta corpo e intestazione dei messaggi come ASCII a 7 bit (penalizzante!)
- di solito non usa server intermedi per inviare la posta.
- fa uso di connessioni persistenti: se il mail server di invio ha molti messaggi da inviare allo stesso mail server in ricezione, può mandarli tutti sulla stessa connessione TCP

Vedi formato messaggi e comandi : 2.4.1 del libro

	HTTP	SMTP
trasferisce file:	da un web server a un web client	da un web server a un altro
e' un:	PRODOTTO PULL : qualcuno carica informazioni su un web server e gli utenti lo usano per ottinarle (PULL) a sé.	PRODOTTO PUSH : il mail server di invio spinge (PUSH) i file al mail server di ricezione
Vincoli :	nessuno	ASCII a 7 bit

IMAP (INTERNET MAIL ACCESS PROTOCOL) definito in [RFC 3501]

- associa a una cartella ogni messaggio inviato al server.
- fornisce comandi per consentire agli utenti di organizzare i messaggi e effettuare ricerche nelle cartelle remote in base a criteri specifici

Gli host vengono identificati da **hostname** e da indirizzi IP

struttura gerarchica:

leggendolo da sinistra a destra, si ottengono informazioni più specifiche sulla collocazione dell'host in Internet.

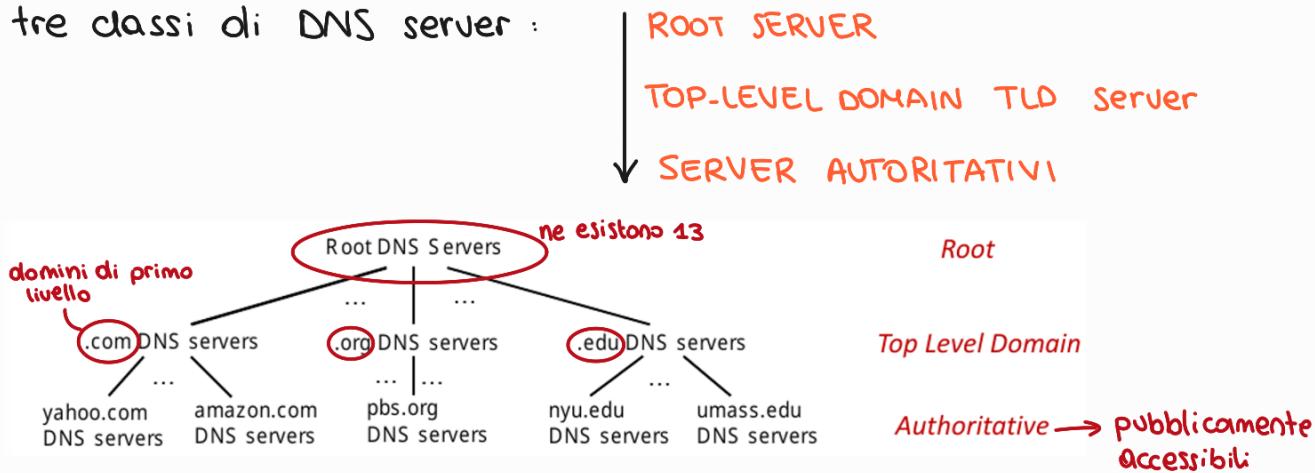
servizio per tradurre hostname in indirizzo IP :

DNS (DOMAIN NAME SYSTEM)

- = database distribuito implementato in una gerarchia di **DNS server**.
 - = protocollo a livello di applicazione che consente agli host di interrogare il database.
- oltre alla traduzione, mette a disposizione altri servizi :
- **HOST ALIASING**
 - **LOAD DISTRIBUTION** (distribuzione del carico di rete)
 - ↳ i siti con molto traffico vengono replicati su più server, ciascuno eseguito su un host diverso e con IP diverso.
- PROPRIETÀ :
 - semplicità progettuale
 - inappropriato per l'attuale Internet (troppi host)
- schema centralizzato :
 - un solo punto di fallimento (se si guasta, ne risente l'intera Internet)
 - un singolo DNS dovrebbe gestire tutte le richieste (TROPPO TRAFFICO)
 - database distante → può causare ritardi
- SOLUZIONE**

