Livello Applicazione

Quando si creano delle applicazioni moderne si sono prese dovute precauzioni nella costruzione nello stesso luogo di tutte le sue parti: ormai i dati di una singola applicazione viaggiano sulla rete e sono locate in zone diverse.

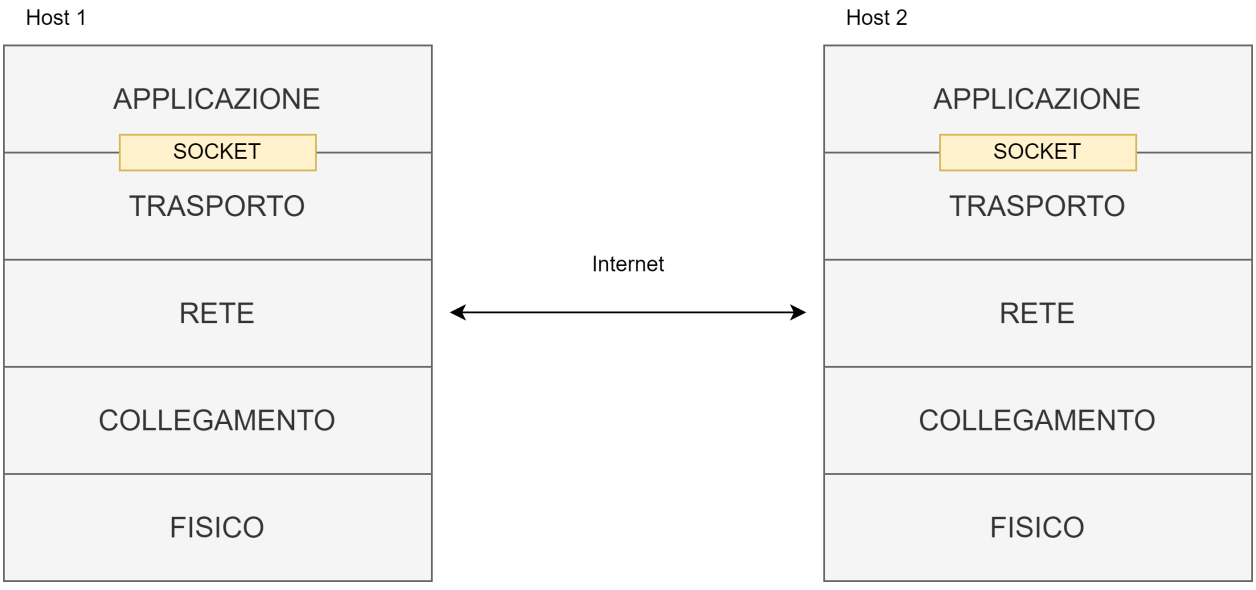
Ci sono due tipi di architettura:

- **Client-Server:** in questo caso abbiamo il client che esegue richieste e il server che risponde come YouTube.
- **Peer-2-Peer** (P2P): in questo caso non è delimitato il limite tra client e server, un host può essere entrambi come ad esempio su BitTorrent.

In generale ci sono due end-point che comunicano attraverso un protocollo che delimita il come deve essere impostato il messaggio e del suo contenuto.

Useremo i **socket (canali di comunicazione)** fondamentali per la comunicazione tra client e server, in laboratorio lavoreremo sul livello applicazione.

Dal punto di visto della programmazione sono delle strutture dati per poter comunicare attraverso i vari layer.



Ogni host in internet è identificato da un indirizzo IP, quando noi scriviamo un indirizzo come ad esempio `www.youtube.com` noi dobbiamo (o meglio il nostro computer) prima interrogare un server chiamato **DNS (Domain Name System)** il quale ci fornisce l'indirizzo IP in cui risiede il sito web.

Bisogna sempre legare ad ogni indirizzo una porta per formare il così detto **end-point** perchè a ogni porta viene connesso un servizio diverso.

L'indirizzo IP è formato da 32 bit, tradotti poi in decimale per trovarci nel risultato più classico: `192.168.0.1` esempio di indirizzo.

Il livello applicativo offre diverse certezze:

- **Integrità dei dati:** in ogni caso il livello invia tutti i dati, ma tollera le perdite.
- **Latenza ridotta:** offre una latenza minima ovvero che i pacchetti siano inviati con pochi ritardi.
- **Throughput**: ovvero quanti dati riescono ad essere inviati (a differenza della **Bandwidth** che è solo il valore nominale). Bisogna creare un algoritmo efficiente per poter non avere delle perdite.
- **Sicurezza:** si offre la sicurezza dei dati tra client e server.

Applicazione	Perdite	Throughput	Time Sensitive **(Delay)**
Trasferimento file	Nessuna	Elastico	No
E-mail	Nessuna	Elastico	No
Videochiamate	Tollerante	10kbps-5Mbps	Sì
Messaggio	Nessuna	Elastico	Sì e no
Giochi	Tollerante	Alcuni Mbps	Sì

Per time sensitive si intende quanto è necessario che venga consegnato in tempo il pacchetto: in un gioco è molto importante che i pacchetti vengano inviati in tempo mentre in una email no perchè tanto prima o poi arriva tutto, nei messaggi come su WhatsApp abbiamo l'invio con poi la spunta quindi possiamo arrabbiarci se non lo invia immediatamente ma è elastico.

Il livello di trasporto (che vedremo più avanti) serve a scegliere un come viene inviato un dato da una applicazione:

TCP

Il primo protocollo è TCP (**Trasmission Control Protocol**) il quale garantisce l'invio e la gestione del flusso di invio per non sovraccaricare intenzionalmente il ricevente, per non dover rinviare un pacchetto, a volte capita di avere un congestione di pacchetti e quindi il protocollo riduce l'invio. Non fornisce garanzie sui tempi di risposta, il throughput e sicurezza.

Questo protocollo si mette d'accordo nella handshake con il server sui dati per il loro invio.

UDP

Il secondo protocollo è UDP (**User Datagram Protocol**) non garantisce la ricezione, non vengono effettuati tutti i controlli che vengono fatti da TCP, viene quindi tollerata una perdita di pacchetti.

Ecco un esempio di applicazioni con relativi protocolli:

Applicazione	Protocollo Applicazione	Protocollo Trasporto
E-mail	SMTP	TCP
Videochiamate	SIP, RTP, proprietari	UPD
Terminale remoto	Telnet	TCP
Trasferimento di file	HTTP (es. YouTube)	TCP o UDP

Sul sito di ietf ovvero l'ente principale che definisce i protocolli, possiamo vedere com'è formato ad esempio SMTP:

```
Network Working Group                                          J. Klensin
Request for Comments: 2821                                      AT&T Labs
Obsoletes: 821, 974, 1869                                             Ap
Updates: 1123
Category: Standards Track


                        Simple Mail Transfer Protocol
```

```
Status of this Memo

   This document specifies an Internet standards track protocol
   Internet community, and requests discussion and suggestions f
   improvements.  Please refer to the current edition of the "In
   Official Protocol Standards" (STD 1) for the standardization
   and status of this protocol.  Distribution of this memo is un
......
```

RFC sta per “Request for Comments” ed è un formato standard per la pubblicazione di specifiche tecniche da parte di IETF (**Internet Engineering Task Force**).

