CAPITOLO 2: Livello applicativo

L’obiettivo concettuale è quello di capire gli aspetti che riguardano l’implementazione delle applicazioni di rete e dei protocolli di rete, ossia capire quali servizi ha una rete e studiare i protocolli di supporto alle applicazioni di rete (http, ftp, dns).

**2.1 Architettura applicazioni di rete**

Le applicazioni di rete sono tante e in continua evoluzione (web, e-mail, giochi, streaming) e consistono in processi distribuiti che comunicano tra loro ma in esecuzione su macchine diverse. L'architettura dell'applicazione è progettata dallo sviluppatore dell'applicazione e regola come l'applicazione è strutturata tra i vari end-system. Quindi lo sviluppatore dovrà decidere fra due paradigmi architetturali: l'architettura client-server o peer-to-peer (P2P):

1. L’architettura client-server ha da un lato delle macchine sempre accese (server) che hanno un indirizzo IP permanente e dall’altro quelli che accedono ai servizi (client). Essi interagiscono solo con i server e possono essere connessi alla rete in modo intermittente, infatti hanno un IP che varia. In questo tipo di architettura, un server non sempre riesce a gestire tutte le richieste dai client; a tale scopo si utilizzando i data center che ospitano numerosi server con lo scopo di creare un potente server virtuale.
2. L’architettura peer-to-peer è nata per la condivisione di file. Non esistono server ma sono gli utenti che si scambiano informazioni tra loro quando sono connessi alla rete. È un’architettura molto scalabile perché tutti posso comunicare con tutti e quindi ho un’alta efficienza di scambio dati. È un’architettura molto potente, tuttavia, non vi sono garanzie di sicurezza, performance e affidabilità.

**2.2 Comunicazione tra processi**

In un’applicazione di rete ho dunque processi che comunicano tra loro ma sono in esecuzione su macchine diverse. Per comunicare si scambiano messaggi attraverso la rete utilizzando dei protocolli che definiscono le modalità di comunicazione. In particolare, definiscono i messaggi scambiati, il loro significato e le azioni da intraprendere:

-Nella client-server c’è un processo che inizia la comunicazione (client) e un processo che aspetta di essere contattato per iniziare la sessione (server). L’interfaccia con cui l’utente si connette ad un’applicazione di rete è detta “User Agent” (UA) che si connette con l’utente e con la rete e gli permette di usare l’applicazione.

-Nella P-2-P quando il peer A chiede al peer B di inviare un file, il peer A è detta client mentre il peer B è detto server.

Ogni messaggio inviato da un processo all'altro deve attraversare la rete. Un processo invia e riceve un messaggio attraverso un'interfaccia detta socket. In particolare, è l'interfaccia che risiede tra il livello di applicazione e il livello di trasporto con cui il programmatore interagisce tramite chiamate di sistema.

**Indirizzamento dei processi**

Affinché il processo in esecuzione su un host invii i pacchetti al processo in esecuzione su un altro host, il processo ricevente deve avere un indirizzo specificando l'indirizzo dell'host e l’identificatore che specifica il processo ricevente nell’host di destinazione. In internet, l'host possiede il proprio indirizzo IP composto da 32-bit. Inoltre, per conoscere l'indirizzo dell'host a cui è destinato il messaggio, il processo mittente deve identificare il processo ricevente poiché un host potrebbe avere in esecuzione tante applicazioni di rete. Perciò si introduce il numero di porta da 16 bit.

**Servizi di trasporto disponibili alle applicazioni**

L'applicazione invia i messaggi all'interno del socket e dall'altra parte il protocollo del livello di trasporto ha la responsabilità di prendere il messaggio dal socket. Molte reti forniscono diversi protocolli di trasporti di rete tra cui scegliere in base ai servizi offerti dal protocollo:

1. Trasferimento affidabile di dati: per supportare le applicazioni in cui le informazioni sono fondamentali, bisogna garantire che i dati inviati siano consegnati correttamente e completamente dall'altra parte dell'applicazione. Se un protocollo fornisce un servizio di consegna dati garantita, allora il dato viene inviato correttamente e interamente e viene detto trasferimento affidabile di dati. Al contrario, se un'applicazione non fornisce questo servizio, allora si potrebbero avere delle perdite al processo ricevente. Questo tipo di applicazioni vengono dette applicazioni loss-tolerant. Ne sono un esempio le applicazioni multimediali.
2. Throughput: poiché le altre sessioni condivideranno la banda lungo il percorso di rete, e poiché queste altre connessioni possono interrompersi, il throughput disponibile può fluttuare nel tempo. Un servizio che può essere offerto è quello di garantire un throughput disponibile ad un certo rate. Da cui, un'applicazione potrebbe richiedere un throughput di r bit/s, e il protocollo di trasporto dovrebbe garantirne almeno quella quantità. Nel caso in cui il protocollo di trasporto non può fornire quel throughput, l'applicazione dovrà codificare i dati ad un livello inferiore o abbandonare la connessione finché non ha il rate necessario. Tali applicazioni vengono detti bandwidth-sensible application o altrimenti vengono dette elastic application.
3. Timing: un protocollo di livello di trasporto può fornire delle garanzie sul tempo. Ne sono un esempio le applicazioni real-time in cui l'influenza del tempo e dei ritardi assume un ruolo centrale.
4. Sicurezza: un protocollo di trasporto può fornire un'applicazione con uno o più servizi di sicurezza che, cifrando i dati prima della consegna, fornisce una comunicazione confidenziale tra due processi anche se il messaggio può essere osservato.

**Servizi di trasporto offerti da Internet**

Internet mette a disposizione delle applicazioni due protocolli di trasporto: UDP e TCP.

1. Servizi TCP: il modello del servizio TCP include un servizio di connessione e un servizio di trasferimento dati affidabile. Quando un'applicazione invoca TCP come suo protocollo di trasporto, essa riceve entrambi questi servizi:
   1. Connection-oriented service: TCP fa in modo che client e server si scambino informazioni di controllo a livello di trasporto prima che i messaggi di livello di applicazioni iniziano a fluire. Questa procedura è detta handshaking, preparando il client e il server alla partenza dei pacchetti e dopo questa fase si dice che esiste una connessione TCP tra le socket dei due processi. Viene detta anche connessione full-duplex poiché entrambi possono inviare dati. Quando l'applicazione termina l'invio deve chiudere la connessione.
   2. Servizio di trasferimento affidabile: i processi comunicanti possono contare su TCP per trasportare dati senza errori e nel giusto ordine. Quando un lato dell'applicazione passa un flusso di byte alla sua socket, può affidarsi a TCP per consegnare lo stesso flusso di byte alla socket di ricezione senza perdita o duplicazione di byte;
   3. TCP include anche un meccanismo di controllo della congestione applicando una “strozzatura” del processo d'invio quando il traffico in rete appare eccessivo.
2. Servizi di UDP: UDP è un protocollo di trasporto leggere dotato di servizi minimalisti. Infatti, è senza connessione, non necessita di hadshaking e fornisce un servizio di dati non affidabile. Il protocollo, quindi, non garantisce che il messaggio raggiunga il processo di destinazione né che questi arrivino con lo stesso ordine in cui siano stati inviati.

