1 Lezione del 03-10-25

1.1 E-mail

L'**e-mail** è un'altra delle applicazioni di largo uso sviluppate su Internet. Come ogni altra applicazione, è supportata da diversi *protocolli*, fra cui **SMTP** (*Simple Mail Transfer Protocol*), **IMAP** (*Internet Message Access Protocol*), **POP** (*Post Office Protocol*, di cui la più nota è la versione *POP3*).

Nel sistema della posta elettronica ci sono 3 componenti principali:

- Gli **user agent**, cioè i client di posta elettronica usati dagli utenti per scrivere e ricevere messaggi di posta elettronica;
- I **server** di posta elettronica;
- Un **protocollo** di posta elettronica, prendiamo SMTP.

I mail server supportano:

- Una **mailbox** che contiene i messaggi in arrivo per ogni utente;
- Una **coda** di messaggi in uscita: questi vengono poi smistati nelle mailbox dei vari utenti;
- Il **protocollo** SMTP per ricevere i messaggi dai client e inviarli ad altri mail server (vedremo che i client non usano direttamente SMTP per richiedere la posta, ma sfruttano altri protocolli).

Come sempre, notiamo che con *mail server* si intende nello specifico il processo che gestisce le richieste degli utenti (e per estensione tutta la macchina (o macchine) che lo eseguono).

Il protocollo SMTP è un protocollo client-server: nonstante ciò, il mail server può comunicare con altri server, e in tal caso si comporta sia come client che come server.

- Si comporta come *client* quando deve inviare posta;
- Si comporta come *server* quando deve ricevere posta.

Questo è chiaro in quanto è il server che *invia* la posta a dover iniziare la comunicazione, mentre il server che *riceve* deve essere solo pronto, appunto, a ricevere.

1.1.1 Protocollo SMTP

Nello specifico, il protocollo SMTP è basato su TCP aperto alla porta 25. Il trasferimento è diretto, e come abbiamo già detto il server che invia si comporta come client per il server che riceve.

Ci sono 3 fasi di trasferimento:

- Handshake fra i due server;
- Trasferimento di messaggi;
- Chiusura della comunicazione.

L'interazione è richiesta/risposta come in HTTP (anche se le richieste vengono dette *comandi*), con messaggi codificati in ASCII a 7 bit.

I comandi contengono solo testo ASCII, mentre le risposte contengono un codice di stato seguito da testo, sempre ASCII, in linguaggio naturale. Questo è per un motivo diagnostico: i comandi client sono letti dalla macchina, mentre del server la macchina legge solo il codice di stato e il commento ASCII è di debug per i tecnici.

Facciamo un'esempio pratico per capire meglio il funzionamento del protocollo. Poniamo che Alice voglia inviare un messaggio a Biagio:

1. Alice `alice@unipi.it` usa l'user agent per comporre un messaggio e-mail al destinatario Biagio `biagio@altroente.edu`;
2. L'user agent di alice invia il messaggio al mail server di `unipi.it` via SMTP, che lo mettte nella coda dei messaggi;
3. Il mail server di Alice apre una connessione TCP, comportandosi come client, con il mail server `altroente.edu`;
4. Il messaggio di Alice viene trasferito verso tale mail server sulla rete TCP;
5. Il mail server di Biagio mette il messaggio nella mailbox di Biagio;
6. In un secondo momento, l'user agent di Biagio richiede la mailbox al suo mail server, che quindi gli invia il messaggio di Alice.

Qui il protcollo SMTP viene usato nella comunicazione fra l'UA di Alice ai server di `unipi.it`, e nella comunicazione fra questi e i server di `altroente.edu`. Per portare la posta dai server di `altroente.edu` all'UA di Biagio, invece, verrà usato un altro protocollo (come già detto IMAP o POP3).

Abbiamo quindi, per riassumere un ultima volta, che i mail server si comportano come *client* quando devono *inviare* messaggi ad altri mail server, e come *server* quando devono *ricevere* messaggi da altri mail server. Verso gli user agent, invece, sono sempre server: si richiede al server di *inviare* una mail, e si richiede al server di *scaricare la mailbox* corrente (il server non ci contatterà come client per inviarci la posta come farebbe con un altro mail server, ma siamo noi a doverlo invocare).

1.1.2 Parole chiave SMTP

Vediamo l'esempio dei messaggi che viaggiano in una connessione TCP, in plain text, durante lo svolgimento del protocollo TCP. Mettiamo ad esempio di leggere la comunicazione fra il server `unipi.it` al servizio di Alice dell'esempio precedente, quando questo inoltra la posta al mail server `altroente.edu` di Biagio.