La sicurezza TCP

All’inizio TCP e UPD non avevano controlli, non c’erano cifratura di alcun tipo, con l’avvento di internet si son dovute prendere delle precauzioni aggiungendo un layer al protocollo ovvero **TSL** (**Transport Layer Security**).

Provvede una cifratura che all’inizio non era necessaria con un nuovo layer tra applicazione e trasporto.

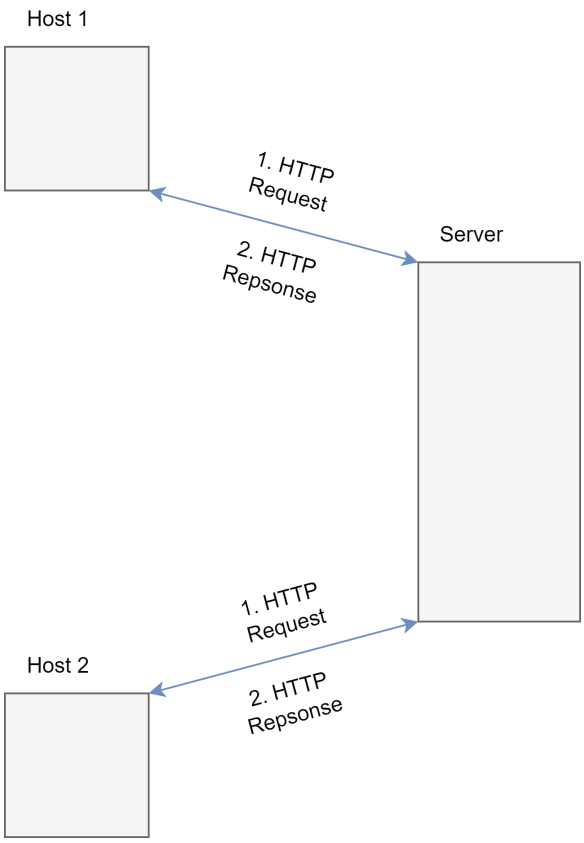
Il livello applicazione definisce: **tipo di messaggio scambiato** (richiesta o risposta), **sintassi** (com'è formato), **semantica** (significato dei campi), **regole** (come gestire i dati).

Web e HTTP

Per prima cosa dobbiamo capire come funziona, il web ovvero con gli URL (**Uniform Resource Locator**) e serve a indirizzare la risorsa in rete in maniera univoca.

Il protocollo **HTTP (Hypertext Trasfer Protocol)** funziona con richieste e risposte, il client così vede il contenuto della pagina HTML. Usa il TCP sulla porta 80 se usiamo la versione sicura viaggiamo sulla 443.

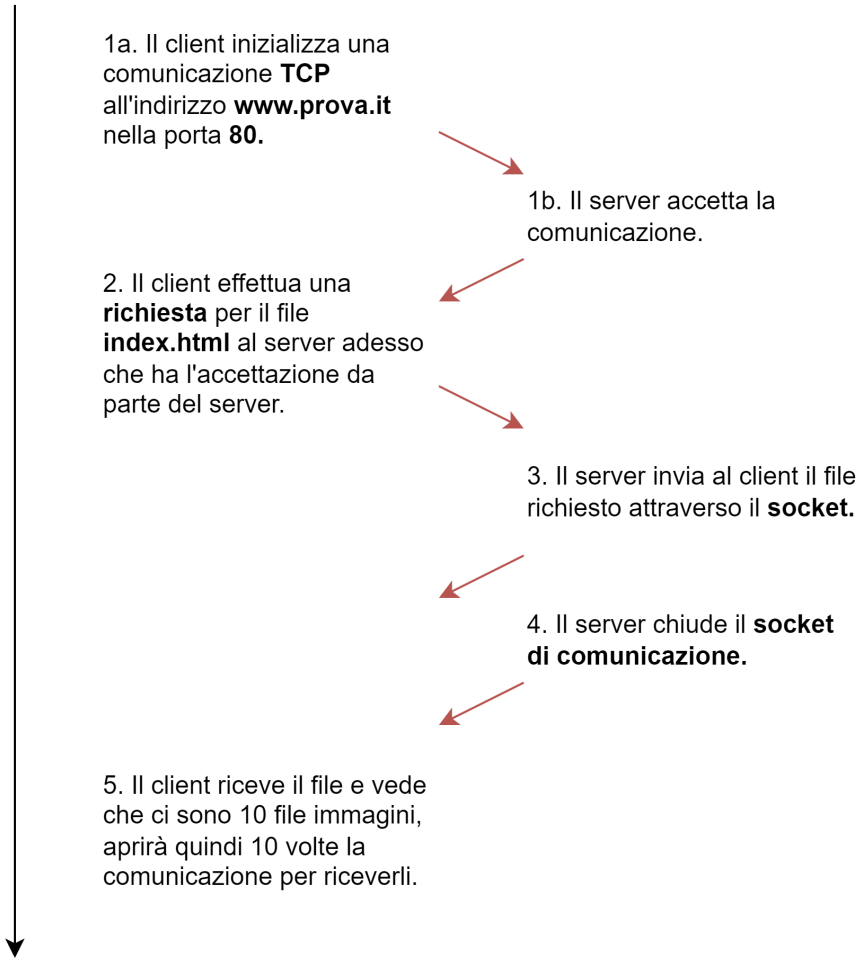
Per prima cosa si stabilisce la connessione con un "OK" da parte del server e appena avviene lo scambio si chiude la comunicazione.



