**Creare un'applicazione di rete**

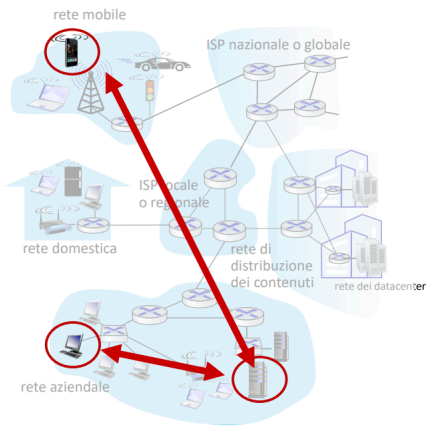
**Introduzione**

- Le applicazioni di rete permettono a programmi di girare su sistemi diversi e comunicare tra loro.
- Esempi: software di un server Web e browser.
- I dispositivi di rete (es. router) non eseguono applicazioni utente.

## Paradigmi di rete:

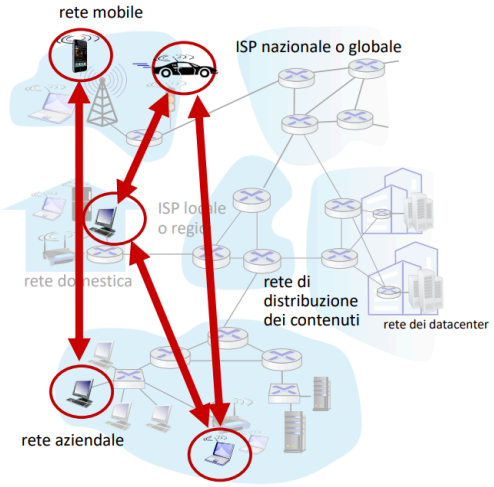
## ***Client-server***

- **Server**:
    - Sempre attivo
    - Indirizzo IP fisso
    - Spesso in datacenter per scalabilità
- **Client**:
    - Inizia la comunicazione con il server
    - Può contattare il server in qualsiasi momento
    - Può avere IP dinamici
    - Non comunica direttamente con altri client
- Esempi: Web, posta elettronica



## ***Peer-to-peer***

- Non c'è un server sempre attivo
- Host (peer) comunicano direttamente tra loro
- I peer richiedono e forniscono servizi a vicenda
- Vantaggi:
    - Scalabilità intrinseca
- Svantaggi:
    - Difficile da gestire
- Esempio: condivisione file P2P (BitTorrent)



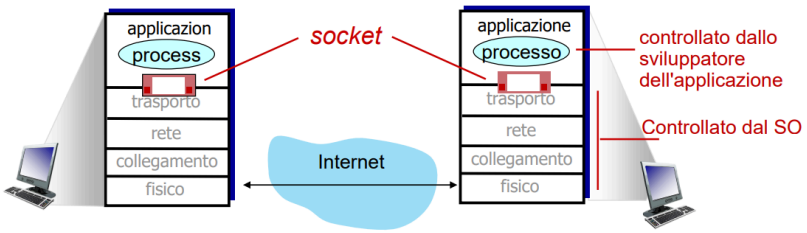

## ***Processi comunicanti***

- Processo: programma in esecuzione su un host
- Comunicazione tra processi:
    - Stesso host: IPC (definito dal SO)
    - Host diversi: scambio di messaggi
- Ruoli:
    - Client: inizia la comunicazione
    - Server: attende di essere contattato

- Nota: nelle applicazioni P2P, un processo può essere sia client che server (a seconda della sessione).

### ***Socket***

- Un processo invia/riceve messaggi tramite la sua socket (come una porta)
- Il processo mittente:
    - Invia il messaggio dalla propria "porta" (socket)
    - Presuppone un'infrastruttura esterna per il trasporto del messaggio




## ***Indirizzamento***

### ***Identificatori dei processi***

- Per ricevere messaggi, un processo necessita di un identificatore univoco.
- Un host ha un indirizzo IP univoco a 32 bit, **ma non identifica univocamente un processo**.
- L'identificatore di un processo comprende:
    - L'indirizzo IP dell'host.
    - Il numero di porta associato al processo.

### ***Numeri di porta***

- I numeri di porta sono assegnati dall'IANA ad applicazioni note:
    - Server HTTP: 80
    - Server di posta: 25

### Esempio

- Per inviare un messaggio HTTP al server `gaia.cs.umass.edu`:
    - Indirizzo IP: 128.119.245.12
    - Numero di porta: 80

# ***Protocolli***

## ***Definizione di un protocollo***

- Un protocollo a livello applicazione definisce:
    - Tipi di messaggi scambiati (es. richiesta, risposta).
    - Sintassi dei messaggi (campi e loro descrizione).
    - Semantica dei messaggi (significato delle informazioni).
    - Regole per l'invio e la ricezione dei messaggi.

## ***Tipi di protocolli***

- Protocolli di pubblico dominio:
    - Definiti nelle RFC (Request for Comments).
    - Accessibili a tutti.
    - Promuovono l'interoperabilità (es. HTTP, SMTP).
- Protocolli proprietari:
    - Definiti da singole aziende (es. Skype, Zoom).

# ***Servizi di trasporto***

## ***Requisiti di un servizio di trasporto***

- Le applicazioni possono richiedere diversi servizi di trasporto a seconda delle loro esigenze:
    - **Affidabilità**:
        - Alcune applicazioni (es. trasferimento file, transazioni web) richiedono un trasferimento 100% affidabile.
        - Altre applicazioni (es. audio) possono tollerare qualche perdita di dati.
    - **Sensibilità al fattore tempo**:
        - Alcune applicazioni (es. telefonia via Internet, giochi interattivi) richiedono bassi ritardi per essere efficaci.
    - **Throughput**:
        - Alcune applicazioni (multimediali) richiedono un'ampiezza di banda minima.
        - Altre applicazioni ("elastiche") utilizzano la banda disponibile.
    - **Sicurezza**:
        - Crittografia, integrità dei dati, ...

## Requisiti del servizio di trasporto di alcune applicazioni comuni

| applicazione | tolleranza alla perdita di dati | throughput | sensibilità al fattore tempo |
|---|---|---|---|
| trasferimento file | no | variabile | no |
| posta elettronica | no | variabile | no |
| documenti Web | no | variabile | no |
| audio/video in tempo reale | sì | audio: 5kbps-1Mbps<br>video:10kbps-5Mbps | sì, centinaia di ms |
| streaming audio/video memorizzati | Sì | come sopra | sì, pochi secondi |
| giochi interattivi | sì | fino a pochi kbps | sì, centinaia di ms |
| messaggistica istantanea | no | variabile | sì e no |

Application Layer: 2-13

### ***Servizio TCP***

- Fornisce un trasporto affidabile tra i processi di invio e ricezione.
- Caratteristiche:
  - Garantisce la consegna dei dati senza errori, perdite e nell'ordine di invio.
  - Controlla il flusso per evitare di sovraccaricare il destinatario.
  - Controlla la congestione per regolare la velocità di trasmissione in caso di rete sovraccarica.
  - È orientato alla connessione, richiedendo un handshake iniziale tra client e server.
- Limitazioni:
  - Non offre temporizzazione, garanzie sull'ampiezza di banda minima o sicurezza.

### ***Servizio UDP***

- Offre un trasferimento di dati inaffidabile tra i processi di invio e ricezione.
- Caratteristiche:
  - Non garantisce affidabilità, controllo di flusso, controllo della congestione, temporizzazione, ampiezza di banda minima o sicurezza.
  - Non richiede un setup di connessione.

### ***Perché utilizzare UDP?***

- Nonostante la sua inaffidabilità, UDP è utile per:
  - Applicazioni in cui la velocità è più importante dell'affidabilità (es. streaming audio/video).
  - Invio di dati che non richiedono una consegna garantita (es. DNS).

***Sicurezza***

- Sia TCP che UDP non offrono sicurezza intrinseca.
- Le password inviate in chiaro attraverso socket attraversano Internet senza protezione.

### Applicazioni Internet: protocollo a livello applicazione e protocollo di trasporto

| applicazione | protocollo a livello applicazione | Protocollo di trasporto sottostante |
|---|---|---|
| trasferimento file | FTP [RFC 959] | TCP |
| posta elettronica | SMTP [RFC 5321] | TCP |
| documenti web | HTTP [RFC 7230, 9110] | TCP |
| telefonia via Internet | SIP [RFC 3261], RTP [RFC 3550], o proprietario | TCP o UDP |
| streaming audio/video | HTTP [RFC 7230], DASH | TCP |
| giochi interattivi | WOW, FPS (proprietario) | UDP o TCP |

**Transport Layer Security (TLS)**

- TLS offre connessioni TCP cifrate con:
  - Controllo di integrità dei dati.
  - Autenticazione end-to-end.
- TLS è implementato a livello applicazione:
  - Le applicazioni utilizzano librerie TLS.
  - Le librerie TLS utilizzano TCP per il trasporto sottostante.
- Il testo in chiaro inviato nella socket attraversa Internet in forma crittografata.

## Conclusione

- La scelta tra TCP e UDP dipende dalle esigenze dell'applicazione:
  - TCP per applicazioni che richiedono affidabilità.
  - UDP per applicazioni che richiedono velocità.
- TLS è fondamentale per garantire la sicurezza delle comunicazioni su Internet.

# WEB e HTTP

## Ripasso della terminologia

- Una pagina web è composta da **oggetti**, ognuno dei quali può essere memorizzato su un server Web diverso.
- Un oggetto può essere un file HTML, un'immagine JPEG, uno script JavaScript, un foglio di stile CSS, un file audio, etc.
- Una pagina web è formata da un file HTML di base che include riferimenti a diversi oggetti, ognuno con un URL specifico (es. `www.someschool.edu/someDept/pic.gif`).

## Panoramica su HTTP

- **HTTP** (Hypertext Transfer Protocol):
  - Protocollo a livello applicazione del Web.
  - Modello client-server:
    - ***Client***: browser che richiede, riceve (usando il protocollo HTTP) e visualizza gli oggetti del Web.
    - ***Server***: il server Web che invia (usando il protocollo HTTP) oggetti in risposta alle richieste.


- **HTTP usa TCP**:
  - Il client inizializza una connessione TCP (crea una socket) con il server sulla porta 80.
  - Il server accetta la connessione TCP dal client.
  - Messaggi HTTP (messaggi di un protocollo di applicazione) scambiati tra browser (client HTTP) e server Web (server HTTP).
  - Connessione TCP chiusa.