**Web**

Il Web è una collezione di risorse/oggetti distribuiti all'interno dei server della rete. Ogni oggetto/risorsa può essere un file statico su una macchina o richieste generate dinamicamente. Il Web opera su richiesta: si può avere ciò che si vuole, quando lo si vuole.

Una pagina web è un contenitore di risorse come un file HTML che può includere dei collegamenti a degli oggetti indirizzabile tramite un URL, che ha il compito di etichettare in modo uniforme una risorsa. L’HTML, invece, è il linguaggio di etichettatura di ipertesti che fornisce lo standard con cui descriviamo un documento ipertestuale che ci permette di formattare del testo aggiungendo riferimento a oggetti, immagini o link ipertestuali ad altri oggetti. Infine, c’è un protocollo HTTP che viene usato dalle pagine web per comunicare. Il web, infatti è un’applicazione che si basa sul protocollo HTTP per far interagire le varie parti tra loro. L'indirizzamento tramite URL avviene in questa maniera:

https://www.blog.libero.it/wp/animelovers/2021/

• Protocollo: “http” che definisce come la pagina viene acceduta;

• Nome dell'host: “//www.blog.libero.it” che indica dove la pagina è localizzata;

• Nome della porta: in automatica viene presa la porta 80 altrimenti deve essere specificata

• Percorso del file: “/wp/animelovers/2021/” è il path alla risorsa di rete, ossia il nome dell’oggetto.

Dunque, lo URL mi da informazioni su quale protocollo usare, a quale macchina collegarsi, a quale numero di porta e quale è l’oggetto di cui devo fare richiesta.

**HTTP**

È un protocollo a livello di applicazione del Web che costituisce il cuore del Web. È implementato in due programmi, client e server, in esecuzione su sistemi periferici diversi che comunicano tra loro scambiandosi messaggi. Un browser web implementa il lato client di http, facendo richiesta di oggetti e ottenendoli dal server, mentre un server web implementa il lato server di HTTP ospitando gli oggetti web indirizzabili tramite URL. HTTP definisce, dunque, come i clients web richiedono le pagine web ai web servers e come i server trasferiscono le pagine web ai clients. Quando un utente richiede la pagina Web, il browser invia al server messaggi di richiesta HTTP per gli oggetti nella pagina. Il server riceve la richiesta e risponde con un messaggio di risposta HTTP contenente gli oggetti.

Il protocollo HTTP utilizza il protocollo TCP come il suo protocollo di livello inferiore. Infatti, il client HTTP instaura una connessione TCP con il server nella porta 80 e una volta che è stabilita, client e server si possono scambiare messaggi che seguono il protocollo HTTP. Una volta terminato lo scambio di messaggi chiudo la connessione. Inoltre, HTTP è detto stateless, ossia i server non mantengono informazione sulle richieste passate quindi non ha memoria. Infatti, se un client chiede in un periodo di tempo ravvicinato lo stesso file, il server non risponde dicendo che l'oggetto è già stato servito, ma risponde come se fosse stata la prima connessione.

In molte applicazioni Internet, client e server comunicano per un lungo periodo di tempo, con il client che inoltra una serie di richieste e il server che risponde a ciascuna di esse. A seconda del tipo di interazione fra i due, il progettista dell'applicazione deve scegliere quale tipo di connessione adottare: connessioni HTTP persistenti, connessioni HTTP non persistenti.

Connessione HTTP non persistente:

Quando un client web collegato con un server web deve fare molteplici richieste per oggetti diversi, per ogni oggetto bisogna usare una connessione TCP diversa, quindi non rimane la stessa connessione tra diversi oggetti. Analizziamo nel dettaglio i passi di trasferimento di una pagina Web dal server al client nel caso di connessione non persistente:

1.Il processo client HTTP inizia una connessione TCP al server sulla porta 80 di default. Associata a questa connessione TCP, dopo che il server ha accettato la connessione, ci sarà un socket al client ed uno al server;

2.Il client HTTP invia una richiesta HTTP al server tramite il suo socket, questa richiesta contiene l'oggetto che richiede;

3.Il processo HTTP del server riceve la richiesta di messaggio nella sua socket e restituisce l'oggetto dal suo archivio, la incapsula in un messaggio di risposta HTTP e la invia al client dalla sua socket;

4.Il processo del server HTTP chiude la connessione;

5.Il client HTTP riceve il messaggio di risposta e la connessione TCP termina. La risposta contiene il file HTML e nel caso in cui vi sono delle referenze ad altri oggetti le esamina;

6.Per ogni referenza all'interno del file HTML ripete i passi 1-5.

Quindi nella connessione non persistente, il client deve avviare una connessione ogni qual volta deve richiedere un oggetto al server avviando una connessione TCP, inviando una richiesta HTTP, aspettando la sua risposta e nel caso ripetere.

In questa ottica, definiamo RTT (Round Trip-Time) il tempo necessario affinché un piccolo pacchetto viaggi dal client al server e viceversa (andata e ritorno). Questo parametro include:

• ritardo di propagazione;

• ritardo di accodamento;

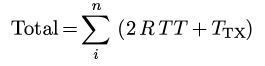
• ritardo di processamento;

• ritardo di trasmissione;

Per ogni oggetto quindi avremmo un tempo totale di:



(Ttx=tempo di trasmissione dell’oggetto)

Per n oggetti:

I difetti di questo tipo di connessione sono i seguenti:

• Per ogni oggetto richiesto occorre aspettare 2\*RTT;

• Si ha un consumo di risorse elevato;

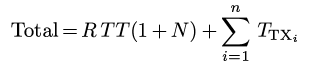
• Anche considerando che un browser può avviare più connessioni in paralleli, esse gravano sul server che ne può gestire di meno.

Connessioni HTTP Persistenti:

Per le problematiche sopra riportate, una valida alternativa è quella di ricorrere alle connessioni persistenti. Nelle connessioni persistenti, il server lascia la connessione TCP aperto dopo aver inviato la risposta. Le successive richieste e risposte tra i due possono essere inviati lungo la stessa connessione. Questo permette di inviare una intera pagina Web su una singola persistente connessione TCP e, in alcuni casi, inviare molteplici pagine Web contemporaneamente. Le connessioni persistenti si differenziano in due tipologie:

1. Connessioni persistenti senza pipeling: questi tipi di connessioni non consentono l'invio incontrollato di richieste e quindi di risposte tra client e server. In particolare, il client deve aspettare la risposta del server per effettuare la nuova richiesta mantenendo, però, la stessa connessione.