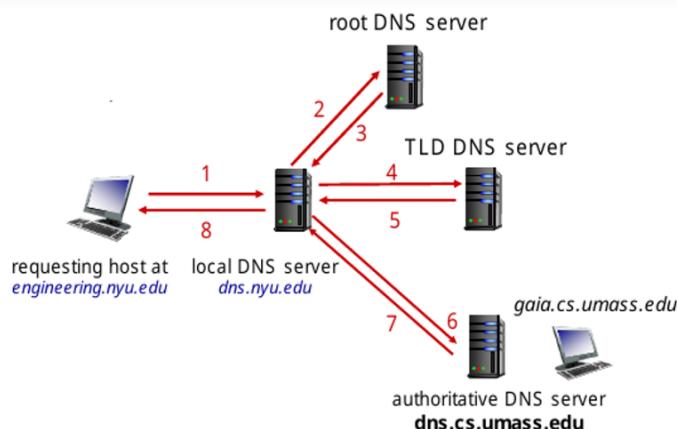
Per risolvere i problemi elencati sopra, il DNS utilizza un gran numero di server organizzati in maniera gerarchica e distribuiti in tutto il mondo.

Esistono tre classi di DNS server :

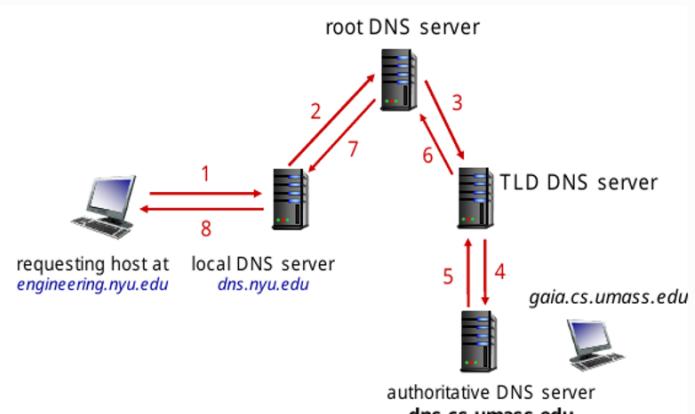


In teoria ogni richiesta DNS può essere iterativa (metodo usato) o ricorsiva.

QUERY ITERATIVE



QUERY RICORSIVE



Il DNS usa il caching per migliorare le prestazioni di ritardo e ridurre il numero di messaggi DNS che "rimbalzano" su Internet.

↳ in una concatenazione di richieste, il DNS server che riceve una risposta può mettere in cache le informazioni (che vengono invalidate dopo un tempo fissato)

I server che implementano il database distribuito di DNS memorizzano i **RECORD DI RISORSA** (RR - resource record)

formato : (name, value, type, ttl)

TIME TO LIVE

(name, value, type, ttl)

- type = A \Rightarrow name = hostname ; value = IP
- type = NS \Rightarrow name = dominio ; value = hostname del server autoritativo
- type = CNAME \Rightarrow name = alias ; value = nome canonico (reale)
- type = MX \Rightarrow value = nome canonico del mail server che ha come alias il campo Name

Il protocollo e' basato su richieste e risposte su connessione UDP non crittografate

VIDEO STREAMING E CDN (content delivery Network)

Il traffico di stream video rappresenta il maggior consumatore di bandwidth
(es. Netflix, YT, ...)

diffuso su larga scala (tantissimi users)
ed in modalita' eterogenea (cablato vs mobile ;
alta vs bassa bandwidth ; ...)