- **HTTP è un protocollo "senza stato" (stateless)**:
    - Il server non mantiene informazioni sulle richieste fatte dal client.
    - Nota: i protocolli che mantengono lo stato sono complessi e richiedono la memorizzazione dello stato passato.

### Connessioni HTTP: due tipi

***Connessioni non persistenti***

1. Connessione TCP aperta.
2. Almeno un oggetto viene trasmesso su una connessione TCP.
3. Connessione TCP chiusa.

**Lo scaricamento di oggetti multipli richiede connessioni multiple.**

***Connessioni persistenti***

- Connessione TCP mantenuta aperta con il server.
- Più oggetti possono essere trasmessi su una singola connessione TCP tra client e server.
- Connessione TCP chiusa alla fine.

## *Connessioni non persistenti*

**Esempio:** L'utente immette l'URL:
`http://www.someSchool.edu/someDepartment/home.html`

1a. Il client HTTP avvia una connessione TCP con il server HTTP (processo) a `www.someSchool.edu` sulla porta 80.
1b. Il server HTTP su `www.someSchool.edu` in attesa di una connessione TCP sulla porta 80 "accetta" la connessione e notifica il client.
2. Il client HTTP invia un messaggio di richiesta HTTP (contenente l'URL) nella socket della connessione TCP. Il messaggio indica che il client desidera l'oggetto `someDepartment/home.html`.
3. Il server HTTP riceve il messaggio di richiesta, crea il messaggio di risposta contenente l'oggetto richiesto e lo invia nella sua socket.
4. Il server HTTP chiude la connessione TCP.
5. Il client HTTP riceve il messaggio di risposta contenente il file HTML e lo visualizza. Esamina il file HTML e trova riferimenti a 10 oggetti JPEG.

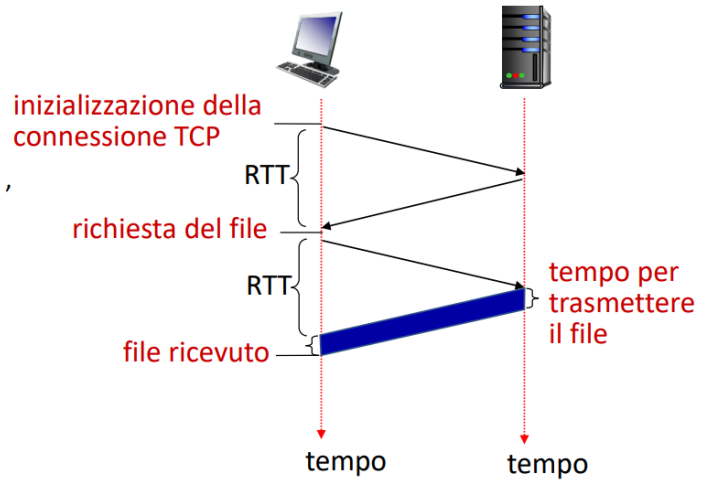
6. I passaggi 1-5 vengono ripetuti per ciascuno dei 10 oggetti JPEG.

**RTT (Round Trip Time):** tempo impiegato da un piccolo pacchetto per viaggiare dal client al server e tornare al client (compresi i ritardi di elaborazione, accodamento e propagazione).

**Tempo di risposta (per oggetto):**

- Un RTT per inizializzare la connessione TCP.
- Un RTT per il ritorno della richiesta HTTP e dei primi byte della risposta HTTP.
- Tempo di trasmissione del file/oggetto.

**Tempo di risposta con connessioni non persistenti = 2RTT + tempo di trasmissione del file.**




**Svantaggi delle connessioni non persistenti:**

- Richiedono 2 RTT per ogni oggetto.
- Overhead del sistema operativo per ogni connessione TCP.
- I browser spesso aprono connessioni TCP parallele per caricare gli oggetti referenziati.

## Connessioni persistenti (*HTTP 1.1*)

- Il server mantiene la connessione TCP aperta dopo aver inviato una risposta.
- I messaggi successivi tra gli stessi client/server vengono trasmessi sulla connessione aperta.
- Il client invia le richieste non appena incontra un oggetto referenziato.
- Un solo RTT per tutti gli oggetti referenziati.

# Messaggio di richiesta HTTP

## Struttura generale

Un messaggio di richiesta HTTP è un testo in formato ASCII, leggibile dall'utente, e si divide in diverse sezioni:

### 1. Riga di richiesta:

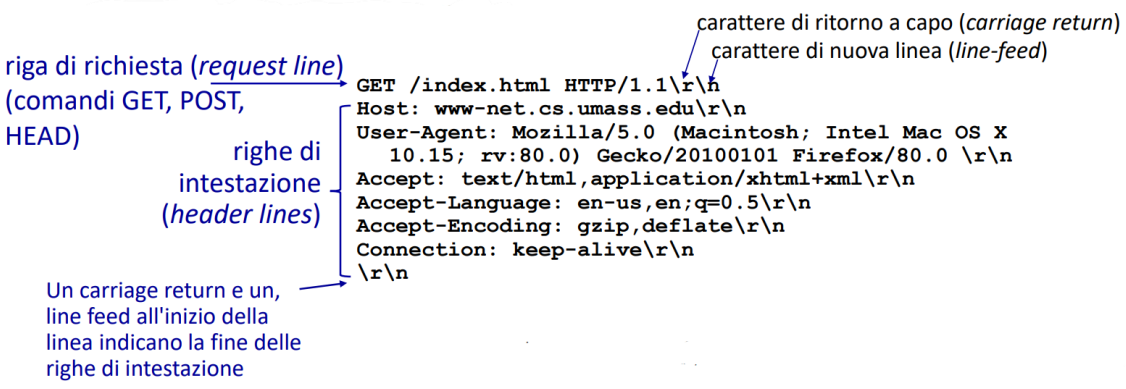
- **Metodo:** indica l'operazione da eseguire (es. GET, POST, PUT, HEAD).
- **URL:** specifica la risorsa richiesta sul server.
- **Versione del protocollo HTTP:** indica la versione del protocollo utilizzata (es. HTTP/1.1).

### 2. Intestazioni:

- Forniscono informazioni aggiuntive sulla richiesta, come:
  - `Host`: hostname e numero di porta del server di destinazione.
  - `User-Agent`: identifica l'applicazione e il sistema operativo del client.
  - `Accept`: tipi di contenuto supportati dal client.
  - `Accept-Language`: lingua preferita dal client.
  - `Accept-Encoding`: algoritmi di compressione supportati dal client.
  - `Connection`: indica se la connessione rimarrà aperta dopo la richiesta (default: `close` in HTTP/1.0, `keep-alive` in HTTP/1.1).

### 3. Corpo della richiesta (opzionale):

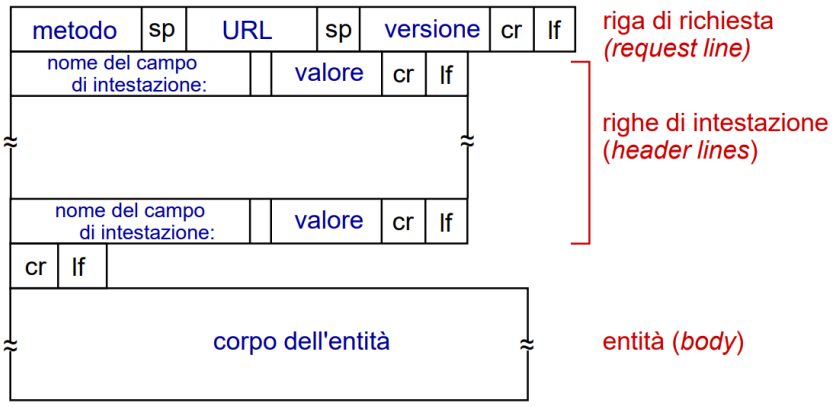
- Contiene dati da inviare al server (es. dati di un form HTML).




## Tipi di metodi di richiesta

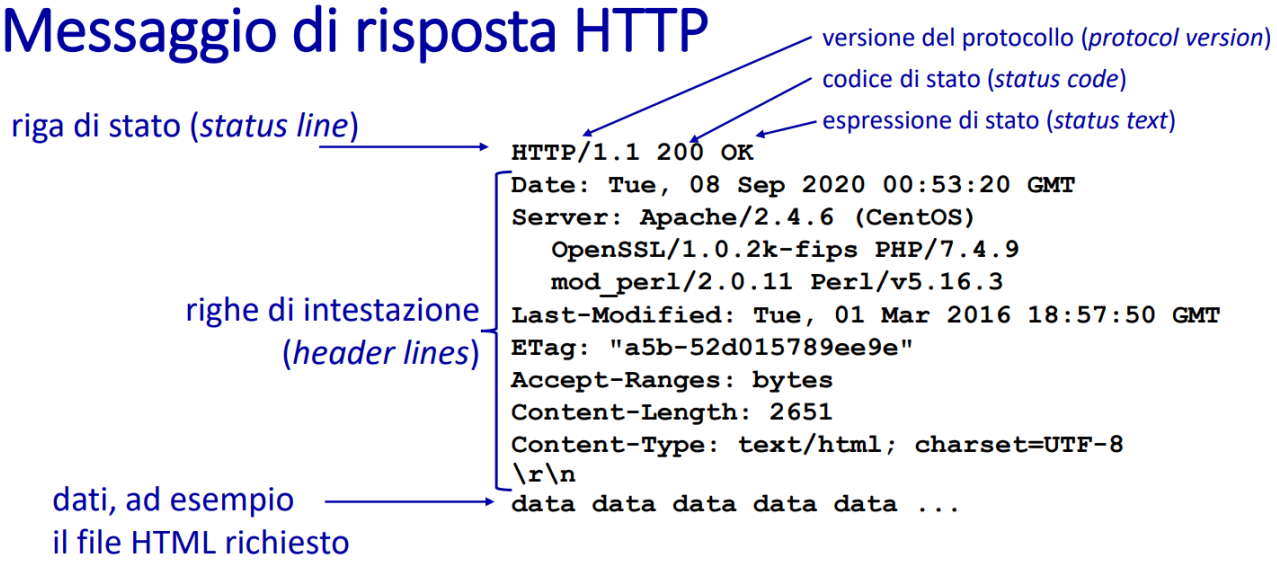
- **GET:** usato per recuperare dati da un server. I dati vengono inviati come parametri URL.
- **POST:** usato per inviare dati a un server. I dati vengono inviati nel corpo della richiesta.