Tramite questo tipo di connessione, l'RTT viene ridotto a 1 RTT ed in particolare, nel caso di invio di richieste e risposte di n oggetti:



1. Connessioni persistenti con pipeling: questi tipi di connessione consentono l'invio e la ricezione di richieste e risposte HTTP in contemporanea, senza quindi dover aspettare la risposta del server. Quindi il tempo che realmente viene speso è quello per l'inizializzazione della connessione TCP e per l'invio del primo oggetto. Il tempo totale per n oggetti vale:



**Formato dei messaggi HTTP:**

Le specifiche HTTP includono la definizione di due formati di messaggi: di richiesta e di risposta.

1. Richiesta: I messaggi HTTP sono scritti in ASCII e facilmente comprensibili dall'uomo e consistono in cinque righe.

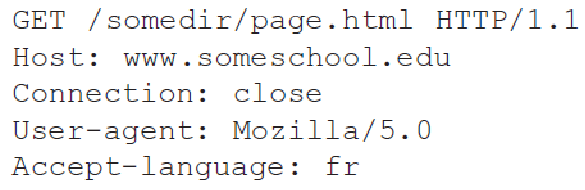
-Request line: è la prima linea di un messaggio di richiesta HTTP. Essa è composta di tre campi:

1. il campo del metodo: può assumere diversi valori come GET, POST, HEAD, PUT e DELETE. Il metodo GET viene usato per richiedere un oggetto. Il metodo POST serve per inviare dati al server che vengono visualizzati nell’entity body. Il metodo HEAD è molto simile al metodo GET, ma viene utilizzato in ambito di sviluppo poiché risponde alla richiesta HTTP, ma non invia l'oggetto richiesto. Il metodo PUT è utilizzato per caricare sul server un file e il metodo DELETE è utilizzato per eliminare un oggetto dal server Web;
2. il campo URL: contiene il percorso dell'oggetto che deve essere gestito dal campo del metodo;
3. il campo della versione del protocollo http richiesta;

-Header line: sono le linee che seguono la request line. Esso è composto da:

1. Host: specifica l'host su cui risiede l'oggetto. Viene specificato poiché necessario dalle caches Web;
2. Connection: il browser sta dicendo che non vuole avere una connessione persistente e una volta in cui il server ha inviato l'oggetto richiesto, vuole che la connessione sia chiusa;
3. User-Agent: è il tipo di browser che sta facendo la richiesta. Questo è utile poiché i browser possono visualizzare le risorse in maniera differente;
4. Accept-Language: è una specifica su quale lingua vorrebbe ricevere la pagina HTML. Se non presente, viene inviata quella di default.

-Entity body: Nel metodo get questo campo è vuoto; mentre nel metodo POST, e contiene i dati ossia il corpo dell’entità.



1. Risposta: Il messaggio di risposta è formato da tre sezioni: la riga di stato, sei righe di intestazione e il body.
   1. body: contiene l'oggetto richiesto;
   2. status line: è formato da tre campi:

-il campo della versione del protocollo;

-Il campo del codice di stato che indica il risultato della richiesta HTTP. I più comuni sono i seguenti:

-200 OK: la richiesta è avvenuta con successo e l'informazione è ritornata;

-301 Moved Permanently: l'oggetto richiesto è stato spostato permanentemente e si specifica nel campo Location il nuovo URL;

-400 Bad Request: questo è il generico codice di errore che indica che la richiesta non è stata compresa dal server;

-404 Not Found: Il documento richiesto non esiste su questo server;

-505 HTTP Version Not Supported: La versione del protocollo di richiesta HTTP non è supportata dal server;

-Il valore del codice di stato;

* 1. header line: formata da:

-Connection: indica che al client che sta per chiudere la connessione TCP una volta inviato il messaggio.

-Date: indica la data e l'ora di quando la risposta HTTP è stata creata e mandata dal server. In particolare, coincide con il tempo in cui il server preleva l'oggetto dal proprio sistema, lo inserisce nel messaggio di risposta e la invia.

-Server: indica che il messaggio è stato generato da un certo server.

-Last-Modified: indica la data e il giorno in cui l'oggetto è stato creato o modificato. Questa linea è utile per il chaching delle pagine Web nei proxy server.

-Content-Lenght: indica il numero di bytes nell'oggetto che è stato inviato;

-Content-Type: indica che il tipo dell'oggetto all'interno della risposta.

HTTP permette un meccanismo di controllo ad accesso ai contenuti tramite nome e password. Se ci deve essere questo accesso autorizzato, quando il client invia il messaggio GET, il server non risponde con l’oggetto ma con un codice di richiesta di autenticazione

**Cookies**

Il protocollo HTTP è stateless quindi non ha memoria. Spesso il sito Web necessita di identificare gli utenti, limitare le attività degli altri utenti o di personalizzare i contenuti degli utenti. A tale scopo, si utilizzano cookies.

I cookie sono formati da quattro componenti:

• una riga di intestazione nel messaggio di risposta HTTP;

• una riga di intestazione nel messaggio di richiesta HTTP;

• un file mantenuto sul sistema dell'utente e gestito dal browser;

• un file mantenuto sul database sul sito.

I cookies sono usati per identificare un utente. La prima volta in cui l'utente visita il sito, l'utente fornisce un identificativo. Durante le successive sessioni, browser trasferisce un cookie header al server per permettere la sua identificazione nel server.

Riassumendo, i cookie permettono l'implementazione di un meccanismo di autorizzazione, permettono di tenere traccia delle preferenze personali di ogni utente e quindi raccomandare/consigliare dei prodotti simili e permette di tenere traccia dello stato della sessione utente. Tuttavia, vi è una battaglia sull'eticità di questi cookie al livello di privacy poiché tramite essi si può conoscere molto sulle abitudini di un utente.

**WEB Caches (proxy server)**

Una Web Cache o proxy server è un'entità di rete che soddisfa le richieste HTTP al posto del Web server originario, ossia fa da intermediario per un insieme di utenti. Esso ha il proprio dispositivo di archiviazione e mantiene le copie degli oggetti recentemente richiesti.

Il browser di un'utente può essere configurato in modo tale che tutte le richieste HTTP dell'utente vengano ridirette nella Web Cache.

