ARCHITETTURA DELLE RETI DI CALCOLATORI 2: Application Layer

Iuri Frosio

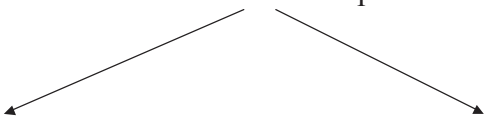
Lab. MAVR

frosio@dsi.unimi.it

Application layer 1

Network applications → ragione d'essere della rete !

Comunicazione tra processi



All'interno dell'host:
Regolate dall'OS

Tra hosts diversi:
Network e protocolli

Le network applications hanno application layer protocols che definiscono il formato e l'ordine dei messaggi scambiati tra processi, nonchè le azioni da intraprendere alla ricezione / trasmissione di un messaggio.

Network applications 1

Network application	:	WEB
Scopo	:	ottenere documenti sul web attraverso una richiesta
Componenti	:	- Standard per il formato dei documenti (HTML) - Web browser (Explorer, …) - Web servers - application layer protocols (HTTP, HyperText Transfer Protocol)

Componenti dell' application layer 1: application layer protocol

A cosa serve un application layer protocol?

1) Tipo di messaggi scambiati;
2) Sintassi dei messaggi scambiati;
3) Semantica dei fields;
4) Regole per determinare come e quando mandare messaggi.

Vi sono diversi application layer protocols:

- Public (es. HTTP);
- Proprietary (es. Internet Phone Products).

Componenti dell'application layer 2: client & server

Ogni application layer protocol prevede due "lati":

1) Client Side

2) Server Side

Es. Web browser → implementa client side dell'HTTP

Web server → implementa server side dell'HTTP

Definiamo: CLIENT = HOST CHE INIZIA LA SESSIONE DI COMUNICAZIONE

SERVER = L'ALTRO!

Componenti dell' application layer 3: socket

PROCESS SOCKET ~ Porta del processo, è l'interfaccia tra l'application layer e il transport layer all'interno di un host. Detto anche API (application layer interface) tra applicazione e rete.

Applic. → Socket → TCP → Internet → TCP → Socket → Applic.

CLIENT SERVER

Developer → controlla interamente il layer sul lato application; sul lato transport può solo scegliere il protocollo (TCP, UDP) ed un paio di parametri (max buffer dimension, max segment size).

Componenti dell'application layer 4: addressing process

Per comunicare tra PROCESSI diversi, sono necessari:

1) Addres (Name) dell'host machine su cui gira il processo;

2) Identifier che specifichi il processo sull'host

1 → 32 bit IP address, identifica UNIVOCAMENTE un end system (meglio: un'interfaccia) nella rete. Deve essere assegnato globalmente.

2 → RECEIVE SIDE PORT NUMBER, es. web server → porta 80, mail server → porta 25… Le porte standard sono in RFC 1700.

Componenti dell' application layer 5: user agents

USER AGENT = Interfaccia utente / network application (es. Netscape Navigator).

Riassumendo:

Utente → User Agent → Network Application → Socket → Internet

HTTP TCP, UDP

La scelta del protocollo 1

Quale protocollo di trasporto (TCP, UDP, proprietary) scegliere?

1) DATA LOSS:	e-mail, file transfer, … → data loss = 0%;
	multimedia applications → data loss <p%;
2) BANDWIDTH:	Internet telephony → rate = 32kbs (bandwidth sensitive);
	e-mail → rate >0kbps (elastica applications);
3) TIMING:	on line games, videoconferenze → delay < p ms;
	e-mail, web browser → delay < p s

La scelta del protocollo 2: TCP

Transmission Control Protocol (TCP) services:

1) CONNECTION ORIENTED SERVICE: handshking client/server, connessione TCP tra i due socket, client e server possono inviarsi messaggi contemporaneamente (full duplex connection);
2) RELIABLE DATA TRANSFER: trasferimento di tutti i dati, senza errori, nel giusto ordine.

TCP non garantisce:

1) MINIMO TRANSMISSION RATE: congestion control mechanism abbassa il rate in caso di traffico intenso;
2) MASSIMO RITARDO: decine di secondi per scaricare un file…

La scelta del protocollo 3: UDP

User Datagram Protocol (UDP) services:

1) CONNECTIONLESS: unreliable data transfer (i dati possono non arrivare, arrivare in ordine errato, …);
2) TRANSMISSION RATE MINIMO: non essendoci un congestion control mechanism, i dati possono essere inviati al rate desiderato.