- **HEAD:** simile a GET, ma restituisce solo le intestazioni della risposta, senza il corpo.
- **PUT:** usato per caricare un nuovo file o sostituire un file esistente sul server.




## Campi di intestazione nella risposta HTTP

- **Date:** data e ora della risposta.
- **Server:** software del server che ha gestito la richiesta.
- **Last-Modified:** data e ora dell'ultima modifica della risorsa.
- **Accept-Ranges:** indica se la risorsa supporta download parziali.
- **Content-Length:** lunghezza del corpo della risposta in byte.
- **Content-Type:** tipo di contenuto del corpo della risposta.



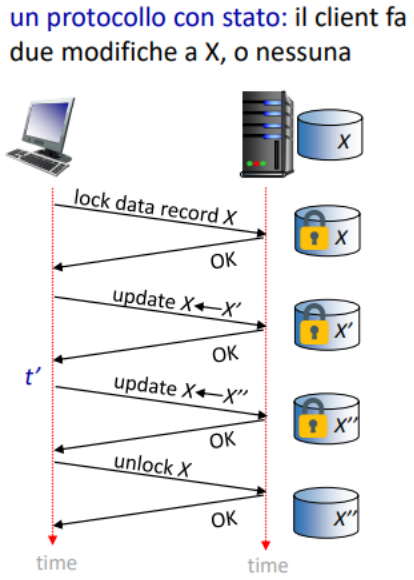
# Codici di stato della risposta HTTP

- I codici di stato indicano l'esito della richiesta.
- Sono raggruppati in 5 categorie:
    - **1xx Informational:** risposta intermedia (assente in HTTP/1.0).
    - **2xx Successful:** richiesta eseguita con successo.
    - **3xx Redirect:** necessario un redirect.
    - **4xx Client Error:** errore del client.
    - **5xx Server Error:** errore del server.
- Alcuni codici di stato comuni:
    - **200 OK:** richiesta eseguita con successo.
    - **301 Moved Permanently:** risorsa spostata in modo permanente (nuova posizione nell'intestazione `Location`).
    - **400 Bad Request:** richiesta non corretta.
    - **404 Not Found:** risorsa non trovata.
    - **500 Internal Server Error:** errore interno del server.

# Mantenere lo stato utente/server: i cookie

**Premessa:**

L'interazione HTTP GET/risposta è senza stato (stateless), il che significa che non c'è memoria di scambi di messaggi precedenti. Ogni richiesta è indipendente e non è necessario che client o server tengano traccia dello "stato" della conversazione.



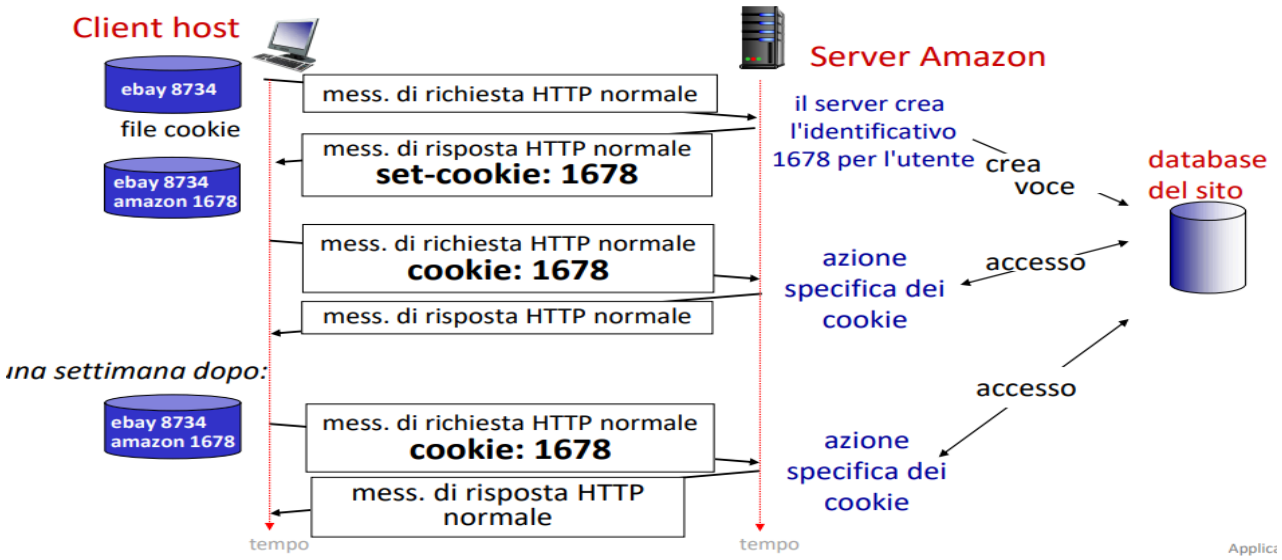

**I cookie:**

I siti web e il browser client usano i cookie per mantenere lo stato tra le transazioni. Questi "biscotti" digitali sono composti da quattro componenti:

1. **Intestazione "Set-Cookie" nel messaggio di risposta HTTP:** inviata dal server al client per creare un nuovo cookie.
2. **Intestazione "Cookie" nel messaggio di richiesta HTTP:** inviata dal client al server per includere i cookie esistenti.
3. **File cookie:** memorizzato sul dispositivo dell'utente e gestito dal browser.
4. **Voce nel database del sito web:** associata all'identificativo del cookie e memorizzante informazioni sullo stato utente.

**Esempio:**

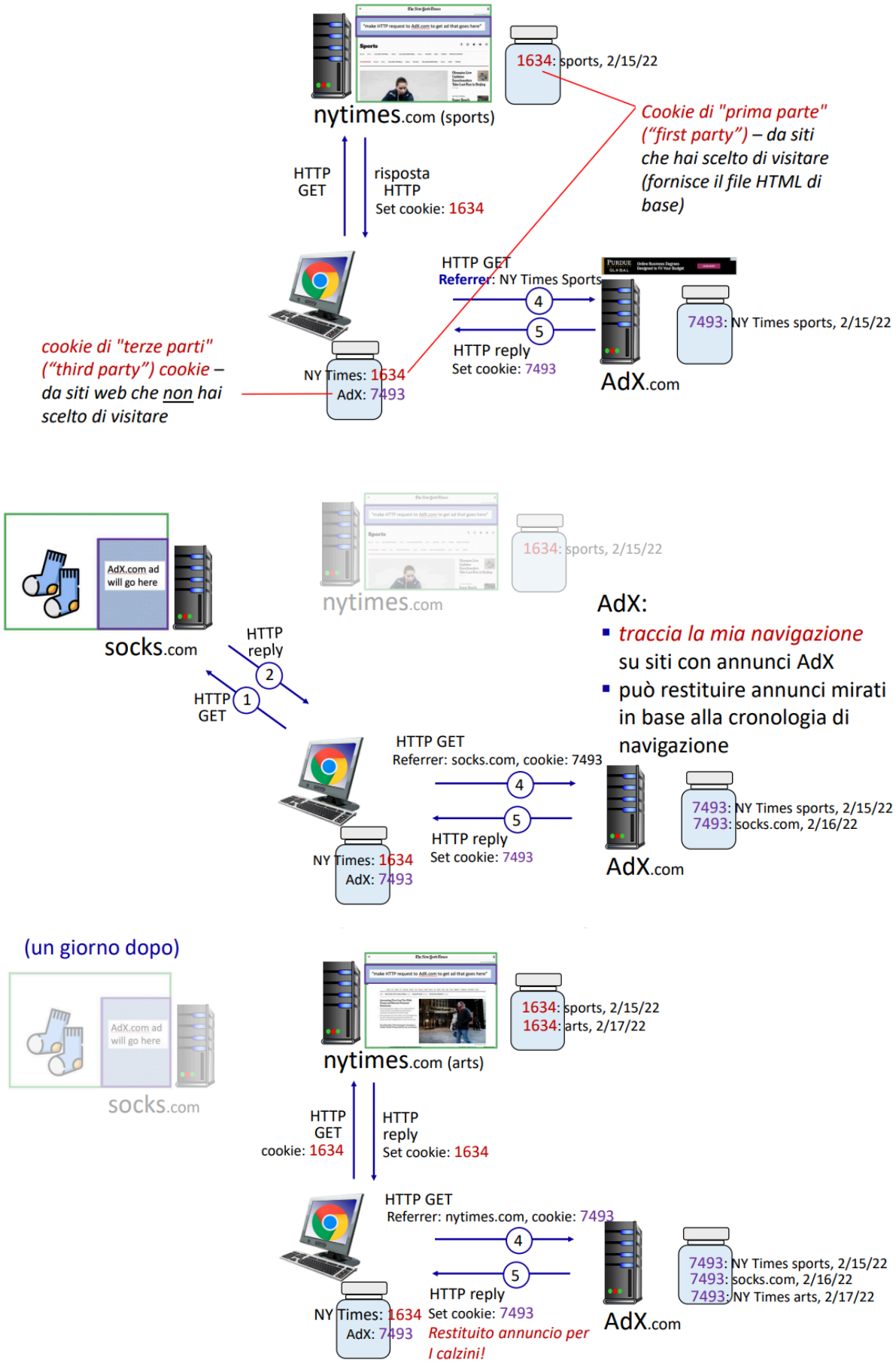
- Susan visita per la prima volta un sito di e-commerce.
- Il server genera un ID univoco e crea una voce nel database associata a quell'ID.
- La risposta HTTP include l'intestazione "Set-Cookie" con l'ID univoco.
- Il browser di Susan salva il cookie sul suo dispositivo.
- Le successive richieste di Susan al sito includeranno l'ID univoco nell'intestazione "Cookie".