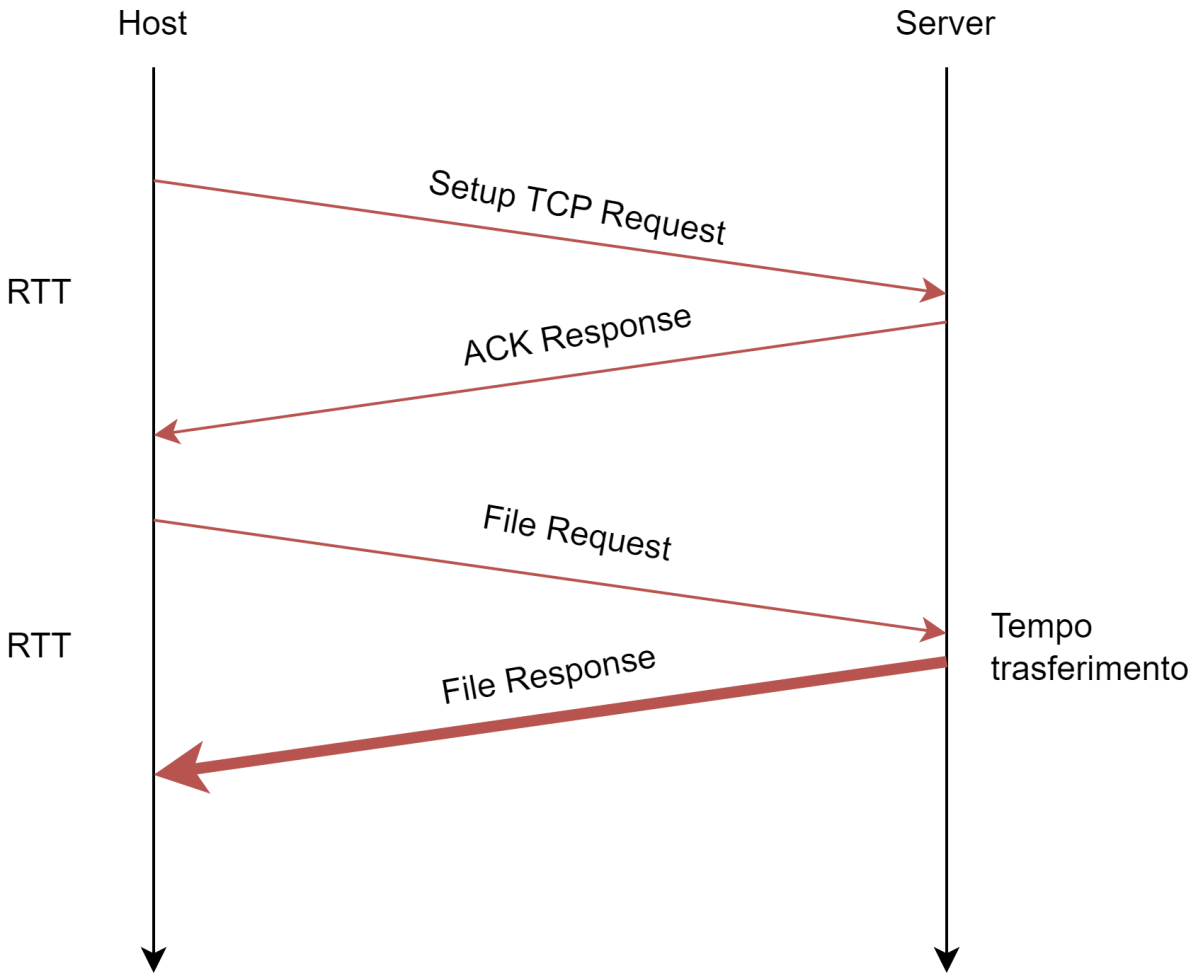
Il server si definisce “**stateless**” ovvero che non mantiene le informazioni riguardo agli utenti. Si possono salvare le informazioni con i così detti **cookies** i quali salvano le informazioni e cambiano l’applicazione di conseguenza.

Tipologie di connessioni

- **Non Persistente**: se a un indirizzo vengono chieste molte informazioni si apre una connessione per ogni oggetti richiesto. Per ogni oggetto viene aperta e chiusa la richiesta
- **Persistente**: possiamo anche non chiudere la comunicazione ma lo vedremo più avanti



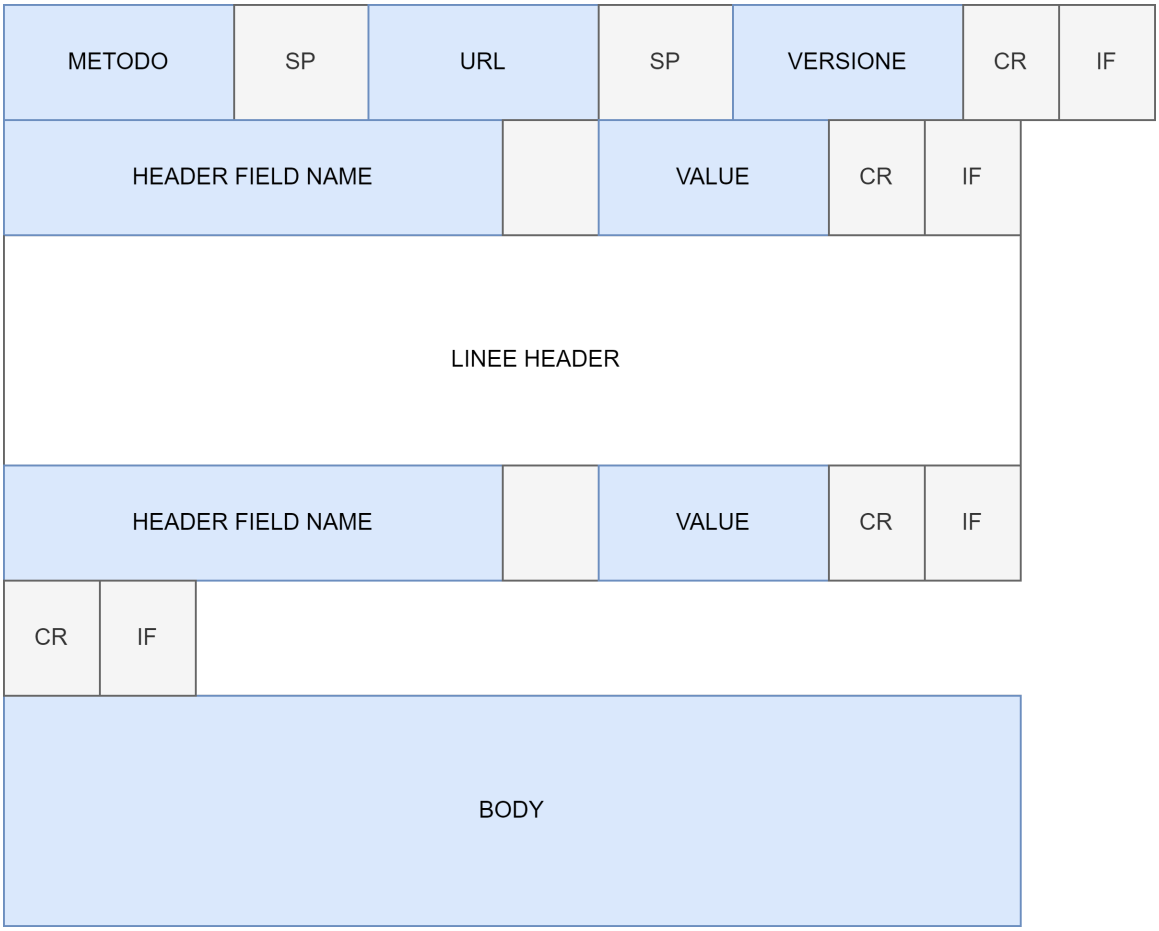
Nella comunicazione tra client e server viene calcolato il **RTT** (**Round Trip Time**) ovvero il tempo impiegato dall'inizio della richiesta alla ricezione della risposta. In generale quindi abbiamo 1 RTT per inizializzare la connessione, 1 RTT per la richiesta HTTP, quindi in generale abbiamo 2 RTT più il tempo di trasferimento dei file (se presenti).



Nelle comunicazioni persistenti invece la connessione dopo il setup non viene più chiusa la comunicazione per tutti gli oggetti presenti dentro il file. Come vediamo nelle non persistenti invece viene aperta e chiusa per ogni oggetto.

Come viene fatta una richiesta

Attraverso due metodi: **GET** e **POST** la prima è in chiaro nell'URL nel secondo invece è nel body e non visibile.



Questo è lo scheletro di una richiesta. Il server la legge e invia a sua volta una response così formata:

```
HTTP/1.1 200 OK
Connection: close
Date: Thu, 18 Aug 2015 15:44:04 GMT
Server: Apache/2.2.3 (CentOS)
Last-Modified: Tue, 18 Aug 2015 15:11:03 GMT
Content-Length: 6821
Content-Type: text/html
(data data data …)
```

La risposta può avere diversi codici:

- **200 OK:** request succeeded, requested object later in this msg.