UDP non garantisce:

1) MASSIMO RITARDO: decine di secondi per scaricare un file…

Più adatto di TCP per applicazioni real time. Problema dei ritardi!

Il protocollo HTTP 1

- HyperText Transfer Protocoll (HTTP), per la navigazione sul web;
- implementato nel client program e nel server program;
- definisce la struttura dei messaggi ed il modo in client e server li scambiano;
- usa per il trasporto il protocollo TCP;

Il protocollo HTTP 2

Web Browser → Client Socket → TCP → Server Socket →Web server

HTTP HTTP

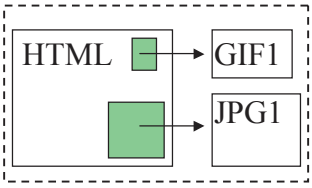
- Una volta fuori dal socket: il messaggio è "nelle mani" di TCP;

- HTTP può dunque assumere che non ci siano errori di trasmissione tra client e server (vantaggio dell'architettura a strati);

- Congestion controll → obbliga la connessione a partire con un transmission rate basso (SLOW START); il rate viene poi incrementato compatibilmente con il traffico in rete;

- STATELESS PROTOCOL → il server non mantiene informazioni di stato sul cliente.

Il protocollo HTTP 3: Web page



- Web page di 3 oggetti (1 HTML + 1 GIF + 1 JPG);

- Ogni oggetto è identificato attraverso un URL;

URL: www.dsi.unimi.it/lezioni/architettura/oggi/lezione2.ppt

HOST NAME PATH

Nome del server che ospita l'oggetto

Indirizzo dell'oggetto nel server

Il protocollo HTTP 4: persistent e non-persistent connection

HTTP 1.0 standard

HTTP 1.1 standard

Esempio:
- Pagina Web di 11 oggetti, 1 base HTML + 10 JPEG;
- Tutti gli oggetti si trovano nello stesso server;
- URL www.dsi.unimi.it/lezioni/architettura/base.HTML

Il protocollo HTTP 5: non-persistent connections

1) HTTP client inizializza la connessione con www.dsi.unimi.it attraverso la porta 80;
2) HTTP client invia HTTP request al server; la richiesta include il path /lezioni/architettura/base.HTML per il client socket creato in 1);
3) Il server riceve la HTTP request, recupera /lezioni/architettura/base.HTML dalla RAM o dall'HD, la manda al client attraverso il server socket creato in 1);
4) Il server HTTP dice a TCP di chiudere la connessione; TCP non la chiude fino a quando l'intero messaggio non è stato trasferito;
5) HTTP client riceve il messaggio; la connessione TCP termina;
6) HTTP client apre base.HTML e ripete i punti 1)..5) per ogni oggetto.

Il protocollo HTTP 6: non-persistent connection

- la non-persistent connection esegue client→server→client;

- grado di PARALLELISMO: numero di connessioni aperte contemporaneamente, può servire a ridurre i tempi;

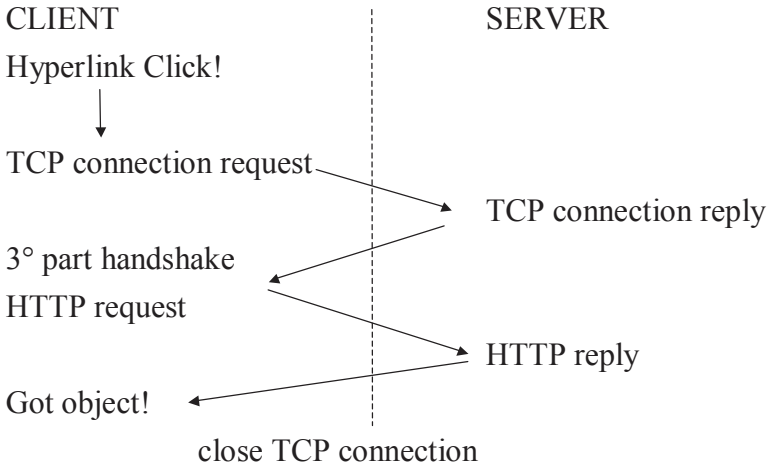
- ROUND TRIP TIME RTT: tempo richiesto da un pacchetto per percorrere client→server→client (propagation delay + queuing delay + packet processing delay);