# I cookie: usi, gestione e implicazioni sulla privacy

**Usi dei cookie:**

- **Autorizzazione:** identificare e autenticare l'utente.
- **Carrello degli acquisti:** memorizzare i prodotti selezionati.
- **Raccomandazioni:** personalizzare l'esperienza utente con suggerimenti mirati.
- **Stato della sessione:** mantenere informazioni sulla sessione di navigazione (es. email web).


nytimes.com (sports)
1634: sports, 2/15/22
Cookie di "prima parte" ("first party") – da siti che hai scelto di visitare (fornisce il file HTML di base)
HTTP GET
risposta HTTP
Set cookie: 1634
HTTP GET
Referrer: NY Times Sports
4
5
HTTP reply
Set cookie: 7493
7493: NY Times sports, 2/15/22
AdX.com
NY Times: 1634
AdX: 7493
cookie di "terze parti" ("third party") cookie – da siti web che non hai scelto di visitare
AdX.com ad will go here
socks.com
HTTP reply
2
HTTP GET
1
nytimes.com
1634: sports, 2/15/22
AdX:
traccia la mia navigazione su siti con annunci AdX
può restituire annunci mirati in base alla cronologia di navigazione
HTTP GET
Referrer: socks.com, cookie: 7493
4
5
HTTP reply
Set cookie: 7493
7493: NY Times sports, 2/15/22
7493: socks.com, 2/16/22
AdX.com
NY Times: 1634
AdX: 7493
(un giorno dopo)
AdX.com ad will go here
socks.com
nytimes.com (arts)
1634: sports, 2/15/22
1634: arts, 2/17/22
HTTP GET
cookie: 1634
HTTP reply
Set cookie: 1634
HTTP GET
Referrer: nytimes.com, cookie: 7493
4
5
HTTP reply
Set cookie: 7493
Restituito annuncio per I calzini!
7493: NY Times sports, 2/15/22
7493: socks.com, 2/16/22
7493: NY Times arts, 2/17/22
AdX.com
NY Times: 1634
AdX: 7493

**Come i cookie mantengono lo stato:**

- **Presso gli endpoint del protocollo:** memorizzando informazioni sul server e sul client.
- **Nei messaggi:** trasportando informazioni all'interno dei messaggi HTTP.

**Cookie e privacy:**

- I cookie possono essere utilizzati per raccogliere informazioni sulle abitudini di navigazione degli utenti.
- I cookie persistenti di terze parti (cookie di tracciamento) permettono di seguire un utente su diversi siti web.
- Il tracciamento può avvenire in modo invisibile all'utente.

**Gestione del tracciamento tramite cookie:**

- Disattivazione predefinita nei browser Firefox e Safari.
- Eliminazione graduale dei cookie di terze parti in Chrome:
    - 1% degli utenti a partire da Gennaio 2024.
    - Estensione a tutti gli utenti nel terzo trimestre del 2024.

**GDPR e cookie:**

- I cookie che identificano un individuo sono considerati dati personali.
- Sono quindi soggetti alla normativa GDPR sulla protezione dei dati personali.

# Web cache: prestazioni e benefici

**Obiettivo:** Soddisfare le richieste del client senza coinvolgere il server d'origine.

**Funzionamento:**

- L'utente configura il browser per utilizzare una web cache (locale).
- Il browser invia tutte le richieste HTTP alla cache.
- Se l'oggetto è presente nella cache:
    - La cache lo fornisce al client.
- Altrimenti:
    - La cache richiede l'oggetto al server d'origine.
    - Memorizza ("cache") l'oggetto ricevuto.
    - Lo restituisce al client.


**Web cache (server proxy):**

- La cache opera come client (per il server d'origine) e come server (per il client originale).
- Il server comunica alla cache la cache consentita dell'oggetto nell'intestazione della risposta: `Cache-Control: max-age=<seconds>` `Cache-Control: no-cache`

**Perché il web caching?**

- **Riduce i tempi di risposta alle richieste dei client:**
  - La cache è più vicina ai client.
- **Riduce il traffico sul collegamento di accesso a Internet istituzionale:**
  - Internet è ricca di cache.
- **Consente ai provider "scadenti" di fornire dati con efficacia.**

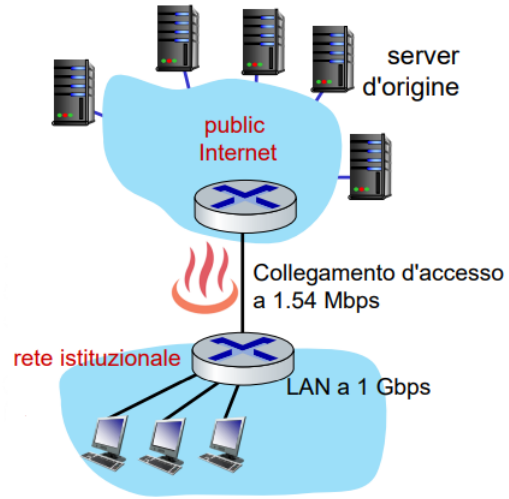
**Esempio di caching:**

**Scenario:**

- Velocità collegamento d'accesso: 1.54 Mbps
- RTT dal router istituzionale al server: 2 s
- Dimensione di un oggetto: 100K bits
- Frequenza media di richieste dai browser istituzionali al server d'origine: 15/s
- Velocità media di trasmissione dei dati ai browser: 1.50 Mbps

**Prestazioni:**

- Utilizzazione del collegamento d'accesso = 0.97
- Utilizzazione della LAN: 0.0015
- End-end delay = ritardo di Internet + ritardo del collegamento d'accesso + ritardo della LAN = 2 s + minuti + microsecondi




**Opzione 1: Collegamento d'accesso più veloce**

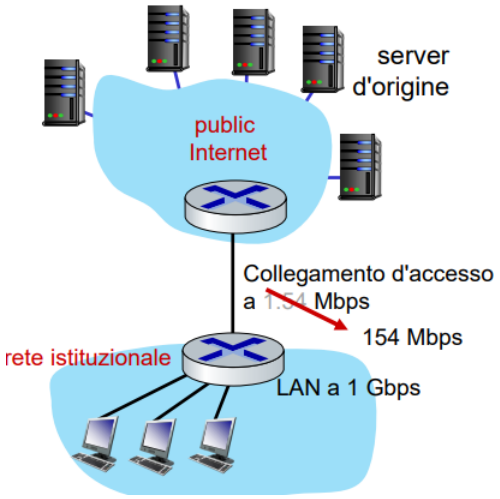
**Scenario:**

- Velocità collegamento d'accesso: 154 Mbps
- RTT dal router istituzionale al server: 2 s
- Dimensione di un oggetto: 100K bits
- Frequenza media di richieste dai browser istituzionali al server d'origine: 15/s
- Velocità media di trasmissione dei dati ai browser: 1.50 Mbps

**Prestazioni:**

- Utilizzazione del collegamento d'accesso = 0.0097
- Utilizzazione della LAN: 0.0015
- End-end delay = ritardo di Internet + ritardo del collegamento d'accesso + ritardo della LAN = 2 s + msecs + microsecondi

**Costo:** Collegamento d'accesso più veloce (costoso!)



**Opzione 2: Installare un web cache:**

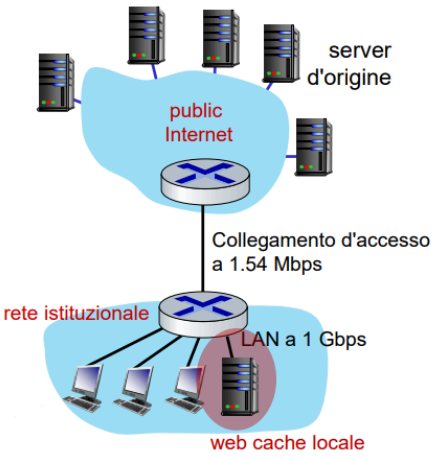
**Scenario:**

- Velocità collegamento d'accesso: 1.54 Mbps
- RTT dal router istituzionale al server: 2 s
- Dimensione di un oggetto: 100K bits
- Frequenza media di richieste dai browser istituzionali al server d'origine: 15/s
- Velocità media di trasmissione dei dati ai browser: 1.50 Mbps

**Costo:** Web cache (economica!)

**Prestazioni:**

- Utilizzazione LAN: ?
- Utilizzazione del link di accesso: ?
- Ritardo end-end medio: ?




**Calcolo dell'utilizzo del collegamento di accesso e del ritardo end-end con la cache:**

Supponiamo una percentuale di successo (hit rate) pari a 0.4:

- Il 40% delle richieste sarà soddisfatto dalla cache, con ritardo basso (msec).
- Il 60% delle richieste sarà soddisfatto dal server d'origine.

**Tasso di trasmissione sul collegamento d'accesso:**

0.6 * 1.50 Mbps = 0.9 Mbps

**Utilizzazione collegamento d'accesso:**

0.9/1.54 = 0.58

**Ritardo end-end medio:**

0.6 * (ritardo dai server d'origine) + 0.4 * (ritardo quando richiesta soddisfatta dalla cache) = 0.6 (2.01) + 0.4 (~msecs) = ~ 1.2 secs

**Conclusione:**

Il ritardo medio end-end con la web cache è inferiore rispetto a quello con un collegamento a 154 Mbps, con un costo significativamente inferiore.

### Vantaggi della web cache:

- **Riduce i tempi di risposta alle richieste dei client.**
- **Riduce il traffico sul collegamento di accesso a Internet.**
- **Riduce il carico sul server d'origine.**
- **Migliora l'affidabilità e la scalabilità del servizio web.**
- **Consente di fornire contenuti con restrizioni geografiche.**
- **Può essere utilizzata per la sicurezza e il filtraggio dei contenuti.**

**Svantaggi della web cache:**

- **Richiede un investimento iniziale in hardware e software.**
- **Può essere complessa da configurare e gestire.**
- **Può introdurre un ritardo aggiuntivo per le richieste che non sono presenti nella cache.**
- **Può essere vulnerabile ad attacchi informatici.**

**Esistono diversi tipi di web cache:**

- **Cache locali:** memorizzate sul dispositivo dell'utente.
- **Cache proxy:** memorizzate su un server proxy.
- **Cache di rete:** memorizzate su un dispositivo di rete.

# GET condizionale

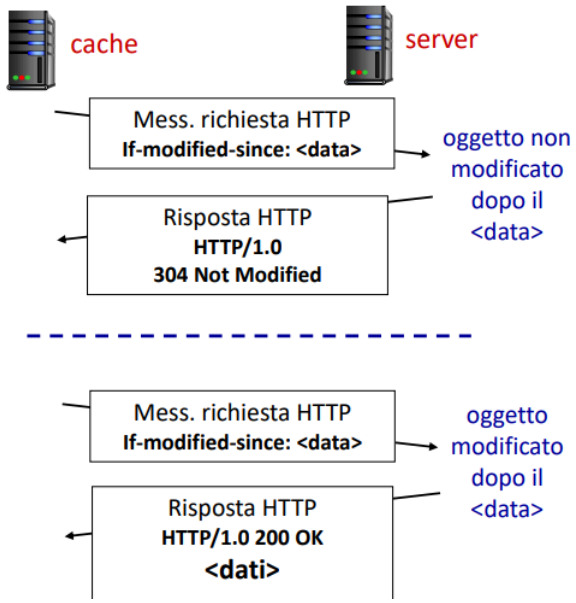
**Obiettivo:** Non inviare un oggetto se la cache ha una copia aggiornata dell'oggetto.