- **301 Moved Permanently (URL redirection):** requested object moved, new location specified later in this msg (Location).
- **400 Bad Request:** request msg not understood by server.
- **404 Not Found:** requested document not found on this server.
- **505 HTTP Version Not Supported**

Cosa sono i cookies

I cookies servono nei siti per mantenere dati degli utenti che li visitano ad esempio vengono usati per: e-commerce, autorizzazioni, raccomandazioni o sessioni utente.

Per mantenere lo stato ci sono due metodi:

- Protocolli endpoint
- Cookies

Cosa sono i server proxy

Servono per soddisfare le richieste del client, sono macchine degli ISP e controlla che richieste vengono fatte, se possiede una copia del sito web ci mette meno a rispondere, si basa sul numero di volte che un utente interagisce con un sito.

Viene posta prima di tutto una richiesta al **proxy** e se non possiede la risorsa va ad interrogare il server di destinazione finale, se invece la possiede potrà soddisfare lui la richiesta così da essere più veloce, non vengono sovraccaricati i router mondiali e i server finali.

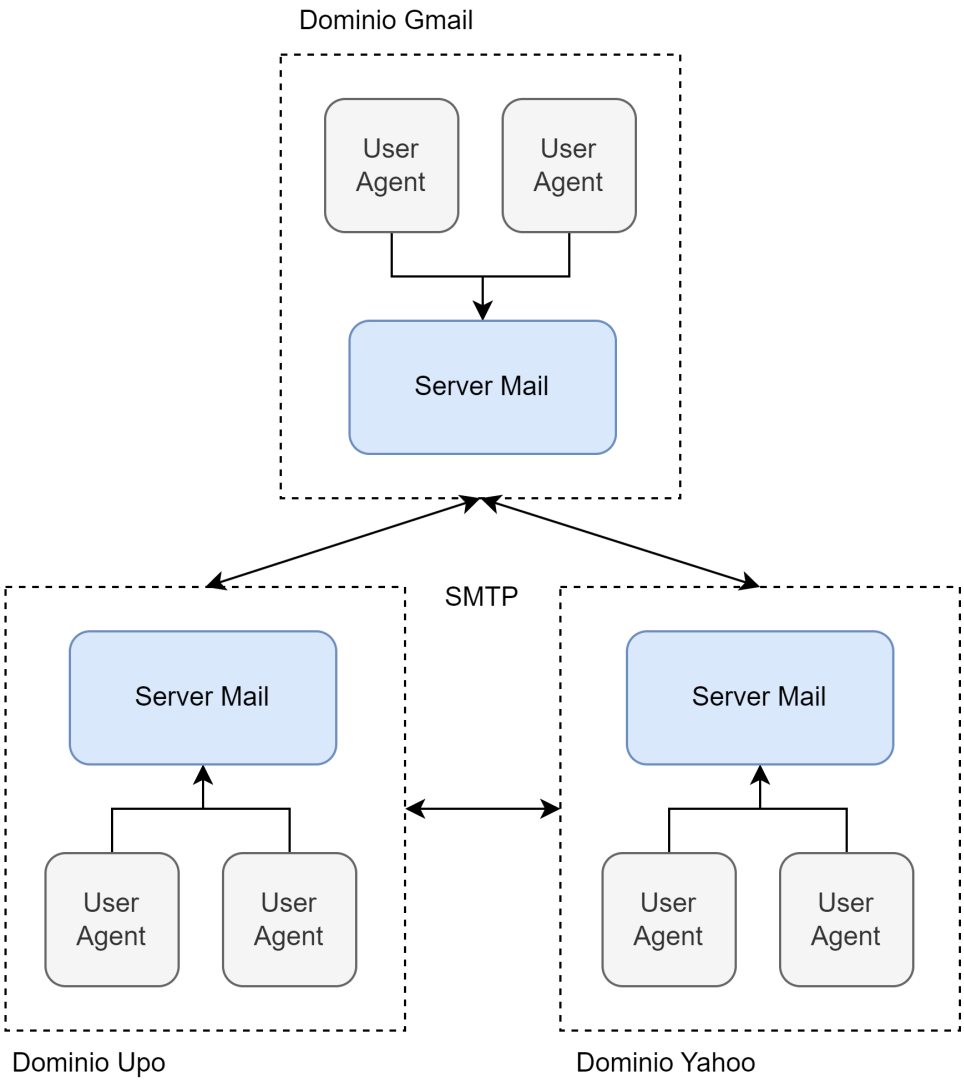
Ci sono anche lati negativi come censura di alcuni siti dato che la domanda del sito passa prima per il proxy e poi inviata al server finale.

Come fa a essere sicuro che ci **sia sempre l'ultima versione**? Attraverso il GET condizionale, ovvero viene indicato che la risorsa non sia stata modificata da una certa data. Il server risponde con il codice 304 se la pagina non è stata modificata dalla data specificata il GET altrimenti andrà al server finale.

Protocollo Email SMTP

Agiscono principalmente 3 entità:

- Utente: colui che legge, scrive la mail
- Server mail
- Protocollo **SMTP** (Simple Mail Transfert Protocol)



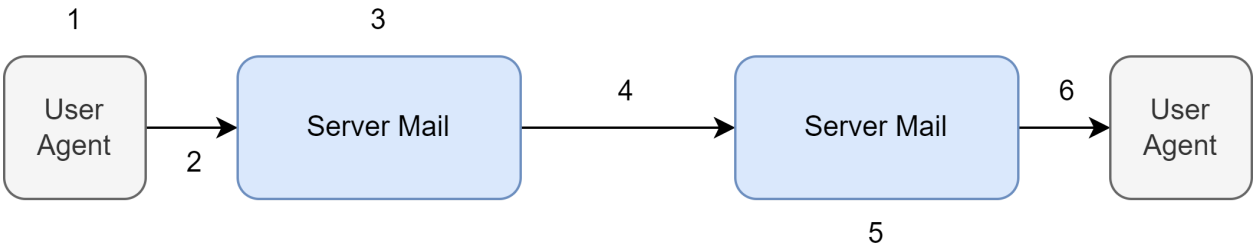
La **mail** è **affidabile** perchè il serve proverà ad inviare ogni 30 minuti le risorse che ha nel buffer, ogni utente ha una sua **mail box dentro** il **mail server** che salva le **mail non ancora scaricate** dentro il **nostro computer** o **dispositivo**.

Tra mittente e destinatario **non** ci sono altri server, tra il nostro server e il server del destinatario non ci sono intermezzi.

Il protocollo SMTP usa il TCP perchè non ci devono essere perdite, la porta di riferimento è la 25, ci sono 3 fasi:

- Handshake
- Invio messaggio
- Chiusura comunicazione

Il formato delle mail è a 7 bit ASCII, per questo a volte la mail è lenta ad essere inviata, proprio perchè c'è ancora ad oggi una conversione.