Un proxy server funziona nella seguente maniera:

1.Il browser stabilisce una connessione TCP al Web Cache e invia una richiesta HTTP per l'oggetto alla Web cache;

2.La Web cache controlla se la copia dell'oggetto è salvata localmente. Se lo è, il proxy server ritorna l'oggetto all'interno di un messaggio di risposta HTTP al browser del client;

3.Se il proxy server non possiede l'oggetto, la web cache apre una connessione TCP al server originario. A questo punto, il proxy server invia una richiesta di connessione TCP con il server originario ed invia la richiesta dell'oggetto;

4.Quando la Web cache riceve l'oggetto, si salva una copia localmente e ne manda un'altra, all'interno di un messaggio di risposta HTTP, al browser del client.

Il vantaggio avviene quando ci sono tanti client che fanno la stessa richiesta per cui la richiesta non viene inviata al server di origine ma al proxy server che, avendo già l’oggetto richiesto, lo restituisce direttamente all’utente stesso. Se non ce l’ha, lo richiede al server di origine, lo ottiene, lo salva e lo invia al client.

Come possiamo notare, la cache è sia server che client durante l'intero processo. Quando riceve la richiesta da un browser e gli invia risposte allora è un server. Quando invia la richiesta e riceve la risposta dal server di origine è un client.

La Web Caching è utile per due motivazioni:

• riduce il tempo di risposta per una richiesta del client.

• in ambito aziendale permette di capire se l’uso della rete è effettuato in maniera corretta ed eventualmente eliminare determinati contenuti.

• riduce il traffico di un accesso istituzionale ad internet e quindi non si deve spendere per un miglioramento della rete.

• riduce globalmente il tempo di risposta, è possibile, quindi, migliorare il traffico sull'intero internet e migliore le prestazioni di tutte le applicazioni.

**Get Condizionale**

Abbiamo introdotto il caching delle richieste HTTP dalla parte dei client al fine di velocizzare l'intera rete fornendo una copia già salvata in un browser o in proxy server. Tuttavia, HTTP possiede un meccanismo che permette alla cache di verificare se un oggetto è stato modificato. Questo meccanismo è detto GET condizionale e consiste in una richiesta al server di origine da parte del proxy server su un oggetto solo se è stato modificato entro una certa data: se è così allora l’oggetto viene restituito altrimenti viene inviato un messaggio di risposta con scritto “non modificato”.

Un messaggio di richiesta HTTP è detto di GET condizionale se presenta:

• Il messaggio di richiesta utilizza il metodo GET;

• Il messaggio di richiesta include una riga di intestazione IF-Modified-Since

**Trasferimento di file: FTP (File Transfer Protocol)**

In una sessione FTP, l'utente utilizza un host locale per trasferire file da o verso un host remoto (utilizza la porta 21 e non l’80). Per accedere deve essere autorizzato a scambiare tali informazioni con il file system remoto e quindi deve fornire username e password. Dopo di che può trasferire file tra i due file system.

L'utente interagisce con FTP tramite un agente software detto FTP User Agent fornendo inizialmente il nome dell'host remoto, in modo che il processo FTP client nell'host locale stabilisca una connessione TCP con il processo FTP server nell'host remoto e fornisce nome identificativo e password, inviate sulla connessione TCP come parte dei comandi FTP. Dopo di che, ottenuta l'autorizzazione può inviare i file dal file system locale a quello remoto.

**HTTP e FTP a confronto**

HTTP e FTP sono protocolli di trasferimento di file e condividono molte caratteristiche in comune, tra cui l'utilizzo della connessione TCP.

1. Tuttavia, FTP utilizza due connessioni TCP dette connessione di controllo e connessione dati

• connessione di controllo: viene utilizzata per inviare informazioni di controllo tra gli host come utente e password, per cambiare directory remote e comandi per inviare e ricevere file. La porta utilizzata è la porta 21;

• connessione dati: è la connessione su cui si inviano e ricevono effettivamente i dati; la porta utilizzata è la porta 20.

Data la proprietà per cui FTP utilizza una connessione di controllo per i dati viene detta fuori banda; mentre il protocollo HTTP è in banda poiché utilizza una sola connessione per i dati e per i dati di controllo.

1. I comandi GET e PUT in HTTP vengono sostituiti in FTP con RETR (per chiedere un file al server) e STOR (per prendere un file e memorizzarlo sul server). Inoltre in FTP vengono usati i comandi LIST (per chiedere al server di inviare un elenco di tutti i file nella directory remota corrente) ed i comandi USER (utilizzato per inviare l'identificativo dell'utente al server) e PASS (usato per inviare la password dell'utente al server)
2. Un'altra differenza con HTTP è che il protocollo FTP deve mantenere lo stato dell'utente. In particolare, associa la connessione di controllo ad uno specifico utente e tiene traccia della directory corrente dell'utente mentre quest'ultimo visita l'albero delle directory sull'host remoto al fine di limitando, di conseguenza, il numero totale di sessioni che FTP riesce a gestire contemporaneamente.

**2.3 Posta Elettronica**

Il servizio di posta elettronica è un altro esempio di protocollo applicativo. In particolare, è un protocollo client server, tuttavia, ha l’obiettivo di permettere lo scambio di messaggi tra utenti ma gli utenti non interagisco mai tra loro ma solo tramite il servizio di posta gestito dai server di posta che interagiscono tra loro. È formato da tre componenti:

• User Agent: permettono all'utente di leggere, rispondere, inoltrare, salvare e comporre i messaggi. Ne è un esempio Gmail.;

• Mail Servers: rappresentano il nucleo dell'infrastruttura delle mail ed in ognuna di queste, per ogni utente, vi è una mailbox con il compito di mantenere e gestire i messaggi che gli arrivano. Inoltre, il loro compito è quello di gestire le code dei messaggi formatasi quando il tentativo di invio dei messaggi non è andato a buon fine;

• Simple Mail Transfer Protocol (SMTP): è un protocollo client/server in cui gli utenti non interagiscono tra di loro ma con il server di posta elettronica comunicando tra i messaggi tra di loro tramite i Mail Server.

**SMTP**

Il protocollo SMTP è il principale protocollo di livello applicativo per le e-mail di Internet. Questo protocollo utilizza un servizio di trasferimento dati di TCP per inviare dal server le email del mittente alla mailbox del destinatario. La porta utilizzata è la porta 25 ed i messaggi che si inviano sono in 7-bit ASCII.

Inoltre, è formato da lato client (eseguito dal mail server del mittente) e lato server (eseguito sulla mailbox del server). Entrambi di questi lati sono in esecuzione sul server: quando il mail server invia e-mail ad un altro mail server si comporta da client, altrimenti si comporta da server. In particolare, si alternano tre fasi:

• handshaking;

• trasferimento di messaggi;

• chiusura della connessione.