**Vantaggi:**

- Nessun ritardo di trasmissione dell'oggetto.
- Nessun uso delle risorse di rete per la trasmissione dell'oggetto.

**Funzionamento:**

- Il client specifica la data della copia dell'oggetto nella richiesta HTTP usando l'intestazione `If-Modified-Since`.
- Il server verifica se la copia nella cache è aggiornata.
- Se la copia è aggiornata, il server risponde con `HTTP/1.0 304 Not Modified` e non invia l'oggetto.
- Se la copia non è aggiornata, il server invia l'oggetto completo.




**Nota sul caching:**

Il caching può essere effettuato da:

- **Web cache:** un proxy speciale a cui il browser invia le richieste invece di indirizzarle al server d'origine.
- **Browser stesso:** conserva una copia degli oggetti richiesti in precedenza.

In entrambi i casi, è importante prestare attenzione al problema dell'aggiornamento degli oggetti:

- Intestazione `Cache-Control`.
- GET condizionale.

# HTTP/2

**Obiettivo principale:** Diminuzione del ritardo nelle richieste HTTP a più oggetti.

**Problemi con HTTP/1.1:**

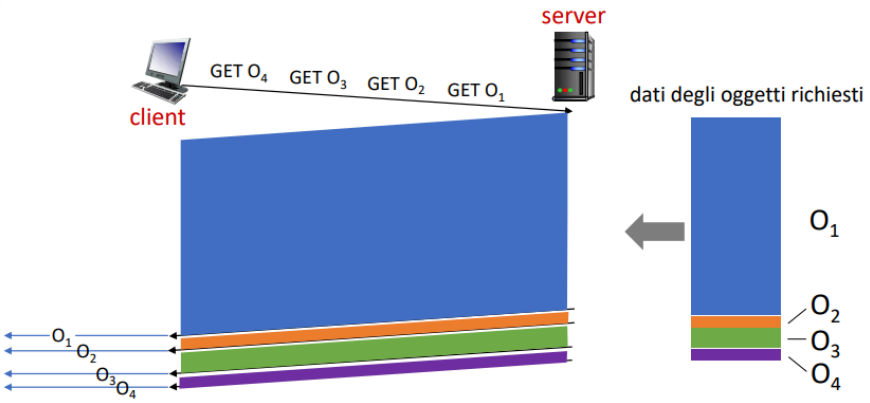
- **Pipeline di GET multiple su una singola connessione TCP:** il server risponde in ordine (FCFS) alle richieste GET.
- **Oggetti piccoli possono dover aspettare per la trasmissione (blocco HOL) dietro a uno o più oggetti grandi.**
- **Il recupero delle perdite (ritrasmissione dei segmenti TCP persi) blocca la trasmissione degli oggetti.**

**Miglioramenti in HTTP/2:**

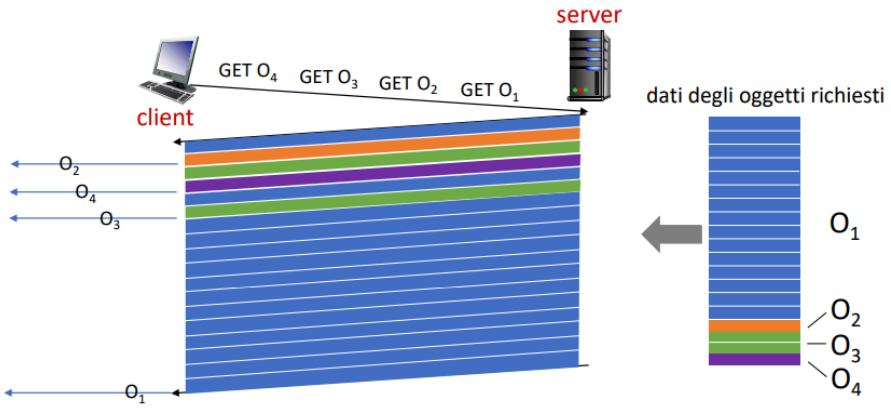
- **Maggiore flessibilità del server nell'invio di oggetti al client.**
- **Ordine di trasmissione degli oggetti basato su una priorità degli oggetti specificata dal client (non necessariamente FCFS).**
- **Invio push al client di oggetti aggiuntivi, senza che il client li abbia richiesti.**
- **Dividere gli oggetti in frame e intervallare i frame per mitigare il blocco HOL.**

**Esempio di mitigazione del blocco HOL:**

- **HTTP/1.1:** Il client richiede 1 oggetto grande (es., file video) e 3 oggetti più piccoli.
- **Oggetti consegnati nell'ordine in cui sono stati richiesti:** O2, O3, O4 aspettano dietro O1.



- **Mitigazione HTTP/2:**
  - Oggetti divisi in frame.
  - Trasmissione dei frame interlacciata.
  - O2, O3, O4 consegnati rapidamente.
  - O1 leggermente in ritardo.




# Da HTTP/2 a HTTP/3

**HTTP/2 su una singola connessione TCP:**

- Il recupero dalla perdita di pacchetti blocca comunque tutte le trasmissioni di oggetti.
- Come in HTTP 1.1, i browser sono incentivati ad aprire più connessioni TCP parallele per ridurre lo stallo e aumentare il throughput complessivo.
- Nessuna sicurezza su una connessione TCP semplice.

**HTTP/3:**

- Aggiunge sicurezza, controllo di errore e congestione per oggetto (più pipelining) su UDP.
- Ulteriori informazioni su HTTP/3 trattando il livello di trasporto.

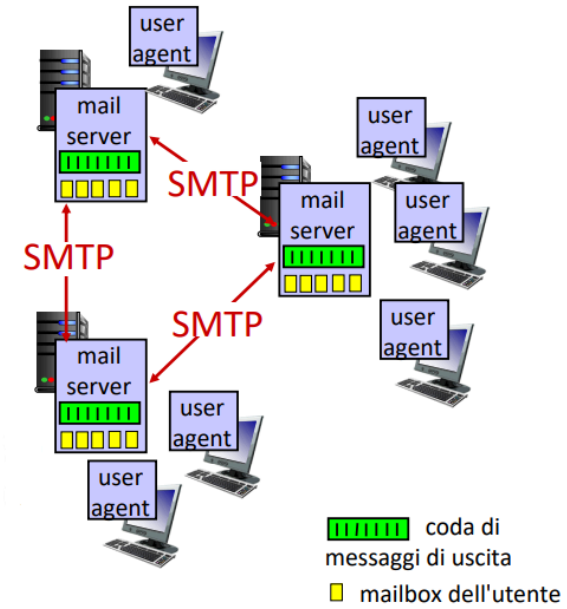
**In sintesi:**

- Il GET condizionale aiuta a ridurre il traffico di rete e il ritardo di caricamento per gli oggetti già memorizzati nella cache.
- HTTP/2 migliora le prestazioni di HTTP/1.1 riducendo il blocco HOL e aumentando la flessibilità del server.
- HTTP/3 aggiunge sicurezza e controllo di errore a HTTP/2 su UDP.

# E-mail: componenti e protocolli

**Componenti principali:**

- **User Agent (agente utente):** detto anche "mail reader", serve per comporre, modificare e leggere i messaggi. Esempi: Outlook, client di posta dell'iPhone. I messaggi in uscita o in arrivo sono memorizzati sul server.
- **Mail server (server di posta):**
  - **Mailbox (casella di posta):** contiene i messaggi in arrivo per l'utente.
  - **Coda di messaggi:** memorizza i messaggi da trasmettere.
- **Simple Mail Transfer Protocol (SMTP):** protocollo utilizzato per inviare messaggi email tra mail server.




**User Agent:**

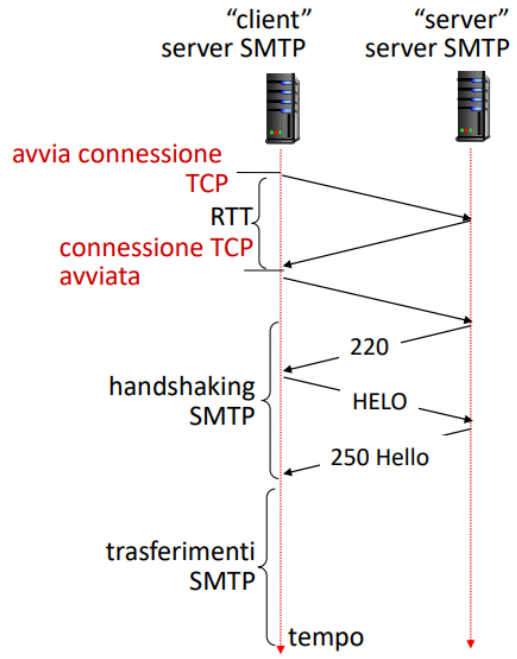
- Invia e riceve messaggi dal mail server.
- Memorizza i messaggi in uscita e in arrivo sul server.

**Mail server:**

- Gestisce le caselle di posta degli utenti.
- Invia e riceve messaggi tramite SMTP.
- Memorizza i messaggi in coda in attesa di invio.

**SMTP:**

- Usa TCP per trasferire messaggi in modo affidabile.
- Porta 25: utilizzata per SMTP.
- Trasferimento diretto: dal server trasmittente al server ricevente.




**Fasi del trasferimento SMTP:**

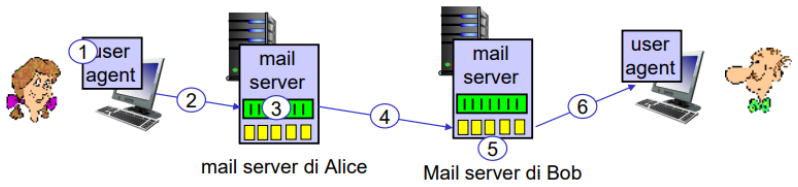
1. **Handshaking (saluto):** identificazione del server trasmittente e ricevente.
2. **Trasferimento dei messaggi:** invio del messaggio dal client al server.
3. **Chiusura:** termine della connessione.