Queste sono le fasi che vengono eseguite dietro le quinte per l'invio di una mail:

1. **Utente X compone** la mail per il destinatario
2. La mail viene **caricata nel server mail** dell'utente X
3. Il server mail **controlla il dominio** (con il DNS) e lo **invia**
4. Viene **inviato con il TCP** il messaggio e viaggia da un server all'altro utente
5. Viene quindi caricato il messaggio nel **server mail di utente Y**
6. **Utente Y può leggere** la mail

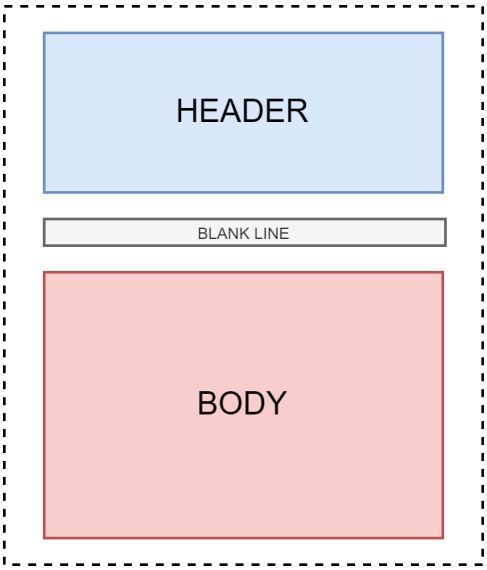
In una semplice interazione ci sono varie presentazioni tra client e server, se va tutto bene il server andrà a rispondere con li codice 250, altrimenti fornirà alcuni errori con altri codici. Il codice 354 dice al client che può iniziare l'invio. Per finire si usa il codice 221.

Conclusioni su SMTP

Il protocollo usa connessioni persistenti, richiede che i messaggi siano a 7 bit ASCII, richiede il codice CRLF (**Carriage Return Line Feed)** per terminare il messaggio.

A confronto con HTTP (prende) il protocollo SMTP invia, HTTP incapsula ogni messaggio mentre SMTP può incapsulare più messaggi.

Il formato SMTP è il seguente:



Comprende un **header** con le informazioni sul destinatario, mittente e soggetto, e poi abbiamo un **body** che contiene il messaggio in ASCII 7 bit.

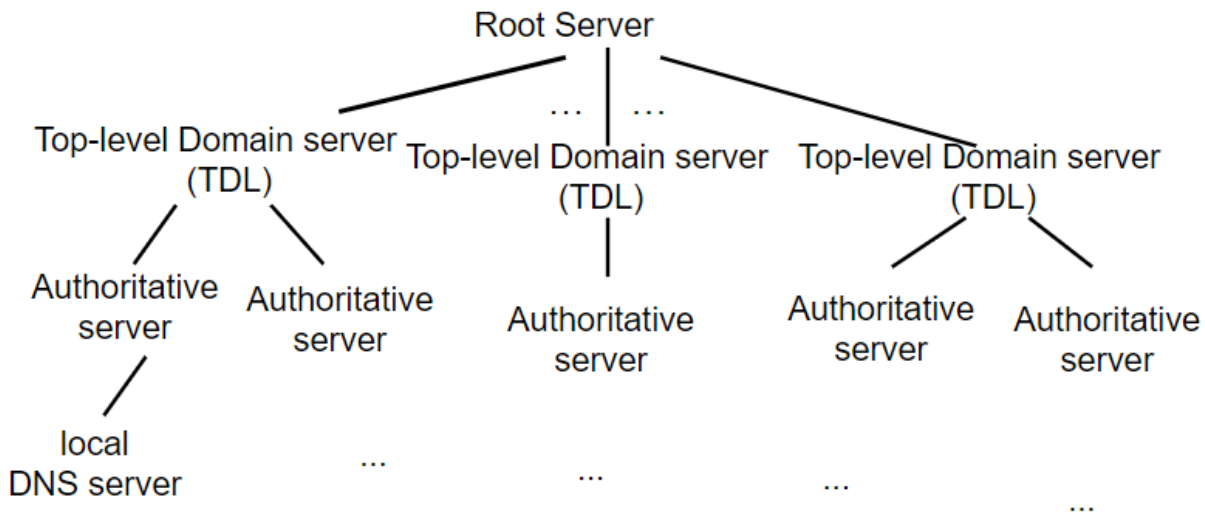
Il protocollo che **da client al server del destinatario** si occupa di far tutto è **SMTP,** ma dal server al destinatario ci sono tre diversi protocolli:

- **POP** (Post Office Protocol): POP3 è la versione vecchia dove abbiamo una fase di autorizzazione, successivamente possiamo inviare il messaggio. I principali comandi sono: USER, PASS, LIST, RETR, DELE, QUIT. Il carattere “.” server per identificare la fine della linea.
- **IMAP** (Internet Mail Access Protocol): mantiene tutti i file in un server e permette di organizzare le mail in cartelle.
- **HTTP:** Gmail, Outlook ecc..

Protocollo DNS

Per poter effettuare una conversione da nomi di dominio all’effettivo indirizzo IP del server, viene applicato il protocollo DNS implementato in un apposito server di conversione.

Il punto forte del **DNS server** è che è **distribuito** come la maggior parte dei sistemi i rete. Infatti non esiste un unico server con tutti i nomi di dominio ma ci sono molti, se non si trova in uno si cerca negli altri.