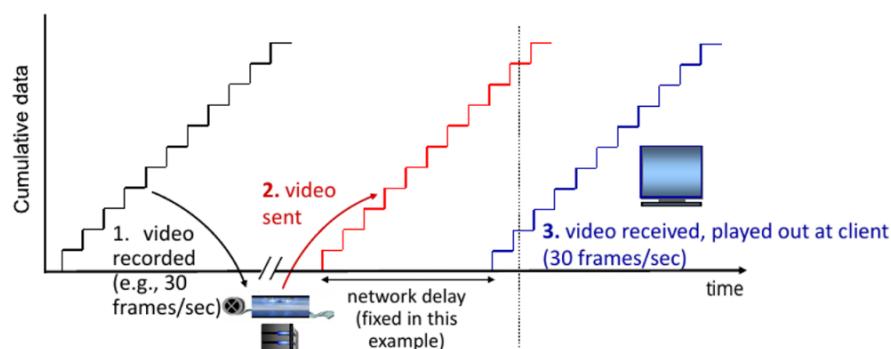
VIDEO = sequenza di immagini visualizzate a velocita' costante

IMMAGINE DIGITALE = array di Pixel rappresentati da bit

codice : utilizza la ridondanza all'interno (codifica spaziale)
e tra le immagini (codifica temporale \rightarrow frame) per diminuire
il numero di bit utilizzati per codificare l'immagine

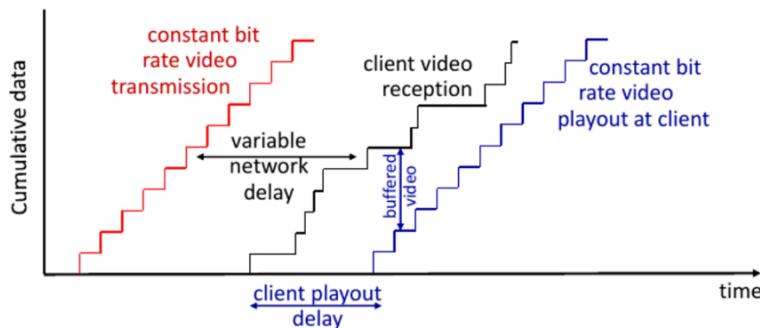
STREAMING STORED VIDEO :

Video server → INTERNET → Client



- la larghezza di banda da server a client varia nel tempo, insieme al livello di congestione della rete
- perdita dei pacchetti o ritardo dovuto alla congestione
⇒ ritardo di riproduzione o scarsa qualità del video
- vincolo di riproduzione continua: i tempi di riproduzione devono corrispondere ai tempi originali
↳ i network delay sono variabili (**JITTER**) ⇒ necessario un buffer client-side *
- altre possibili azioni: interazione del client (pausa, fast-forward, rewind, ...)

* PLAYOUT BUFFERING (buffer di riproduzione) → per compensare i ritardi



DASH (Dynamic Adaptive Streaming over HTTP)

- **server**: divide il video in vari blocchi, ciascuno codificato in velocità diverse.
↳ **manifest file**: fornisce URL per blocchi diversi
- **client**: stima periodicamente il bandwidth e sceglie la velocità di codifica necessaria nei diversi momenti
↳ **intelligence**: il client decide quando richiedere il blocco, a quale velocità di codifica richiederlo e dove richiederlo.

CDN : content delivery network

- distribuito → archivia copie multiple di video di più siti

Esempio : Netflix

↳ l'abbonato richiede il contenuto, il service provider restituisce il manifest.
in questo modo il client recupera il contenuto a velocità elevata

PROGRAMMAZIONE SOCKET ~ come creare un'applicazione di rete

PRIMA DECISIONE DA PRENDERE : TCP o UDP ?

Ricordando che : TCP → orientato alle connessioni e fornisce un canale affidabile per il flusso dei byte

UDP → senza connessione e invia pacchetti di dati indipendenti da un sistema all'altro (no garanzie di consegna)

vedi esempi da slide 95 e 2.7.1/2.7.2 del libro + codici su moodle