**Esempio**

1.Alice invoca il suo User Agent per le e-mail fornendo l'indirizzo e-mail di Bob, compone il messaggio e dice all'user agent di inviare il messaggio;

2.L'user agent di Alice invia il messaggio al suo e-mail server, dove sarà messo nella coda di messaggi;

3.Il lato client di SMTP, in esecuzione sul server di Alice, vede il messaggio nella coda messaggi. Apre una connessione TCP sul server SMTP, in esecuzione sul mail server di Bob;

4.Dopo un certo tempo di handshaking, il lato client di SMTP di Alice invia il messaggio sulla connessione TCP;

5.Al mail server di Bob, il lato server di SMTP di Bob riceve il messaggio. Il mail server di Bob la posizione nella mailbox di Bob;

6.Bob invoca il suo user agent e legge il messaggio quando vuole.

Il protocollo SMTP non utilizza server intermediari tra i due utenti anche se questi si trovano a grandissima distanza. Nel caso in cui il server è down, il lato client SMTP prova l'invio del messaggio inseguito. Una volta che la connessione è stabilita, il server e il client attuano l'handshaking trasferendosi informazioni, tra cui l'indirizzo e-mail del mittente e del destinatario. Dopo questa fase, si invia il messaggio.

Inoltre, il protocollo SMTP si basa sul servizio di trasferimento dati affidabile di TCP per inviare il messaggio al server senza errore. Riapplicando questo procedimento per ogni messaggio da inviare al server, in alternativa si chiude la connessione.

**Confronto SMTP e HTTP**

1. Entrambi i protocolli sono usati per trasferire i file da un host all'altro. HTTP trasferisce i file da un Web Server ad un Web client; mentre SMTP trasferisce file da un mail server ad un altro. Quando si trasferiscono i file entrambi i protocolli utilizzano connessioni persistenti.
2. Inoltre, HTTP è un protocollo pull poiché qualcuno carica informazioni su un Web server e gli utenti utilizzano HTTP per prelevare le informazioni del server quando vogliono: la connessione TCP è inizializzata dalla macchina che vuole ricevere il file (HTTP chiede dati); mentre il protocollo SMTP, è un protocollo push: il mail server mittente esegue un “push” del file al mail server ricevente. La connessione TCP è iniziata dalla macchina che vuole inviare il file (SMTP invia dati).
3. Il protocollo SMTP richiede che le mail siano scritte in carattere a 7-bit ASCII mentre HTTP in ASCII ma non con il vincolo di 7-bit.
4. In conclusione, HTTP incapsula ogni oggetto in un messaggio di risposta HTTP; SMTP, al contrario, tutti gli oggetti vengono messi in un messaggio.

**Formato messaggi mail**

Quando un messaggio e-mail è inviato da una persona all'altra, vi è un header che contiene le informazioni che precedono, separato da una linea vuota, il body del messaggio stesso.

In particolare, contiene una serie di linee di intestazioni contenenti:

1. Header composto da: From, chi è l'utente che invia la e-mail, To, a chi è indirizzata l'e-mail, Subject, cioè la specifica del contenuto dell'e-mail;
2. Riga vuota
3. Corpo del messaggio scritto in ASCII

MIME: consente di specificare il formato del messaggio così da includere, talvolta, anche allegati multimediali (codificati all'interno del body). In caso di oggetti di vario tipo (text, image, audio, video) si utilizza la specifica multi part type.

**Mail Access Protocoll**

Consideriamo ora il percorso che prende un'e-mail quando viene inviata ad un User Agente. In un qualche punto lungo il percorso, l'e-mail deve essere depositata. Il protocollo SMTP è l'incaricato a questo compito. Infatti, come già discusso, ha il compito di fare il push delle e-mail tra mail servers mittente e destinatario.

In particolare, l'User Agent mittente utilizza SMTP per inviare il messaggio nel proprio mail server ed è il mail server che utilizza nuovamente SMTP, come client, per consegnare l'e-mail all'User Agent destinatario.

L'User Agent destinatario non può utilizzare il protocollo SMTP per ottenere il messaggio poiché questa operazione è un'operazione pull, mentre SMTP è un protocollo push. A tale scopo si introducono tre protocolli di accesso alle e-mail: POP3, Access Protocol (IMAP) e HTTP.

**POP3 - Post Office Protocol**

Il protocollo POP3 è un protocollo molto semplice, ma dalle funzionalità limitate. Inizia quando l'User agente apre una connessione al mail server sulla porta 110. Quando la connessione TCP viene stabilita, POP3 segue 3 fasi:

• Autorizzazione: l'user agent invia, in chiaro, username e password per essere autenticato. I comandi utilizzati sono: user <username> e pass <password>;

Una volta autenticato si passa alla prossima fase;

• Transazione: l'user agent preleva i messaggi, può selezionare i messaggi da eliminare e ottenere le statistiche di e-mail; ad ognuno di questi comandi il server risponde con due possibili opzioni: +OK: usato dal server per indicare che il precedente comando è andato a buon fine oppure -ERR: usato dal server per indicare che qualcosa è andato storto con il precedente comando. Inoltre, questa fase può essere configurata per download and delete o download and keep. La sequenza dei comandi istanziati dall'user agente dipende su quali modi si sta operando:

-download and delete mode: l'user agent utilizzerà i comandi list, retr, delete, quit. Dal lato del mail server, una volta ricevuto il comando quit eliminerà tutti i messaggi.

-download and keep: l'user agente utilizzerà i comandi precedenti ad eccezione del delet. I messaggi rimarranno salvati.

• Update: Il mail server elimina i messaggi segnati per la cancellazione;

Questo protocollo non mantiene informazioni sullo stato dell'user agent.

**IMAP**

IMAP è un protocollo di accesso mail, possiede più funzionalità di POP3, ma molto più complesso. In IMAP il server mantiene le caselle di posta in arrivo e tutti i folder degli utenti all’interno del server stesso. Un server IMAP associa ad ogni messaggio una cartella, quando un messaggio arriva al server viene associato nella cartella INBOX. Talvolta, la mailbox può spostare l'e-mail in una cartella creata dall'utente, leggerla, eliminarla.

Il protocollo IMAP fornisce dei comandi che permettono all'utente di creare cartelle e muovere i messaggi tra le cartelle, di cercare messaggi che corrispondono ad uno specifico criterio e, inoltre, si mantengono informazioni sullo stato dell'utente lungo la sessione.

**HTTP**

Quando un utente vuole accedere alla propria mailbox, i messaggi vengono spediti dal mail server al proprio browser utilizzando il protocollo HTTP.

Se volessimo inviare un messaggio, quest'ultimo è inviato dal browser al mail server, dopo di che si utilizza il protocollo SMTP come visto fino ad ora.