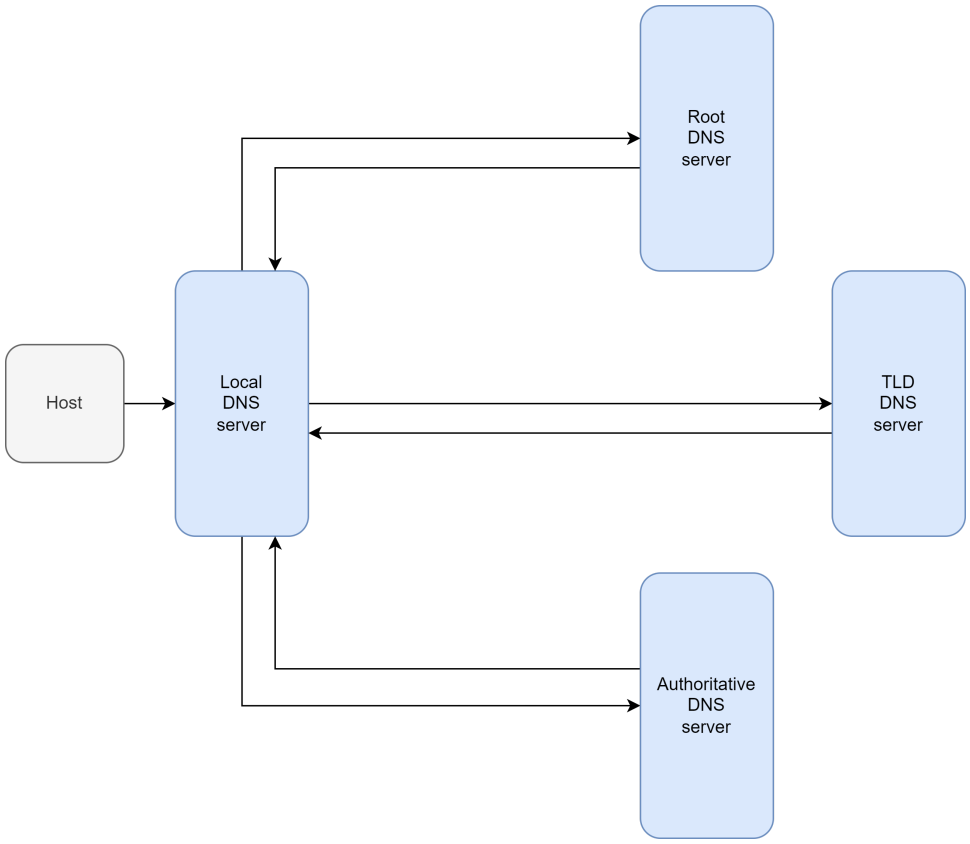


- **Root**: contengono informazioni sui server radice come .com, .net o .org.
- **TDL**: Top-Level-Domain sono i server responsabili per la parte dell'indirizzo con .com, .edu, .org, net ma anche per i paesi come .it, .fr, .ru.
- **Authoritative**: sono i server responsabili per una organizzazione, ad esempio uniupo.
- **Local**: server locale di un ISP, viene chiamato **default name server**, quando viene fatta una DNS query se il server ha il nome da tradurre nella **cache locale** lo risolve altrimenti fa da **proxy** per indirizzare in altri server di gerarchia maggiore.

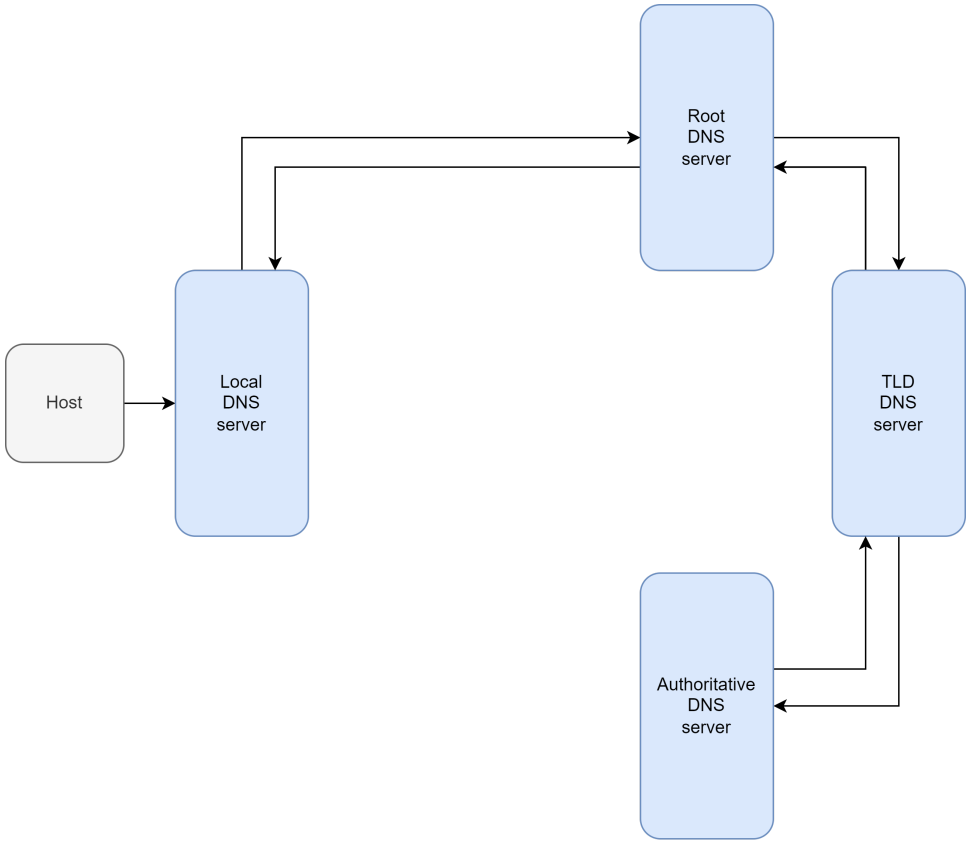
Tipologie di Query DNS

Ci sono due tipologie di interrogazioni DNS per poter svolgere la conversione:

- **Query iterativa**: contatta tutti gli altri server DNS e quindi tutti il carico di lavoro viene svolto dal DNS locale.



- **Query ricorsiva:** permette di poter lasciare il carico al server e far propagare la richiesta senza far pesare il server locale.



Appena un server dei nomi trova un **mapping** che **non aveva al suo interno**, lo **salva nella cache**, per prevenire di avere mapping datati si imposta un TTL.

I nomi di dominio salvati in TLD **non sono salvati nei server DNS** locali perchè sono raramente visitati e quindi noi vogliamo in cache solamente i mapping più richiesti.

Record DNS

I nomi di dominio vengono mappati secondo uno standard definito come RR (Resource Record), nel seguente formato: **RR (name, value, type, ttl).**

Ci sono diversi tipi:

- **A**: **name** è un nome di host, **value** è l'ip
- **NS**: **name** è un dominio, **value** è il nome del server autoritativo
- **CNAME**: **name** è un alias del vero nome, **value** è il nome vero
- **MX**: **value** è il nome del server mail

ID	FLAGS
DOMANDA	RISPOSTA
AUTORITY RRs	RRs ADDIZIONALI
DOMANDA	
RISPOSTA	
AUTORITY	
INFORMAZIONI ADDIZIONALI	

Attacchi DNS

Ci sono diverse tipologie di attacchi DNS, il primo è il **DDoS** ovvero bombardare di richieste un server per sovraccaricare il suo lavoro, non è più usato oggigiorno perchè ci vorrebbe troppa potenza di calcolo.

Bombardare il TLD che è il più pericoloso, **cambiare il traffico di navigazione** attraverso il man in the middle oppure **DNS poisoning**.

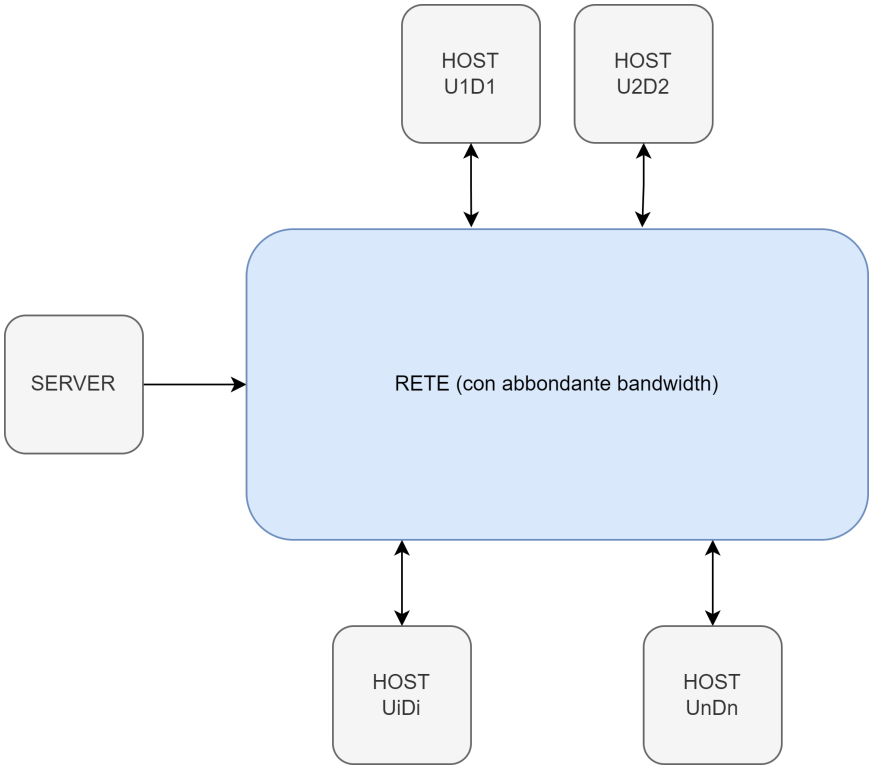
Architettura Peer-To-Peer P2P

Il server non è sempre attivo, i sistemi finali comunicano direttamente. I peer sono connessi in modo intermittente.