- click su un hyperlink → 3 way handshake = 1 RTT +

HTTP request e reply = 1 RTT =

TOTAL = 2 RTT

Il protocollo HTTP 7: non-persistent connection



Il protocollo HTTP 8: non-persistent connection

Limiti non-persistent connection:

- nuova connessione per ogni oggetto;
- TCP buffers e variabili nel server per ogni connessione con un grande numero di connessioni;
- ogni oggetto subisce 2 RTT;
- ogni oggetto subisce TCP LOW START.

Il protocollo HTTP 9: persistent connection

- Il server lascia aperta la connessione TCP dopo la HTTP reply;
- HTTP server chiude la connessione TCP quando è inutilizzata per un certo periodo di tempo (timeout interval);
- C'è slow start solo all'inizio di una nuova connessione;
- Persistent connection:
  1) Without pipelining: il client fa HTTP request solo dopo aver ricevuto l'HTTP reply precedente dal server; c'è 1 RTT per ogni oggetto;
  2) With pipelining: default per HTTP 1.1, il client fa richieste al server in ogni momento, per una pagina web intera c'è 1 RTT.

Il protocollo HTTP 10: request message

HTTP REQUEST MESSAGE (ASCII CODE)

GET /lezioni/architettura/base.HTML HTTP/1.1

Host: www.dsi.unimi.it

Connection: close

User-Agent: Mozilla/4.0

Accept-language (it)

Il protocollo HTTP 11: request message

1) Request line: method field + URL field + HTTP version field;
2) Host → server dove risiede l'oggetto
3) Connection: close → indica di usare una non-persistent connection;
4) User-Agent → indica il browser utilizzato sul client; il server può scegliere la versione più adatta al borwser specifico per un certo docuumento;
5) Accept-language → linguaggio preferito (altrimenti: default);
6) Blank line → Entity Body, spazio dedicato ai dati (se presenti);

Il protocollo HTTP 12: request message

Comandi per la request line:

GET → per ottenere una pagina web;

POST → usato quando è necessario riempire un campo prima di inviare l'HTTP request (es. ricerca di una parola con yahoo!) – il contenuto della pagina dipende dalla richiesta dell'utente;

HEAD → simile a GET, il server risponde solo con un messaggio e non con l'intero oggetto; usato nel debugging.

Il protocollo HTTP 13: response message

HTTP RESPONSE MESSAGE (ASCII CODE)

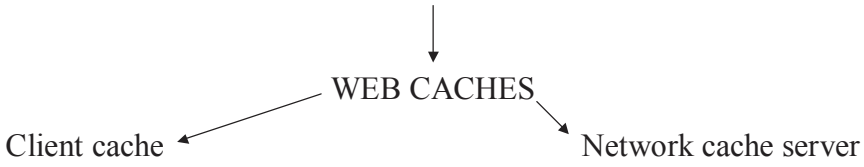
HTTP/1.1 200 OK	initial status line
Connection: close	6 header lines
Date: Thu, 09 Aug 1998 12:00:01 GMT	
Server: Apache/1.3.0 (Unix)	
Last-Modified: Mon, 22 Jun 1998 09:23:24 GMT	
Content-Length: 6821	
Content-Type: text/html	
(data… data …)	entity body

Il protocollo HTTP 14: response message

1) Intital status line: protocol version + status code (=200 OK, =301 Moved permanently, =400 Bad request, =404 Not found, =500 HTTP version not supported, …) + corresponding status;
2) Header lines: non-persistent connection, data di creazione del messaggio da parte del server, tipo di server, data di creazione/ultima modifica dell'oggetto (→ caching), numero di byte dell'oggetto, tipo di oggetto;
3) Entity body: dati dell'oggetto.

Il protocollo HTTP 15: conditional get & caching

Mettendo da parte oggetti già visti, si possono ridurre 1) i tempi di accesso e 2) il traffico in Internet



I "depositi di oggetti già visti" sono detti CACHE

Il protocollo HTTP 16: conditional get & caching

CLIENT CACHE: copia di un oggetto + data di ultima modifica.

```
GET /fruit/kiwi.gif HTTP/1.0
User-agent: Mozilla/4.0
If-Modified-Since: Mon, 22 Jun 1998 09:23:24 GMT
```

Client conditional get request

```
HTTP/1.0 304 Not Modified
Date: Wed, 19 Aug 1998 15:39:29 GMT
Server: Apache/1.3.0 (Unix)
```

Server response message

Il server non spedisce l'oggetto a meno che non sia stato modificato!