**2.4 DNS - Domain Name System**

Il DNS nasce dall'esigenza di identificare gli host e concilia l’approccio identificativo effettuato con il nome e con l’indirizzo IP: infatti, il DNS permette di tradurre i nomi degli host nei loro indirizzi IP.

Un identificatore per un host, più conosciuto, è l'hostname (per esempio www.facebook.com), generalmente alfanumerico e forniscono informazioni circa la posizione dell'host in internet. Poiché gli hostname essendo alfanumerici e di lunghezza variabile non si prestano al routing che deve fare un router per indirizzare hosts.

A tale scopo introduciamo gli indirizzi IP. Essi sono indirizzi gerarchici, poiché ogni periodo che va da 0 a 255 fornisce informazioni aggiuntive sulla locazione del host, formati da 32 bit.

Per avere un mapping tra hostname ed indirizzo ip, occorre un servizio di directory che traduce gli hostname in indirizzi IP. La soluzione a questo problema è data dal domain name system DNS.

Il DNS è un database distribuito (e non centralizzato), che utilizza il protocollo UDP, in una gerarchia di DNS servers e un protocollo di livello applicativo che permette agli host di eseguire query al database distribuito*. La traduzione viene effettuata nel seguente modo:*

*• L'utente esegue il lato client dell'applicazione DNS;*

*• Il browser estrae l'hostname dall'URL e lo passa alla parte cliente dell'applicazione DNS;*

*• Il client DNS invia una query contenente l'hostname al DNS server;*

*• Il DNS client riceve una risposta che include l'indirizzo IP per l'hostname;*

*• Una volta che il browser riceve l'indirizzo IP dal DNS, può iniziare una connessione TCP al processo server HTTP localizzato sulla porta 80 dell'indirizzo IP.*

**Servizi forniti dal DNS**

Il protocollo DNS fornisce i seguenti servizi:

• Host aliasing: un host con un nome complicato può avere uno o più nomi che riferiscono allo stesso indirizzo IP;

• Mail server aliasing: Il protocollo DNS può essere invocato da un'applicazione mail per ottenere dall'hostname l'indirizzo IP;

• Load distribution: DNS è utilizzato anche per implementare una distribuzione di carico lungo i server replicati.

Un'implementazione effettiva di DNS potrebbe essere quella di centralizzare tutte le traduzioni degli indirizzi su un unico DNS server. Tuttavia, ciò è irrealizzabile. Infatti, le problematiche sono le seguenti:

• Single point failure: Se l'unico server DNS va in crash, lo fa anche tutto Internet;

• Traffic Volume: Un unico server DNS dovrebbe gestire tutte le query, per tutte le richieste HTTP, per i messaggi etc..

• Distant centralizer database: Un singolo server DNS non può essere vicino a tutti i client che inviano query DNS;

• Manutenzione: L'unico server DNS dovrebbe tenere traccia di tutti i record di tutti gli host internet. Oltre ad essere di grandi dimensioni, dovrebbe aggiornare frequentemente ogni host.

La soluzione a cui si è arrivati, è la realizzazione di un database distribuito gerarchicamente.

**Database distribuito gerarchicamente**

Al fine di gestire il problema di scalabilità, il DNS utilizza un grande numero di server (nessuno dei quali possiede la mappa di tutti gli indirizzi IP), organizzati in maniera gerarchica distribuiti lungo tutto il mondo. Vi sono tre classi di indirizzi server:

• root DNS server: forniscono l'indirizzo IP per i server TLD;

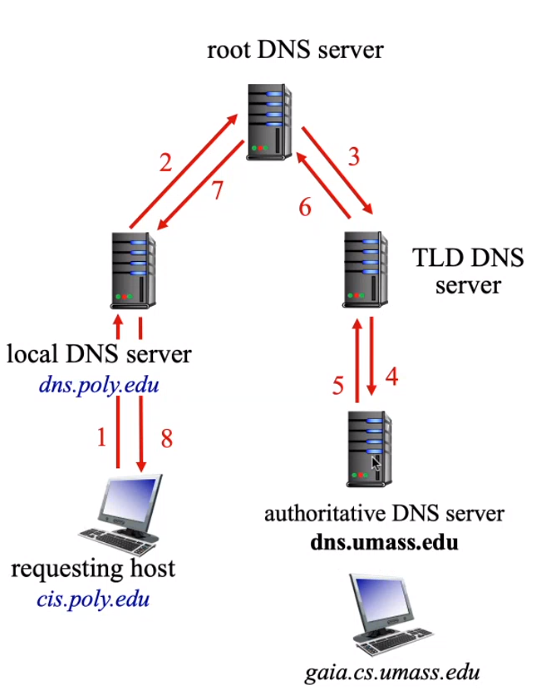
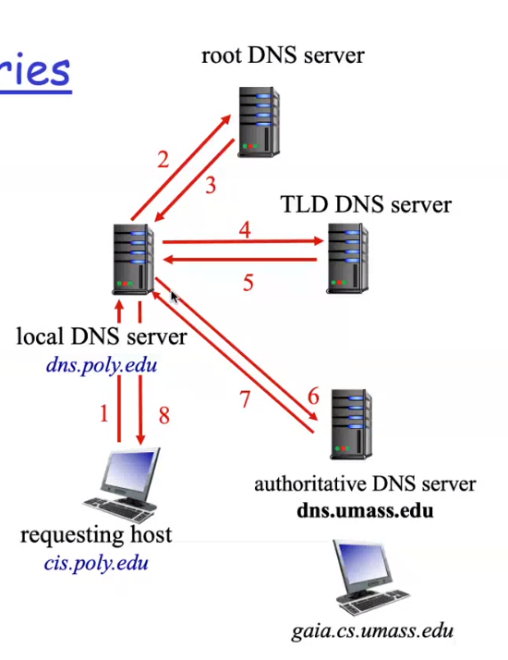
• top-level-domani DNS (TLD): sono del tipo .com/.uk e forniscono l'indirizzo IP ai server autoritari;

• authoritative DNS server: sono utilizzati per gli enti pubblici.

• local DNS server: non fanno parte necessariamente delle gerarchie, ma sono forniti dagli ISP (università, azienda) che possono salvarsi gli hostname per velocizzare il loro servizio.

**Esempio**

Supponiamo che l'host desidera l'indirizzo IP di una pagina. Ci sono due modi per agire: Query ricorsiva e Query iterativa.