Ad esempio Skype serviva per poter effettuare chiamate in modo P2P e quando lo avevamo aperto in background aiutavamo altri a migliorare la connettività delle chiamate.

Distribuzione di file: Client-Server

Il server è l'unico che possiede il file e deve caricare tutte le sue parti nella rete, quindi per calcolare il tempo massimo dobbiamo calcolare per il server:



- Tempo per **una copia**: F/Us
- Tempo per **inviare N copie**: NF/Us

Per il client invece:

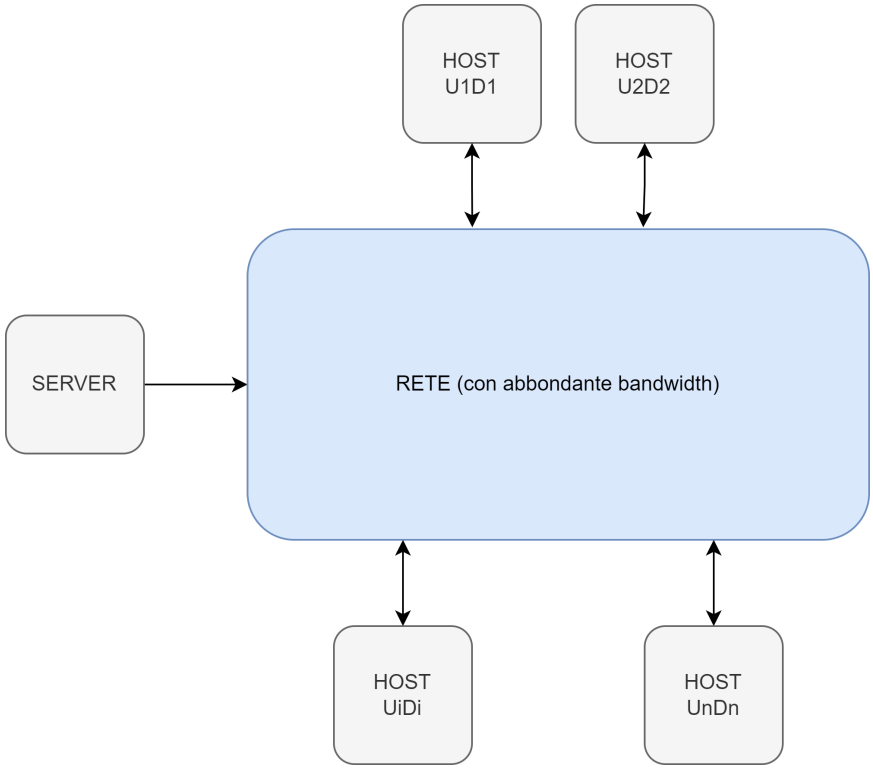
- d_min: minimo rate di download
- Tempo **minimo di download**: F/d_min

$$Formula : D_{d-c} : max(NF/u_s, F/d_{min})$$

Il tempo quindi per distribuire N copie tra i client incrementa per colpa dei client con il download rate più basso.

Distribuzione di file: Peer-2-Peer

Invece quà abbiamo una serie di peer che svolgono azioni di server e client, quindi tutti i client sono poi proprietari del file e possono abbassare i tempi di download.

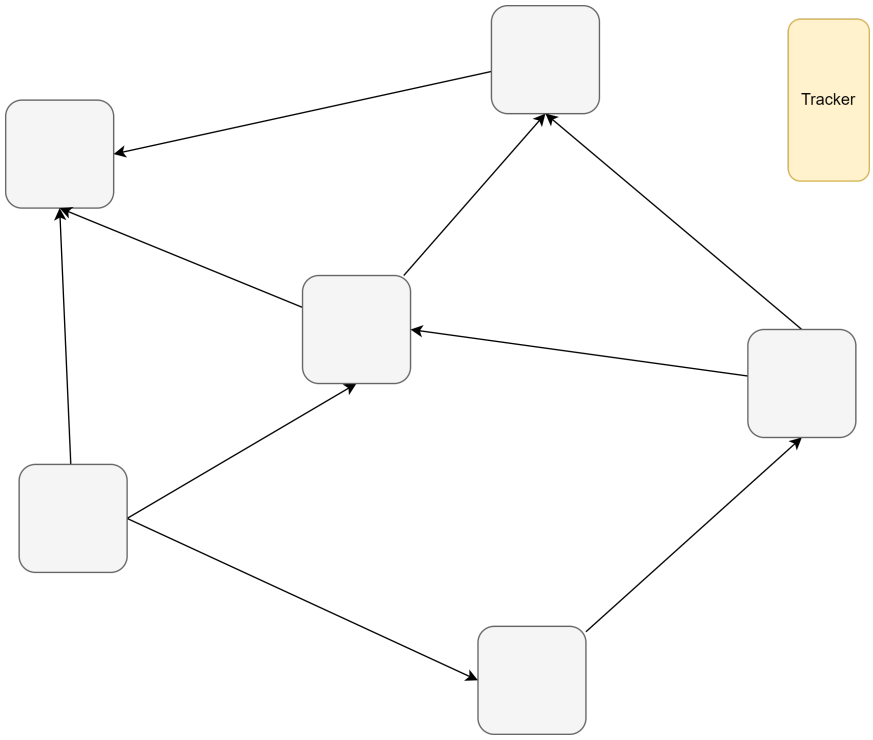


$$Formula: D_{p2p}: max(F/u_s, F/d_{min}, NF/(u_s + \sum u_i))$$

BitTorrent

Si chiama così perchè i file sono **divisi in chunks** e sono distribuiti in tutta la rete di peer, serve però un punto di accesso a questa rete.

Abbiamo il tracker che è una entità che detiene la lista di peer per poter sapere a chi condividere il file.



Il nuovo peer che si è unito al torrent, **non ha chunks** di dati **ma li accumula nel tempo**, si **registra** al **tracker** per prendere la lista di peer nel torrent.

Mentre sta **scaricando il peer fa l'upload di chunks in altri peers**. Ci sono però peer che potrebbero scaricare tutto il file e poi andarsene (**free-riders**), per questo viene calcolato quanto ho scaricato e quanto ho caricato (**churn**).