Qui `S:` significa server (`altroente.edu`) e `C:` significa client (`unipi.it`):

```
S: 220 altroente.edu
C: HELO unipi.it
S: 250 Hello unipi.it, pleased to meet you
C: MAIL FROM: <alice@unipi.it>
S: 250 alice@unipi.it... Sender ok
C: RCPT TO: <biagio@altroente.edu>
S: 250 biagio@altroente.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Ciao Biagio !
```

```
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 altroente.edu closing connection
```

Iniziamo con l'handshake: il server invia 220 (tutto ok) e il client si introduce con `HELO` e il suo indirizzo: a questo punto il server risponde con `250` (acknowledge dell'`HELO`).

Da qui in poi il client può comunicare la posta che vuole inviare. Con `MAIL FROM` si indica l'indirizzo di posta elettronica dell'utente che sta inviando posta, a cui segue una risposta 250 dal server che segnala se quell'utente è riconosciuto. Con `RCPT TO` si indica poi l'indirizzo di posta elettronica dell'utente destinatario, a cui segue un'altra risposta 250 dal server che segnala se quell'utente è presente sul mail server.

Segue poi la sezione `DATA`, a cui il server risponde con 354, e quindi la mail vera e propria. Alla fine del messaggio (chiuso con un . su una nuova linea) il server risponde con un'altro 250, e il client chiude la connessione con `QUIT`. L'ultimo acknowledge è del server con 221 (chiudo connessione).

1.1.3 Confronto fra SMTP e HTTP

Si può dire che HTTP è un protocollo di tipo **pull**, mentre SMTP è un protocollo di tipo **push**: in HTTP lato client si richiedono risorse, in SMTP si inviano. Inoltre, in SMTP si usano connessioni *persistenti*.

Abbiamo poi che in HTTP ogni oggetto è incapsulato in una risposta, mentre in SMTP si possono avere trasferimenti *multipart* (più messaggi di posta per connessione TCP), e che in SMTP si usa ASCII su 7 bit.

1.1.4 Protocolli di richiesta

Come abbiamo detto, il protocollo SMTP è effettivamente usato solo per la comunicazione fra mail server e per inviare messaggi a mail server: per richiedere la posta dal server si usano altri protocolli come **IMAP** (*Internet Messagge Access Protocol*) e **POP** (*Post Office Protocol*). Questi protocolli forniscono la possibilità di fare richieste di messaggi, eliminazione, accesso a directory di messaggi presenti sul server, ecc...

Inoltre, servizi come Gmail e Hotmail forniscono interfacce Web (via il protocollo HTTP) costruite su SMTP (per inviare messaggi) e IMAP (o POP, in particolare POP3) per richiedere messaggi.

La differenza principale fra IMAP e POP è che *POP* era pensato per singoli dispositivi: la posta veniva scaricata su locale e rimossa dal server, per cui non vi si poteva accedere da altri dispositivi. *IMAP* risolve questo problema mantenendo i messaggi sul server e concedendone l'accesso da parte di più dispositivi. In verità, oggi la maggior parte dei mail server è configurato per mantenere in remoto anche la posta scaricata via POP3. Per client locali questo significa che IMAP e POP3 distinguono fondamentalmente fra due politiche più o meno aggresive di caching, mentre i client su HTTP usavano in ogni caso IMAP (non si scarica mai nessun messaggio sul disco locale).

Per concludere, da quello che abbiamo appreso in 4.2.1, possiamo dire che POP è *stateless*, mentre IMAP è *stateful* (mantiene stato sotto forma di mail precedenti).

1.2 Il DNS

DNS (*Domain Name System*) è il sistema che usiamo per tradurre *nomi di dominio* DNS, semplici da ricordare per gli umani, in *indirizzi IP*, semplici da usare per le macchine.

1.2.1 Motivazione del DNS

Di base, un'**indirizzo IP**v4 (IP versione 4) è rappresentato da un numero a 32 bit, diviso in 4 numeri da 8 bit rappresentati come unsigned e separati da punti: 8.8.8.8 (il servizio DNS di Google) sarebbe `00001000_00001000_00001000_00001000`. Un certo numero di bit dal più significativo vengono detti **maschera** di rete, ed identificano la parte dell'IP che identifica la rete locale: i bit successivi identificano gli host. Ad esempio, `10.104.111.163/24` significa che i primi 24 bit identificano la rete, e i successivi 8 gli host su quella rete.