1. Nella prima l’host contata il suo DNS locale, che contatta il root server se necessario che contatta il server TPD se necessario il quale contatta il DNS server autoritario se necessario ed infinite ritorna l'indirizzo IP all'host richiedente (nota che utilizzando il caching il traffico viene molto diminuito). Come abbiamo visto nell'esempio, in generale, non potremmo conoscere il nome del server autoritario, ma possiamo conoscere il server intermediario che deve poter risalire al nome del server autoritario. Dunque, in quella ricorsiva si richiede la risoluzione del nome contattando i diversi server e affidando loro ogni volta il compito di risolvere la corrispondenza e ciò può creare alto carico nei nodi interpellati.
2. Nella seconda l’host contatta il suo DNS locale che contatta il root server e qui cambia come il root server gestisce la richiesta. Infatti, stavolta restituisce l’indirizzo del TLD server da contattare e poi è il DNS server che deve contattare il TLD e se necessario poi contatterà lui direttamente il DNS autoritativo. In altre parole, la query dall'host richiedente al server DNS è ricorsiva e le rimanenti query sono iterative ed è il DNS che si occupa di gestire la corrispondenza.

**DNS Caching**

DNS sfrutta intensivamente il DNS caching per migliorare le performance sul ritardo e ridurre il numero di messaggi DNS in Internet. L'idea è molto semplice: in una query a catena, quando un server DNS riceve una risposta può salvarsi quel mapping tra hostname ed indirizzo in una propria cache.Se l'hostname/Indirizzo IP è in cache nel server DNS ed arriva un'altra query per lo stesso hostname, il server DNS può fornire l'indirizzo IP desiderato anche se non è autoritario.

Dal momento che gli hostname e gli indirizzi IP mediamente non sono permanenti, i server DNS scartano le informazioni in cash dopo un certo periodo di tempo.

**DNS Records and Messages**

DNS Records

Il DNS è un database che memorizza i Results Records (RR) ossia i record di risorsa che forniscono gli hostname all'indirizzo IP. Essi hanno la seguente forma:

(Name, Value, Type, TTL)

• TTL: è il tempo di vita del record di risorsa. Determina quando la risorsa dovrebbe essere rimossa dalla cache;

• Il significato di Name e Value dipende dal Type che specifica che tipo di informazione contiene:

Se Type=A, allora Name è l'hostname e Value è l'indirizzo IP dell'hostname;

Se Type=NS, allora Name è il dominio e Value è l’IP del server autoritativo DNS di quel dominio;

Se Type=CNAME, allora Value è un alias per il nome canonico della macchina;

Se Type=MX, allora Value è il nome canonico di un mail server associato a quel nome;

DNS Messages

I messaggi di richiesta DNS possono essere di risposta o di richiesta ed hanno entrambi lo stesso formato. Importante è che il DNS è un protocollo che impiega UDP e non TCP e la porta 53. Ogni messaggio è formato da:

• La prima sezione di intestazione da 12 bytes è formato da:

1. Un campo a 16 bit per l'identificazione della query per permettere al client di confrontare le risposte ricevute con la query che ha inviato;
2. Numerosi flag:

-flag di query/risposta: indica se il messaggio è una query o una risposta;

-flag autoritario: viene impostato in un messaggio di risposta quando un server DNS è un server autoritario;

-flag di ricorsione: viene impostato quando un client desidera che il server DNS attui una ricorsione quando non si ha il record dell'hostname;

-flag di ricorsione disponibile: viene impostato se il server DNS supporta la ricorsione;

3. Ulteriori quattro campi che indicano:

• La sezione question contiene informazioni sulle query che saranno effettuare. Include:

-un campo nome contenente il nome che viene richiesto;

-un campo tipo che indica il tipo di richiesta che si sta facendo;

• La sezione di risposta viene riempita in una risposta da un server DNS. Contiene i record di risorsa per il nome che è stato originariamente chiesto, specificando Type, Value, TTL.

• La sezione autoritaria contiene i record di altri server autoritari;

• La sezione addizionale contiene altri record aggiuntivi.

**Insert Records into the DNS Database**

La prima cosa che occorre fare quando si crea una compagnia è quello di registrare il dominio al register. Essa è un'entità commerciale che verifica l'unicità del nome del dominio, inserisce il nome del dominio nel database DNS e chiede una tassa per questo servizio.

Inoltre, occorre fornire al register il nome e gli indirizzi IP dei server DNS autoritari primari e secondari. A quel punto questa entità si assicura che i record di Tipo NS e Tipo A siano inseriti nel server Top-Level-Domain TDL; d'altra parte, la compagnia deve accertarsi che i record di risorsa di Tipo A del Web Server e di Tipo MX del mail server siano inseriti nel server DSN autoritario.

**2.5** **Peer-to-Peer (P2P)**

In una architettura P2P, non vi è garanzia di presenza di server sempre accesi. Al contrario, si hanno coppie intermittenti di host connessi, detti peer, che comunicano direttamente tra di loro.

Supponiamo di avere un’applicazione P2P che permette la distribuzione di file di grandi dimensioni da un singolo server a molteplici server. Quindi, ogni peer può ridistribuire qualsiasi porzione del file che ha ricevuto su qualsiasi altro peer. Così facendo, si supporta il compito del server di distribuire questo file. Affrontiamo due argomenti: la distribuzione dei file e il modo di cercare informazioni:

1. **Distribuzione File**

Per confrontare un'architettura Client-Server con una peer-to-peer, consideriamo un modello di distribuzione di un file con un numero fissato di peer per entrambe le architetture.

Il server e i peers sono connessi ad internet con i link. Definiamo:

- rate di upload del link di accesso del server u\_s;

- rate di upload del link di accesso dell'i-esimo peer u\_i;

- rate di download del link di accesso dell'i-esimo peer d\_i;

- dimensione del file F;

- il numero di peer che voglio ottenere la copia N;

Il tempo di distribuzione è il tempo speso per ottenere la copia del file a tutti i peer.

Per entrambe le architetture, possiamo semplificare il modello assumendo che:

-Il nucleo di Internet possiede abbastanza bandwidth e tutti i colli di bottiglia si trovano nella rete di accesso;

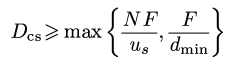
-Il server e i client non partecipano in nessun'altra applicazione di rete. Quindi tutti i loro upload e download sono totalmente impiegati nel file.

1- Determiniamo il tempo di distribuzione per l'applicazione client-server D\_cs. In questo scenario, nessun peer contribuisce alla distribuzione del file. In particolare:

• Il server deve trasmettere una copia del file ad ogni peer. Trasmettendo così N∗F bits. Dato che il rate di upload del server è u\_s, impiegheremo (N\*F)/(u\_s) secondi;

• Definiamo d\_min il rate di download del peer con il più basso rate di download d\_min={d\_(1,)d\_2,…,d\_N}. Quest'ultimo non può prendere il file in meno di F/(d\_min) secondi.