**Interazione comando/risposta:**

- Comandi: testo ASCII a 7 bit.
- Risposta: codice di stato e espressione.

**Scenario di invio di un'e-mail:**

1. Alice compone il messaggio con il suo user agent.
2. Lo user agent di Alice invia il messaggio al server di posta di Alice.
3. Il server di posta di Alice mette il messaggio in coda.
4. Il client SMTP del server di Alice apre una connessione TCP con il server di posta di Bob.
5. Il client SMTP invia il messaggio di Alice al server di posta di Bob.
6. Il server di posta di Bob mette il messaggio nella casella di posta di Bob.
7. Bob legge il messaggio con il suo user agent.



**Note finali su SMTP:**

- Confronto con HTTP:
  - HTTP: client pull.
  - SMTP: client push.
- Entrambi hanno un'interazione comando/risposta in ASCII e codici di stato.
- HTTP: ogni oggetto è incapsulato nel suo messaggio di risposta.
- SMTP: più oggetti vengono trasmessi in un unico messaggio.
- SMTP usa connessioni persistenti.
- SMTP richiede che il messaggio (intestazione e corpo) sia in formato ASCII a 7 bit.
- Il server SMTP usa CRLF.CRLF per determinare la fine del messaggio.

**Formato dei messaggi di posta elettronica:**

- SMTP: definito nell'RFC 5321.
- Sintassi dei messaggi: definita nell'RFC 2822.

**Righe di intestazione:**

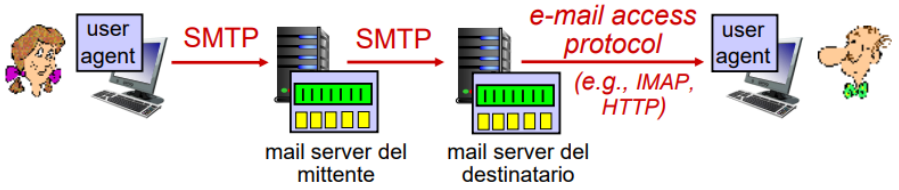
- To/A:
- From/Da:
- Subject/Oggetto:

**Corpo:**

- Il "messaggio".
- Solo caratteri ASCII.

**Protocolli di accesso alla posta:**

- **SMTP:** consegna/memorizzazione sul server del destinatario.
- **IMAP (Internet Mail Access Protocol):**
  - Messaggi memorizzati sul server.
  - Consente di recuperare, cancellare e archiviare i messaggi.
- **HTTP:** Gmail, Hotmail, Yahoo!Mail, etc.
  - Interfaccia web sopra a SMTP (per l'invio) e IMAP (o POP) per il recupero delle email.



# Risoluzione dei nomi: File hosts e DNS

**Problema:**

- A livello applicativo, ci sono molti identificatori (nomi, codici fiscali, numeri di carta d'identità) per persone, host e router di Internet.
- A livello di rete, è necessario utilizzare indirizzi IP (32 bit) per indirizzare i datagrammi.
- Gli esseri umani preferiscono utilizzare nomi come "cs.umass.edu" invece di indirizzi IP.

**File hosts:**

- Soluzione locale che associa un indirizzo IP a uno o più hostname.
- Esempio:

```
185.300.10.1 host1

185.300.10.2 host2

merlin 185.300.10.3 host3

arthur king 185.300.10.4 timeserver
```

- Vantaggi:
  - Semplice da configurare e utilizzare.
- Svantaggi:
  - Non scalabile per grandi reti.
  - Richiede la manutenzione manuale su ogni nodo.
  - Può creare conflitti se diversi nodi hanno file hosts con lo stesso nome.

**DNS (Domain Name System):**

- Sistema di database distribuito implementato in una gerarchia di name server.
- Protocollo a livello di applicazione che consente la traduzione di nomi in indirizzi IP.
- Funzionalità critica di Internet, implementata come protocollo applicativo.

**Servizi DNS:**

- Traduzione di hostname in indirizzi IP.
- Alias di host: nome canonico e alias.
- Alias del server di posta.
- Distribuzione del carico: più indirizzi IP corrispondono a un solo nome.

**Perché non centralizzare il DNS?**

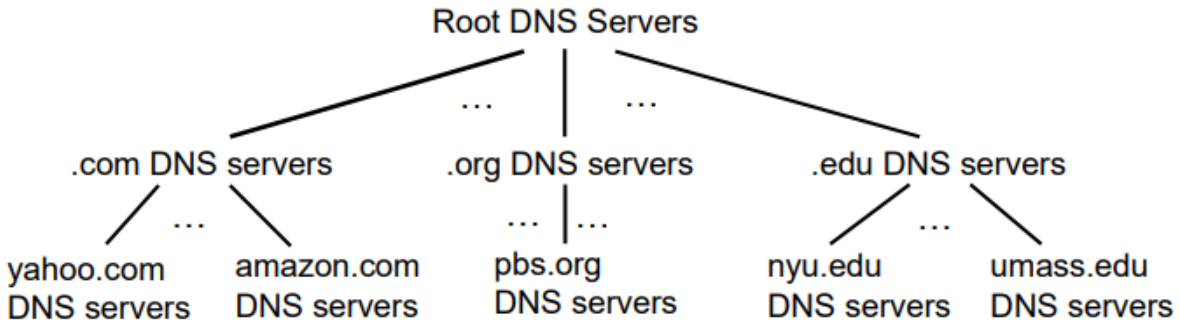
- Single point of failure.
- Volume di traffico elevato.
- Database centralizzato distante.
- Manutenzione complessa.
- Non scala!

**DNS come database distribuito:**

- Gestisce miliardi di record.
- Molte più letture che scritture.
- Quasi tutte le transazioni Internet interagiscono con il DNS.
- Decentralizzato organizzativamente e fisicamente.
- Affidabilità e sicurezza elevate.

**Gerarchia DNS:**

- Root (radice).
- Top Level Domain (TLD).
- Authoritative (server autoritativi).



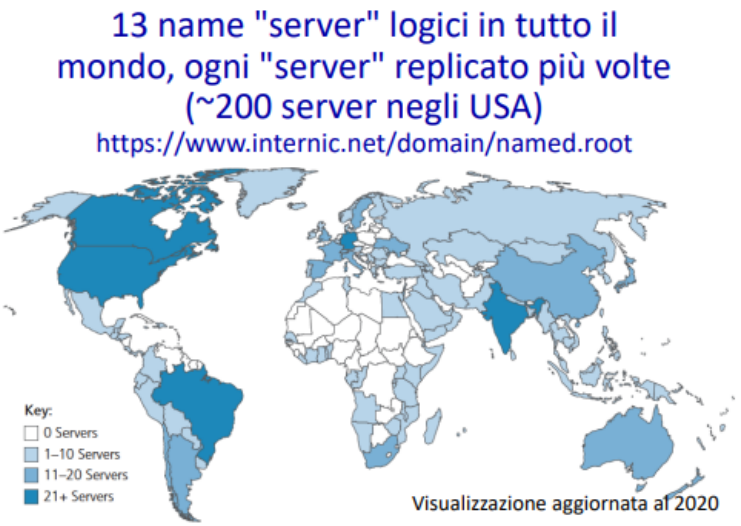

**Risoluzione di un nome DNS:**

- Il client vuole l'indirizzo IP di www.amazon.com.
- Interroga il root server per trovare il TLD server per .com.
- Interroga il TLD server .com per ottenere il server autoritativo per amazon.com.
- Interroga il server autoritativo per amazon.com per ottenere l'indirizzo IP di www.amazon.com.

**Root name server:**

- Fornisce gli indirizzi IP dei TLD server.
- Funzione incredibilmente importante di Internet.

- DNSSEC offre sicurezza e integrità dei messaggi.
- ICANN gestisce il root DNS domain.




Il 20/03/2023 ci sono 1813 istanze gestite da 12 operator, coordinate dallo IANA (fonte: https://root-servers.org/)

**Top-Level Domain (TLD) e server autoritativi:**

- Gestiscono i domini .com, .org, .net, .edu, .aero, .jobs, .museums e TLD locali di alto livello.
- Esempio: Network Solutions gestisce i server TLD per i domini .com e .net.
- Server DNS autoritativo: forniscono i mapping ufficiali da hostname a IP per gli host dell'organizzazione.

# Name server DNS locali

- Quando un host effettua una richiesta DNS, la query viene inviata al suo **server DNS locale** (che funge da name server predefinito).
- Il server DNS locale risponde alla query:
  - Dalla sua cache locale di coppie nome-indirizzo (che potrebbe non essere aggiornata!).
  - Inoltrando la richiesta alla gerarchia DNS per la risoluzione.
- Ogni ISP ha un proprio server DNS locale.
- Per trovare il tuo server DNS locale:
  - MacOS: `% scutil --dns`
  - Windows: `>ipconfig /all`
- Il server DNS locale non appartiene strettamente alla gerarchia dei server DNS.

# Interrogazione DNS

**Esempio:** L'host `engineering.nyu.edu` vuole l'indirizzo IP di `gaia.cs.umass.edu`.

**Interrogazione iterativa:**

- Il server contattato risponde con il nome del server da contattare.
- "Io non conosco questo nome, ma puoi chiederlo a questo server".

**Interrogazione ricorsiva:**

- Affida il compito di tradurre il nome al server contattato.
- Carico pesante ai livelli superiori della gerarchia?

# Caching e aggiornamento dei record DNS

- Una volta che un (qualsiasi) name server impara la mappatura, la mette nella cache e la restituisce immediatamente in risposta a una query.
- Il caching migliora i tempi di risposta.
- Le voci della cache vanno in timeout (scompaiono) dopo un certo tempo (TTL).
- I server TLD sono in genere memorizzati nella cache dei server dei nomi locali.
- Le voci nella cache potrebbero essere obsolete.
- Se l'host con nome cambia il suo indirizzo IP, potrebbe non essere conosciuto su Internet fino alla scadenza di tutti i TTL!
- Traduzione nome-indirizzo best-effort!

# Record DNS

- Il DNS è un database distribuito che memorizza 7 tipi di record di risorsa (RR).
- Formato RR: (nome, valore, tipo, ttl).