- **Richieste di chunks**: i peer hanno una lista di chunks, periodicamente bisogna aggiornarsi per avere sempre tutti i chunks aggiornati. Bisogna richiedere sempre i più rari prima perchè essendo una rete fatta da entità che possono entrare e uscire quando vogliono, bisogna sempre avere tutti.
- **Invio di chunks (tit-for-tat):** dobbiamo scegliere 4 peer, viene scelto ogni 10 secondi per veder quali sono i 4 vicini migliori per il download di chunks. Ogni 30 secondi (attenzione che non c'è un calcolo dietro, gli sviluppatori dei protocolli hanno semplicemente visto che andava bene) invece viene scelto casualmente un peer per l'invio di chunks, viene verificato che possa entrare in top 4, per il così detto "**unchoke**".

Video streaming e CDNs

Il traffico di video è il maggiore consumatore della banda di internet, Netflix consuma circa il 37% della banda di un ISP. Come si può gestire un tale carico?

Possiamo ragionare a livello applicativo, quindi per esempio mettere più copie della stessa serie su più server quando sappiamo che c'è un picco di richieste.

I video sono una sequenza di immagini, circa 24 al secondo, il fatto è che ogni frame ha molte parti che si ripetono, possiamo inviare quindi solo le parti che cambiano.

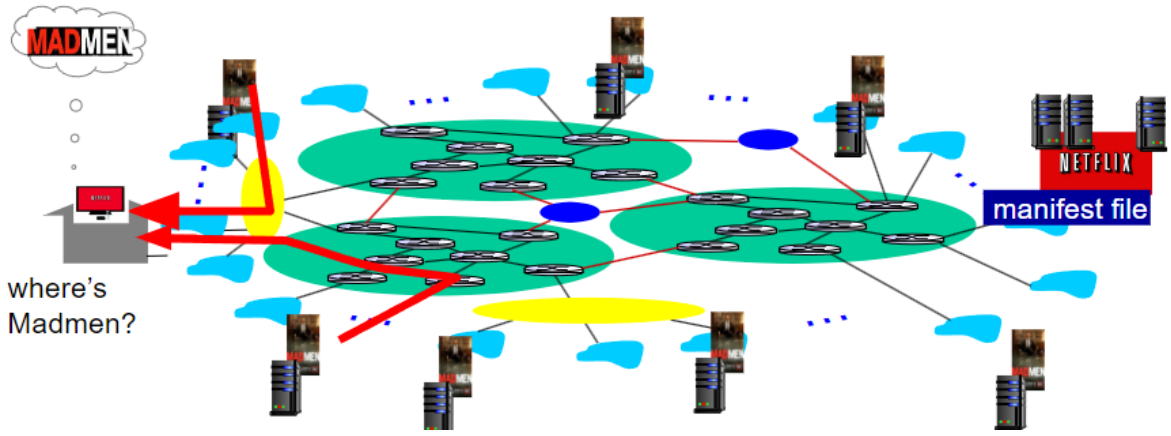
Abbiamo due tipi di bit rate:

- **CBR**: Constant Bit Rate, il coding delle immagini è fatto a un rate fisso.
- **VBR**: Variable Bit Rate, il coding è fatto a un rate variabile, ovvero quando il video è pixellato in un punto e poi riprende normale.

Protocollo DASH

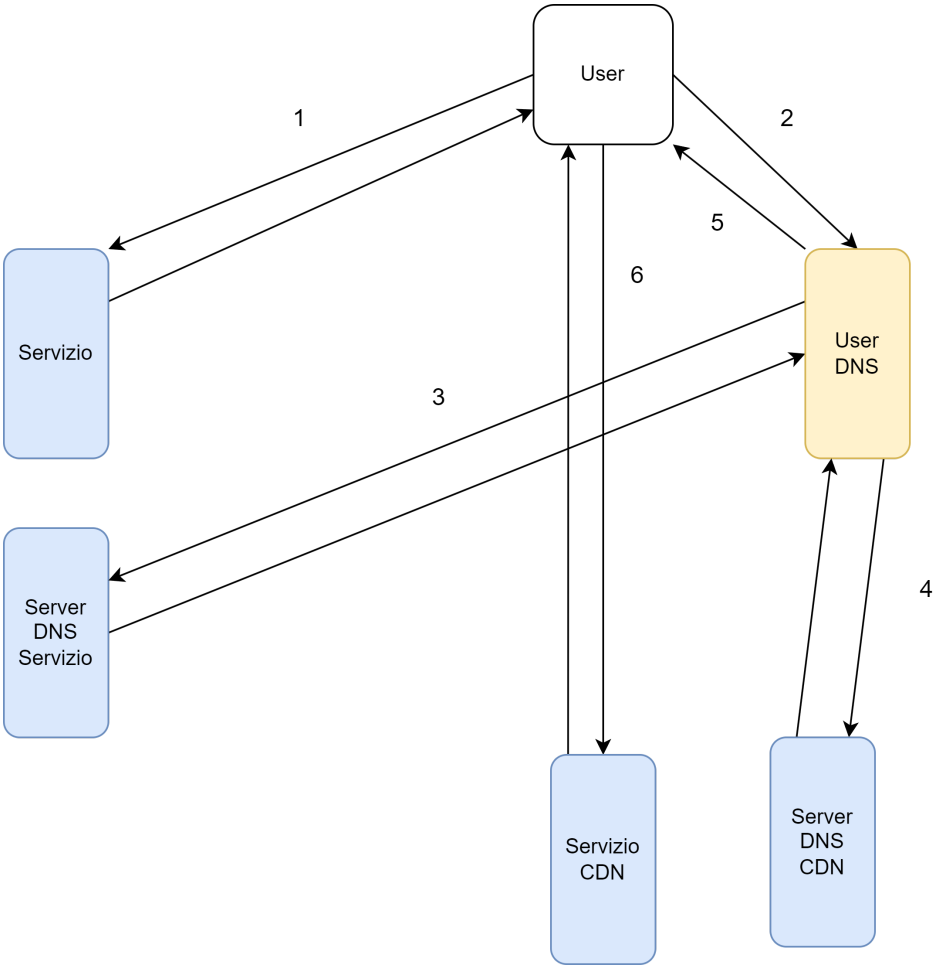
Il protocollo è Dynamic Adaptive Streaming over HTTP ovvero dove, il server divide il video in più chunks, viene fornito un file di **manifesto per sapere tutte le risorse in URL**. Il client controlla la bandwidth e si cerca di capire qual è la migliore per evitare di avere un video bloccato.

Le **Content Delivery Networks** ovvero quando abbiamo più copie che sono all'interno dei server:



A seconda della situazione verrà scelta o una o l'altra, il provider del servizio nel nostra cosa Netflix, dovrà fornire un file "**manifest**" su com'è distribuito il suo contenuto, in quali server e con quale qualità.

Come viene svolta l’interrogazione CDN?



1. L’utente va nella piattaforma di streaming per vedere il contenuto e riceve come risposta l’URL del contenuto
2. L’URL viene risolto dal DNS locale dell’utente e inizia la ricerca
3. Il DNS locale comunica con il DNS del servizio, se l’IP risposta non è presente nella cache va a recuperare l’indirizzo del servizio CDN nel suo server DNS.
4. Il DNS locale allora interroga un altro DNS ovvero quello della piattaforma di CDN, a questo punto ha finalmente l’IP del server con l’effettivo video
5. Il server DNS locale restituisce all’utente l’ip del server
6. L’utente può interrogare il server del servizio che al suo interno ha il video e si apre una comunicazione DASH.