Il protocollo HTTP 17: web cache

WEB CACHE (o PROXY SERVER): il server mantiene nella propria cache copia degli oggetti; il client fa HTTP request al proxy server.

1) Il browser (client) crea una connessione TCP con il proxy server e fa a questo la sua HTTP request;
2) Se il proxy server ha una copia dell'oggettoin cache, la manda al client;
3) Altrimenti apre una connessione TCP con il server che contiene l'oggetto, fa l'HTTP request e ottiene l'oggetto;
4) Il proxy server registra allora l'oggetto nella propria cache e lo spedisce al client.

Il protocollo HTTP 18: web cache

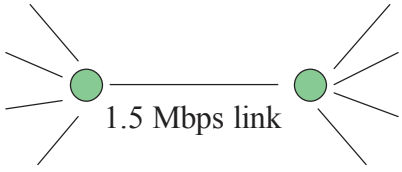
Utilità web cache:

- riduzione tempi di risposta (non occorre arrivare fino all'origin server per accedere all'oggetto);
- riduzione traffico di accesso per un institutional link;
- vantaggi anche per chi ha una connessione lenta.

Osservazione:

Il proxy server si comporta da server e da client allo stesso tempo!

Il protocollo HTTP 19: web cache



High speed LAN (10Mbps)

Public Internet

Dimensione media mex: 100kbits

Nell'istituto: 15 request/s

Tempo di reperimento pagine in Internet: 2s (Internet delay)

Total response time = LAN delay + access (link) delay + Internet delay

LAN traffic intensity = La/R = (15req/s)*(100kbits/req)/(10Mbps) = 0.15

Il protocollo HTTP 20: web cache

Sul link di accesso:

NO CACHE → La/R = (15req/s)*(100kbits/req)/(1.5Mbps) = 1 (!!!)

Soluzioni: access link con transmission rate più alto (costoso!) oppure:

WEB CACHE → La percentuale di richieste soddisfatte dalla cache (hit rate) è tipicamente del 20…70%. Supponiamo hit rate = 0.4.

Il 40% delle richieste viene allora soddisfatto con t~0.

Per il restante 60%, il traffic intensity sul link si riduce a 0.6 (ritardi di poche decine di ms). La soluzionè è meno costosa rispetto all'installazione di un access link con R grande!

Spiegazione esercizio

Il 40% di richieste provenienti dalla LAN è soddisfatto dalla cache. Siccome la LAN è molto veloce, riteniamo trascurabile il tempo in cui un host all'interno della LAN riceve la pagina richiesta dal server che implementa la cache all'interno della LAN.
Il 60% delle richieste non viene soddisfatto dalla cache. Ciò significa che le richieste di pagina devono uscire all'esterno della LAN per essere soddisfatte, cioè devono transitare sul link. Se calcoliamo l'intensità di traffico (La/R) sul link di accesso, questa risulta essere di 0,6 (dunque 0,6 è l'intensità di traffico, non il ritardo!). Il calcolo da fare è:

traffic intensity = La/R = (0.6*15req/s)*(100kbits/req)/(1.5Mbps) = 0.6

Per intensità di traffico pari a 0,6, i ritardi tipici sono nell'ordine delle decine di ms.

Electronic Mail 1: SMTP

Componenti:

- User agent: programma di posta elettronica, es. Eudora;
- mail servers;
- SMTP (Simple Mail Transfer Protocol)

Ogni utente ha una mailbox, per accedere alla quale è necessario effettuare la procedura di autenticazione (username, password);

Il protocollo SMTP usa TCP per il trasferimento dei dati;

SMTP è "vecchio": i messaggi hanno elementi sempre a 7 bit → è necessario convertire i dati non testuali prima di spedirli!

Electronic Mail 2: SMTP

1) A compone il messaggio e lo indirizza a B@serverB.it;
2) Lo user agent A manda il messaggio al mail server A, che lo mette in una message queue;
3) Il mail server A apre una connessione TCP con il mail server B;
4) Handshaking, invio del messaggio;
5) Il mail server B mette il messaggio nella mailbox di B;
6) B usa lo user agent B per leggere la propria posta.