Il problema è che gli indirizzi IP sono difficili da ricordare: gli umani preferiscono indirizzi leggibili come `tizio.caio.com`, cioè i cosiddetti **nomi di dominio**.

1.2.2 Realizzazione del DNS

Per tradurre da indirizzi leggibili a indirizzi IP (necessari per aprire connessioni TCP o UDP e quindi comunicare effettivamente col server) si usa il DNS.

I problemi del DNS sono quindi:

1. Mantenere l'**univocità** dei nomi di dominio verso gli indirizzi IP;
2. Permettere di risalire (**risolvere**) da un nome DNS all'indirizzo IP corrispondente.

La soluzione che è stata data è di creare un *database distribuito* implementato con una gerarchia di tanti *name server* che contengono **registri** di nomi di dominio. Protocolli a livello application permettono quindi agli host (e a loro volta i name server) di **risolvere** nomi di domini DNS in indirizzi.

Questo rappresenta una funzione base di Internet, implementata al livello application, e che distribuisce la maggior parte della complessità all'*"edge"* di internet.

Per assicurare l'univocità si sono formate *autorità* come **ICANN** (*Internet Corporation for Assigned Numbers and Names*), e la sussidiaria **IANA** (*Internet Assigned Numbers Authority*) che definiscono quali domini possono essere usati in base all'area di applicazione, geografica, ecc... Queste autorità delegano quindi ad altre autorità il compito di assegnare nomi, i cosiddetti identificatori *top-level*, **TLD** (*Top Level Domain*) a determinate regioni geografiche (`.uk`, `.io`, ecc...) e gestirne i domini, e la cosa può chiaramente ripetersi ricorsivamente, fino al cosiddetto livello *autoritativo*, dove la traduzione in IP può effettivamente accadere.

In particolare, l'autorità di registrazione dell'identificatore top-level (`.it`) risiede a Pisa all'interno del CNR.

Come sappiamo, esistono anche domini top-level associati non ad entità geografiche ma generici, come `.com`, `.org`, ecc... Questi sono amministrati direttamente dall'ICANN, mentre i registri sono detenuti da compagnie private.

1.2.3 Funzionamento del DNS

La risoluzione dei nomi di dominio viene fatta in maniera simile a quella degli indirizzi IP, partendo da destra verso sinistra per risalire al name server che contiene l'IP cercato. Ad esempio, nel DNS `www.google.com`, prima si controlla il `com` per capire il primo name server da contattare. Questo a sua volta prendera il `google` per contattare il prossimo name server, e così via. Il `www` è arcaico e viene mantenuto principalmente per ragioni storiche.

Abbiamo quindi che i problemi prima nominati vengono risolti rispettivamente nei seguenti modi:

1. L'*univocità* dei nomi di dominio è assicurata da **autorità** nazionali ed internazionali (ICANN, IANA, ecc...);
2. La risoluzione da un nome di dominio DNS all'indirizzo IP corrispondente è assicurata da **name server** che contengono **registri** di nomi di dominio.

Non è detto che le autorità che gestiscono un TLD gestiscono anche il registro corrispondente (abbiamo detto che questo non è il caso per i TLD generali come `.com`).

1.2.4 Risoluzione DNS

Vediamo i dettagli tecnici. Un server DNS comunica con un attraverso protocollo coi client, fornendo i servizi:

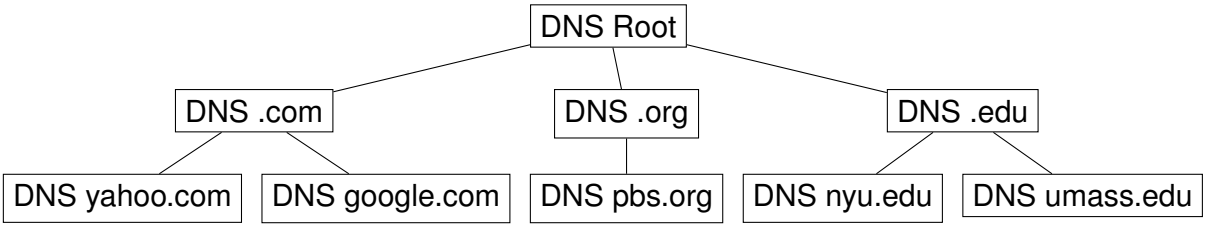
- **Risoluzione** da dominio a indirizzo IP;
- **Aliasing** degli **host**: più nomi possono puntare allo stesso IP;
- **Aliasing** dei server di **posta**: allo stesso dominio possono corrispondere web server come mail server;
- **Distribuzione** del **carico**: più server (con i loro IP) possono rispondere allo stesso nome di dominio per dividersi le richieste.