**Tipi di record:**

- **A:**
  - `name` è l'hostname.
  - `value` è l'indirizzo IP.
- **NS:**
  - `name` è il dominio (ad esempio, `foo.com`).
  - `value` è l'hostname dell'authoritative name server per questo dominio.
- **CNAME:**
  - `name` è il nome alias di qualche nome "canonico" (nome vero).

    - `value` è il nome canonico.

- **MX:**
    - `value` è il nome del server di posta associato a `name`.

# Messaggi DNS

- Domande (query) e messaggi di risposta (reply), entrambi con lo stesso formato.

**Intestazione del messaggio:**

- Identificazione: numero di 16 bit per la domanda; la risposta alla domanda usa lo stesso numero.
- Flag:
    - Domanda o risposta.
    - Richiesta di ricorsione.
    - Ricorsione disponibile.
    - DNS server autoritativo.

# Inserimento di record nel database DNS

**Esempio:** Abbiamo appena avviato la nuova società "Network Utopia".

1. Registriamo il nome `networkuptopia.com` presso il DNS registrar (ad esempio, Network Solutions, oppure un altro dei concorrenti accreditati dall'ICANN).
    - Forniamo al registrar il nome e gli indirizzi IP degli authoritative name server (primario e secondario).
    - Il registrar inserisce due RR nel TLD server .com:
        - (`networkutopia.com`, `dns1.networkutopia.com`, `NS`).
        - (`dns1.networkutopia.com`, `212.212.212.1`, `A`).
2. Inseriamo localmente nell'authoritative server:
    - Un record A per `www.networkuptopia.com`.
    - Un record MX per `networkutopia.com`.

# Sicurezza del DNS

**Attacchi DDoS (distributed denial-of-service)**

- **Bombardamento di traffico ai root server:**
    - Finora senza successo.
    - Filtraggio del traffico in atto.
- **Server DNS locali come bersaglio:**

    - Memorizzano nella cache gli indirizzi IP dei server TLD.
    - Attacco ai server TLD potenzialmente più pericoloso.

**Attacco di spoofing:**

- Intercettazione delle query DNS e restituzione di risposte fasulle.
- Avvelenamento della cache DNS.

**RFC 4033: DNSSEC**

- Servizi di autenticazione per la sicurezza del DNS.

# Architettura Peer-to-peer (P2P)

**Caratteristiche:**

- **Nessun server sempre attivo.**
- **I peer (sistemi periferici arbitrari) comunicano direttamente.**
- **I peer richiedono e forniscono servizi tra loro.**

**Vantaggi:**

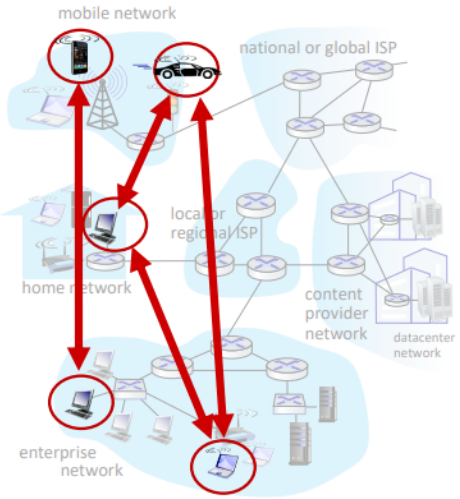
- **Scalabilità intrinseca: nuovi peer portano nuove capacità e richieste di servizio.**

**Svantaggi:**

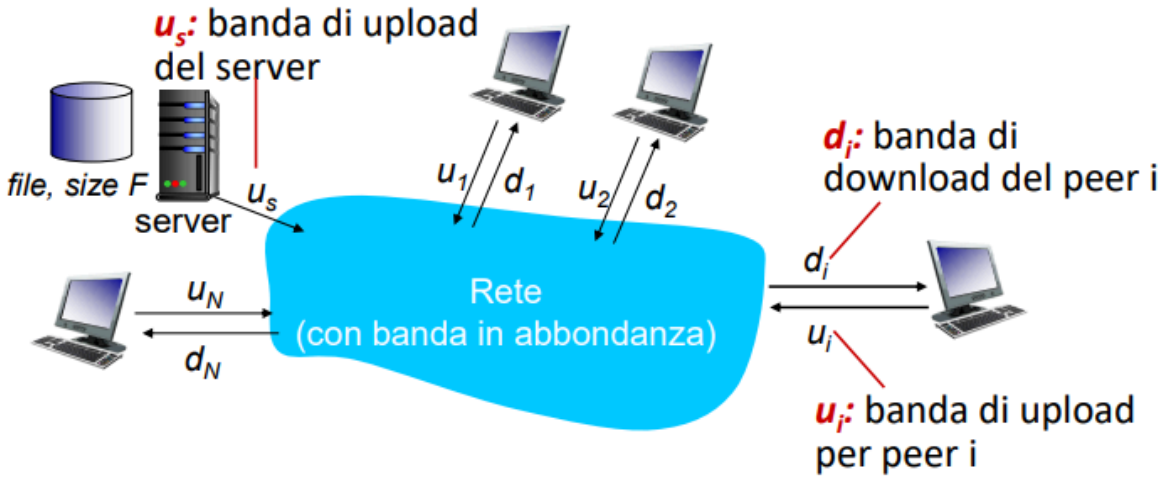
- **Gestione complessa: i peer sono connessi a intermittenza e cambiano indirizzo IP.**

**Esempi:**

- **Condivisione di file (BitTorrent).**
- **Streaming (KanKan).**
- **VoIP (Skype).**




# Distribuzione di file: confronto tra client-server e P2P




**Client-server:**

- **Tempo di trasmissione per il server:**
  - **Trasmissione sequenziale di N copie del file:**
  - **Tempo per una copia: F/us**
  - **Tempo per N copie: NF/us**
- **Tempo di download per il client:**
  - **Minimo = banda di download più bassa (dmin)**
  - **Tempo di download per il client più lento: F/dmin**

*Tempo per distribuire F a N client usando l'approccio client-server*

$$D_{c\text{-}s} \geq \max\{NF/u_s, F/d_{min}\}$$

aumenta linearmente in N

**P2P:**

- **Tempo di trasmissione per il server:**
  - **Trasmissione di una sola copia del file:**
  - **Tempo per una copia: F/us**
- **Tempo di download per il client:**
  - **Minimo = F/dmin**
  - **Tempo di download per il client più lento: F/dmin**
- **Capacità di upload aggregata:**
  - **Limitata da us + ∑(ui)**

*Tempo per distribuire F a N client usando l'approccio P2P*

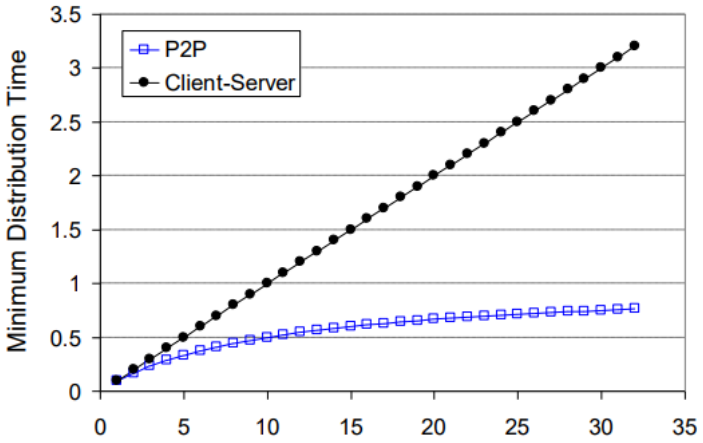
$$D_{P2P} \geq max\{F/u_s, F/d_{min}, NF/(u_s + \Sigma u_i)\}$$

aumenta linearmente in N …
… ma anche questo, dato che ogni peer porta con sé la capacità di servizio

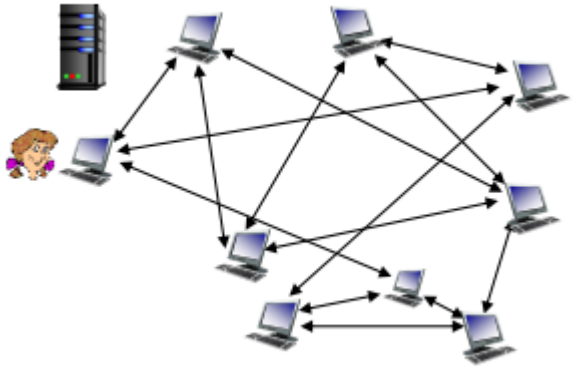
**Conclusione:**

- **P2P offre una migliore scalabilità rispetto al client-server.**
- **La gestione dei peer connessi a intermittenza è un problema complesso in P2P.**

banda di upload del client = $u$, $F/u$ = 1 ora, $u_s = 10u$, $d_{min} \geq u_s$




# Distribuzione di file P2P: BitTorrent



**Caratteristiche:**

- **File diviso in chunk (parti), in genere di 256 kB.**
- **I peer nel torrent inviano/ricevono chunk del file.**

**Tracker:**

- **Tiene traccia dei peer che partecipano al torrent.**
- **Fornisce ai nuovi peer un elenco di altri peer con cui connettersi.**

**Torrent:**

- **Gruppo di peer che partecipano alla distribuzione di un file.**

**Comportamento di un peer:**

- **Un nuovo peer non ha chunk del file, ma li accumulerà nel tempo da altri peer.**
- **Si registra con un tracker e ottiene la lista di un sottoinsieme di peer.**
- **Stabilisce una connessione con un sottoinsieme di questi peer ("vicini").**
- **Informa periodicamente il tracker che è ancora nel torrent.**
- **Mentre scarica chunk, un peer invia i chunk già in suo possesso agli altri peer.**
- **Un peer può cambiare i peer con cui scambia i chunk.**
- **I peer possono andare e venire.**
- **Una volta che un peer ha acquisito l'intero file, può lasciare il torrent (egoisticamente) o rimanere (altruisticamente) come seeder.**