Unendo queste considerazioni, possiamo scrivere il tempo di distribuzione come:



(Ossia il tempo che ci mette il server ad inviare il file N volte e quanto ci mette a scaricarlo l’utente più lento)

Quindi il tempo cresce linearmente con il numero dei peers.

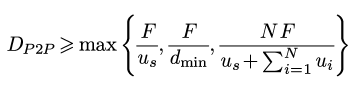
1. In una architettura P2P, ogni peer può contribuire nella distribuzione del file. In particolare, quando un peer riceve qualche dato, lo può usare con la propria capacità di upload per ridistribuirlo ad altri peer. Determiniamo il tempo di distribuzione del file:

• All'inizio della distribuzione, solo il server possiede il file. Per prendere questo file nella community di peer, il server deve inviare ogni bit in almeno una volta nel suo access link. Inoltre, il tempo di distribuzione minima è almeno F/(u\_s);

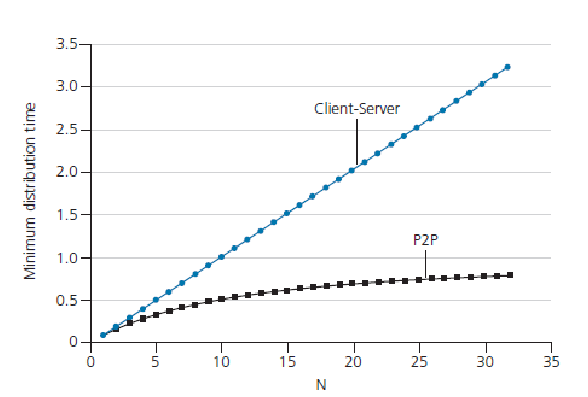
• Il peer con il minor download rate non può ottenere tutti gli F bit del file in meno di F/(d\_min) secondi. Il tempo di distribuzione è almeno F/(d\_min);

• La capacità totale di upload del sistema per intero è uguale al rate di upload del server più i rate di upload di ogni peer. Quindi, il sistema deve consegnare F bit in ogni N peer, consegnando in totale N\*F ad un rate non maggiore di u\_tot. Il tempo di distribuzione è almeno

Unendo queste considerazioni, possiamo scrivere il tempo di distribuzione come:



(Ossia il tempo che ci metto ad inviare il file almeno una volta, il tempo che ci mette l’utente più lento e quanto tempo ci mette la totalità di bit che vanno ricevuti dagli utenti diviso la capacità totale di upload di rete usata per rendere disponibili i bit).

Ciò dimostra che all'aumentare del numero di utenti, il tempo di distribuzione non cresce linearmente, provando la scalabilità di un'architettura P2P conseguenza del fatto che i peer distribuiscono anche il file.

BitTorrent è un'applicazione che utilizza il protocollo P2P. Si ha un file detto Torrent che contiene i metadati su un insieme di file folder che dovranno essere distribuiti (quali sono i file, quanto sono grandi) e questi torrent vanno ottenuti dagli utenti. Ci sarà un’individuazione di un tracker, ossia dei server che hanno la funzione di tenere traccia dei peer che stanno condividendo un torrent. L’insieme di utenti che in un istante stanno scaricando/condividendo descritti all’interno di un torrent sono detti swarm. Quindi quando un utente è interessato ad un certo torrent, deve scoprire un tracker per quel torrent, collegarsi ad esso, collegarsi con diversi peer e scambiarsi i file. L’idea di come scambiare i dati, a chi chiederli e a chi inviarli, è questa:

Il file è diviso in porzioni piccole detti chunks di dimensione fissa 256kb. Un peer che si unisce allo swarm non ha nessun chunks ma li accumula piano piano nel tempo e registrandosi sul tracker si connette ad altri peer. Mentre scarica i chunks, li mette a disposizione degli altri peer e una volta che ha l’intero file può uscire o rimanere contribuendo all’upload del file, infatti, si può uscire e rimanere in qualsiasi momento.

Per ottimizzare le prestazioni dell’applicazione si utilizza la tecnica del “il più raro prima” ossia si mandano i chunk più rari prima in modo che diminuisce la possibilità che quel chunk scompaia nel caso in cui un peer uscisse.

Inoltre, occorre decidere a quali richieste i peer devono rispondere. A tale scopo, BitTorrent utilizza un algoritmo in cui si dà priorità ai 4 vicini che con cui stiamo scaricando al rate massimo. Ogni 10 secondi, si ricalcola i rate e il possibile insieme di 4 peer e ogni 30 secondi, si sceglie un vicino addizionale randomicamente e invio i chunks: se i due peer sono soddisfatti con lo scambio, si metteranno a vicenda nella top quattro peers e continueranno a scambiarsi i chunk finché saranno compatibili.

1. **Ricerca di informazioni**

Nelle applicazioni P2P devo avere un indice di quali utenti condividono un determinato oggetto e un indice della posizione di un utente. Quindi nelle reti P2P, a differenza delle client-server in cui se ho il nome del servizio, tramite il DNS contatto il server e posso ricercare informazione, non so quali sono gli utenti che partecipano ad un certo istante all’applicazione e quali informazioni stanno condividendo. Per risolvere questo problema sono state create diverse soluzioni:

1. Napster: applicazione in cui i peer si connettono al server comunicando il loro ID e i loro contenuti. Quindi abbiamo sia una parte client-server con tutte le info sugli utenti sia una parte P2P dove gli utenti si contattano tra loro per scambiarsi file. Il problema però è che è centralizzato in un solo server e deve gestire tutte le informazioni.
2. Query Flooding: (tecnica usata da Gnutella) elimino il server centralizzato e ogni utente mantiene localmente un indice solo dei file che invia. Al posto del server centralizzato c’è un server che riceve solo l’indirizzo IP di ogni utente appena si connette e invia a ciascuno di questi la lista intera degli IP. Gli utenti così si contattano tra loro in modo da formare una specie di rete dove ogni utente è connesso solo ad alcuni utenti. L’idea è quindi di inviare la richiesta di un oggetto a quelli connessi vicini che, se non possiedono la risposta, la riinoltrano agli altri utenti connessi vicini e così via fino a che non si ottiene la risposta.
3. Soluzioni Gerarchiche: hanno un costo più basso delle precedenti ed è una via di mezzo tra una soluzione totalmente centralizzata e totalmente distribuita. L’idea è che alcuni nodi della rete assumono il ruolo di super peer che hanno il compito di essere contattati da un peer normale e ricevere la sua lista dei file. Ogni peer quindi è collegato ad un solo super peer e tutti i super peer sono collegati tra loro come in una rete P2P. Il vantaggio è che faccio richieste solo tra super peer, i quali hanno tutte le info dei peer collegati a loro, e quindi le richieste sono diminuite notevolmente rispetto alle soluzioni precedenti.