Abbiamo detto che il DNS è un database **distribuito**, quindi sviluppato gerarchicamente su più *name server*. Questa soluzione è stata scelta, al contrario di un database *centralizzato*, per una serie di motivi:

- Evita un singolo punto di fallimento;
- Riduce il volume di traffico su ogni name server;
- Il database centrale risulterebbe molto distante per larga parte degli utenti (distribuiti su tutto il mondo);
- La manutenzione è più semplice (si fa su unità più piccole).

1.2.5 Database DNS

La struttura gerarchica del DNS è modellizzabile come una serie di registri che rimandano l'uno all'altro:



I registri ad ogni livello sono raggruppati in diverse categorie:

1. **Root**, il registro base che punta agli altri registri;
2. **TLD** (*Top Level Domain*), che contiene i domini top-level nazionali (`.jp`, `.it`, ecc...) o generici (`.org`, `.edu`, ecc...);

3. **Autoritativi**, i server DNS che si raggiungono dai TLD e che hanno effettivamente il compito di terminare la risoluzione restituendo un IP.

Quando un client vuole risolvere un DNS, ad esempio per `www.google.com`, compie i seguenti passi:

1. Si rivolge al root server per trovare il server DNS `.com`;
2. Si rivolge al server DNS `.com` per trovare il server DNS `google.com`;
3. Si rivolge al server DNS `google.com` per ottenere l'IP di `www.google.com`.

In verità, la situazione è più complicata: i server Root sono più di uno (13), e sono largamente replicati (più di 200 solo negli Stati Uniti). Questo è fondamentale in quanto la sicurezza del servizio DNS deve essere assicurata.

Addirittura, i server sia TLD che autoritativi vengono replicati, usando configurazioni *primario/secondario* o meccanismi come *anycast* per assicurare la ridondanza del servizio.

1.2.6 Server DNS locali

Ogni ISP fornisce solitamente un server DNS proprio, detto anche *Default Name Server*. Anche i DNS locali sono solitamente configurati con name server *primario/secondario* da contattare nel caso l'uno o l'altro fallisca.

Quando un host fa una richiesta DNS ad un server DNS locale, questo controlla la sua cache, comportandosi quindi da proxy, e inoltrando la richiesta al DNS Root (o molto più tipicamente a un servizio di risoluzione DNS, sopratutto nel caso di router che per motivi economici non implementano l'intero protocollo DNS) solamente se non è capace di soddisfare la richiesta da solo.

Il meccanismo di caching è supportato dalla struttura gerarchica dei nomi di dominio DNS: ad esempio se il DNS locale conosce `dns.nyu.edu`, potrebbe rispondere alle richieste di risoluzione di `engineering.nyu.edu` inviando richieste direttamente al server autoritativo `nyu.edu`, che ha già contattato.

Abbiamo quindi che un host connesso ad un ISP (come un comune router) si rivolge solitamente al servizio DNS fornito da tale ISP per le sue richieste DNS. Questo quindi usa un servizio di risoluzione pubblico come 1.1.1.1 (Cloudflare) o 8.8.8.8 (Google) per risolvere il DNS (si dice che si comporta da *forwarder* DNS), o in alcuni casi si comporta esso stesso come risolutore DNS contattando direttamente il server Root.

Nessuno ci nega (e a volte avviene) che l'host eviti il DNS locale rivolgendosi direttamente a un servizio di risoluzione.

1.2.7 Risoluzione iterativa/ricorsiva

La risoluzione dei nomi di domnio DNS da parte di un server DNS locale che si comporta come risolutore DNS (come da parte di un risolutore pubblico come quelli nominati sopra) può accadere in due modi principali: **iterativo** e **ricorsivo**:

- **Iterativo**: in questo caso il server DNS si rivolge ad altri server DNS accettando sia traduzioni complete che riferimenti ad altri server DNS: nel caso il server contattato non possa tradurre potrà almeno reindirizzarci verso un server che può farlo;

- **Ricorsivo**: in questo caso il server DNS accetta solo traduzioni complete, o al limite messaggi che segnalano il fallimento della richiesta. Questa è la modalità in cui vengono interrogati la maggior parte dei servizi di risoluzione pubblici.