**Richiesta e invio di chunk di file:**

**Richiesta di chunk:**

- **In ogni momento, peer diversi hanno sottoinsiemi diversi di chunk.**

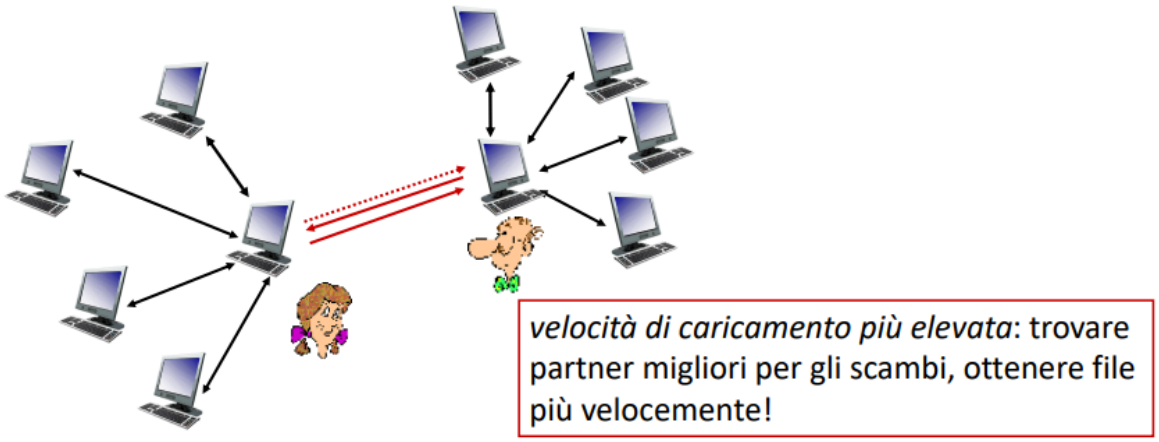
- **Alice chiede periodicamente ai peer vicini l'elenco dei chunk in loro possesso.**
- **Alice richiede ai peer i chunk mancanti, adottando la strategia del rarest first ("prima i più rari").**
- **Un peer appena entrato può chiedere un blocco in modo casuale.**
- **Quando sta per completare il file, può adottare la strategia end game.**

**Invio di chunk:**

- **Alice invia i chunk ai quattro peer vicini che attualmente le inviano i chunk alla velocità più alta.**
- **Altri peer sono detti choked ("soffocati" o "limitati").**
- **Alice rivaluta i primi 4 posti ogni 10 secondi.**
- **Ogni 30 secondi, Alice seleziona in modo casuale un vicino e inizia a inviare chunk.**
- **Questo peer è detto "optimistically unchoked" ("non limitato/soffocato in maniera ottimistica").**
- **Il nuovo peer scelto può entrare nella top 4.**

**Tit-for-tat:**

- **Strategia di reciprocità per l'invio di chunk.**
- **Alice aiuta i peer che la aiutano.**
- **Incentiva la cooperazione e massimizza la velocità di download.**




**Esempio:**

1. **Alice sceglie Bob come "optimistically unchoked".**
2. **Alice diventa uno dei primi quattro fornitori di Bob; Bob ricambia.**
3. **Bob diventa uno dei primi quattro fornitori di Alice.**

**Vantaggi:**

- **Velocità di download più elevate.**

- **Maggiore affidabilità.**
- **Scalabilità.**

**Svantaggi:**

- **Dipendenza dal tracker.**
- **Vulnerabilità a attacchi DDoS.**

# Streaming video e CDN: contesto

**Traffico video in streaming:**

- Grande consumatore di larghezza di banda Internet.
- Netflix, YouTube, Amazon Prime: 80% del traffico ISP residenziale (2020).

**Sfide:**

- Scala: come raggiungere ~1B di utenti?
- Eterogeneità: utenti con capacità diverse (cablati o mobili, con diverse larghezze di banda).

**Soluzione:**

- Infrastruttura distribuita a livello di applicazione.

# Contenuti multimediali: video

**Video:**

- Sequenza di immagini visualizzate a tasso costante.
- Esempio: 24 immagini al secondo.

**Immagine digitale:**

- Un array di pixel.
- Ogni pixel rappresentato da bit.

**Codifica:**

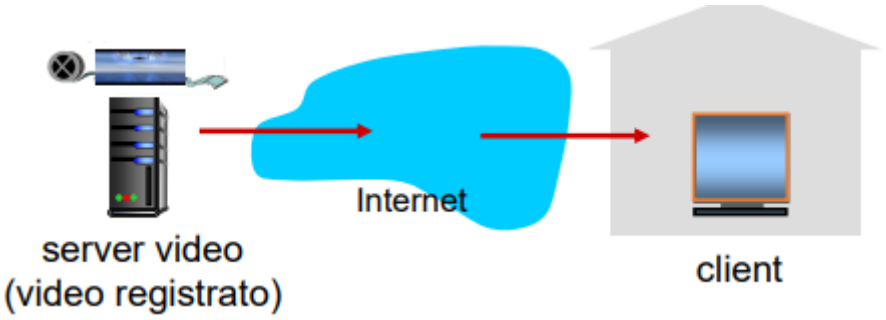
- Utilizzare la ridondanza all'interno e tra le immagini per ridurre il numero di bit utilizzati.
- Spaziale (all'interno di una data immagine).

- Temporale (da un'immagine all'altra).

**Video:**

- CBR (constant bit rate): bit rate costante.
- VBR (variable bit rate): bit rate cambia con la quantità di codifica spaziale e temporale.
- Esempio:
    - MPEG 1 (CD-ROM) 1.5 Mbps.
    - MPEG2 (DVD) 3-6 Mbps.
    - MPEG4 (spesso usato in Internet, 64Kbps – 12 Mbps).

## Streaming video di contenuti registrati

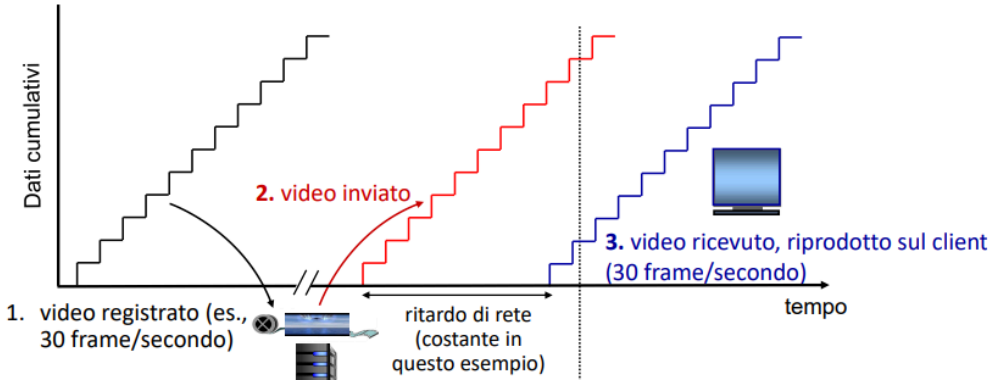



**Sfide principali:**

- La larghezza di banda da server a client varia nel tempo, con il variare dei livelli di congestione della rete.
- La perdita di pacchetti e i ritardi dovuti alla congestione ritardano la riproduzione o comportano una scarsa qualità video.

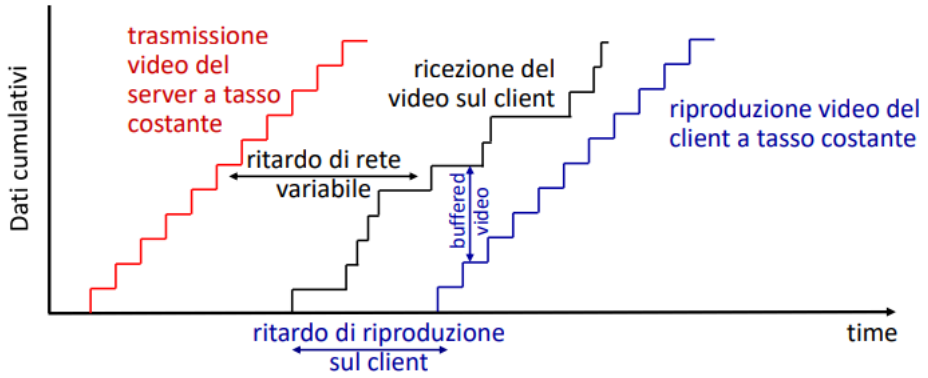
**Streaming:**

> In questo momento, il client sta riproducendo la parte iniziale del video, mentre il server sta ancora inviando la parte successiva del video.



**Vincolo di riproduzione continua:**

- Quando la riproduzione inizia, dovrebbe procedere secondo i tempi di registrazione originali.
- I ritardi di rete sono variabili (jitter), quindi avrà bisogno di un buffer lato client.




**Altre sfide:**

- Interattività del client: pausa, avanzamento veloce, riavvolgimento, salti attraverso il video.
- I pacchetti video possono essere persi e ritrasmessi.

# Buffering lato client e ritardo di riproduzione

- Compensare il ritardo aggiunto dalla rete e il jitter (variazione) del ritardo.

# Streaming multimediale: DASH (Dynamic, Adaptive Streaming over HTTP)

**Server:**

- Divide il file video in più chunk.
- Ogni chunk è codificata in più versioni, con bit rate differenti.
- Versioni diverse sono memorizzate in file diversi.
- I file sono replicati in vari nodi CDN.

**Manifest file:**

- Fornisce gli URL per i diversi chunk.

**Client:**

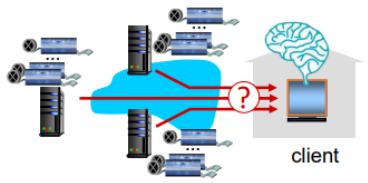
- Stima periodicamente la banda da server a client.
- Consultando il manifesto, richiede un chunk alla volta.
- Sceglie la versione con il bit rate più alto sostenibile data la larghezza di banda corrente.
- Può scegliere versioni con bit rate differenti in momenti diversi (a seconda della larghezza di banda disponibile in quel momento), e da server diversi.

## “Intelligenza” sul client

Il client determina:

- Quando richiedere un chunk (in modo che non si verifichi la starvation del buffer o l'overflow).
- Che encoding rate richiedere (qualità più alta quando c'è più larghezza di banda).
- Dove richiedere il chunk (può richiedere dal server che è "vicino" al client o ha banda larga).

**Streaming video = codifica + DASH + buffering di riproduzione.**




# Reti per la distribuzione di contenuti - Content distribution networks (CDNs)

**Sfida:**

- Come trasmettere contenuti in streaming (selezionati tra milioni di video) a centinaia di migliaia di utenti simultanei?

**Opzione 1: unico, enorme data center (Non scalabile).**

- Singolo punto di rottura (single point of failure).
- Punto di congestione della rete.
- percorso lungo (possibilmente congestionato) verso i clienti lontani.