Mail Server A → SMTP → TCP → SMTP → Mail Server B

Electronic Mail 3: Telnet

Un esempio di dialogo tra due mail server (TelNet):

… Handshking…
S: 220 Hamburger.edu
C: HELO crepes.fr
S: 250 Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr… Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu… Recipient OK
C: DATA
S: Enter mail, end with "." on a line by itself
C: Do yuo like ketchup?
C: How about potatoes?
C: .
S: 250 Message accepted delivery
C: QUIT
S: 222 hamburger.edu closing connection

- SMTP usa persistent connection; è necessario un QUIT per terminare la connessione;
- Il server (S) dà l'OK comunicando il suo nome;
- Il client (C) risponde con HELO;
- HELO accepted dal client;
- Il client comunica di avere posta da consegnare;
- Il server accetta la richiesta;
- Il client comunica il destinatario (RCPT TO);

Electronic Mail 4: Telnet

Un esempio di dialogo tra due mail server (TelNet):

… Handshking…
S: 220 Hamburger.edu
C: HELO crepes.fr
S: 250 Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr… Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu… Recipient OK
C: DATA
S: Enter mail, end with "." on a line by itself
C: Do yuo like ketchup?
C: How about potatoes?
C: .
S: 250 Message accepted delivery
C: QUIT
S: 222 hamburger.edu closing connection

- Il server accetta il destinatario;
- Il client avverte che sta per inviare i dati;
- Il server si dichiara ready;
- Dati della mail;
- La mail termina con una riga contenente un solo punto;
- Il server comunica di avere ricevuto i dati;
- Il client non deve spedire altra posta: decide di chiudere la connessione (QUIT);
- il server conferma la chiusura della connessione.

Electronic Mail 5: protocolli di accesso alle mailboxes

HTTP è un PULL PROTOCOL (la connessione è iniziata dal client – il client sceglie i file che vuole ricevere!);

SMTP è un PUSH PROTOCOL (la connessione è iniziata da chi deve spedire → chi riceve posta non sceglie di farlo!).

Dal momento che l'host destinatario non può restare in perenne attesa di posta, è necessario un protocollo per accedere alla propria mailbox !

Electronic Mail 6: protocolli di accesso alle mailboxes

User agent A → Mail Server A → Mail Server B → User Agent B

SMTP SMTP POP3 o IMAP o HTTP

Per "scaricare" la posta si possono usare i protocolli POP3 (porta 110) o IMAP; si può inoltre usare il protocollo HTTP (utile per utenti "viaggiatori").

Electronic Mail 7: protocolli di accesso alle mailboxes (POP3)

```
…TCP handshaking…
C: telnet MailServer 110
S: +OK POP3 server ready
C: user alice
S: +OK
C: pass hungry
S: user succesfully logged on
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: This is the message 1.
S: It is finished!
S: .
C: dele 1
C: quit
S: +OK POP3 server signing off
```

autenticazione

lista di messaggi nella mailbox e loro dimensione (“.” = fine)

primo messaggio: accesso e cancellazione (“.” = fine)

quit

The Internet Directory Service (DNS) 1

Identificazione host → HOSTNAME, ad esempio www.dsi.unimi.it;

Problemi hostname:

- non danno info sulla localizzazione geografica dell’host;
- lunghezza variabile.

Accanto all’hostname → IP ADDRESS, es 149.159.146.194;

- 4 campi da 8 bit;
- organizzazione gerarchica.

DNS (Domain name system) associa gli indirizzi IP ai nomi!

The Internet Directory Service (DNS) 2

DNS = Database distribuito implementato in una gerarchia di name servers + protocollo appropriato che permette di comunicare tra host e name servers per accedere al translation service.

Name servers → di solito macchine UNIX + sofware BIND;

Protocollo DPS → utilizza UDP, porta 53;

DNS → comunemente implementato in altri application layer protocols (HTTP, SMTP, FTP, …);

Translation service → associazione dell'indirizzo IP all'hostname.

The Internet Directory Service (DNS) 3: translation service

1) Scrivo www.dsi.unimi.it/prova.html; in explorer è implementato il client side di DSN;
2) Il client side di DNS manda un messaggio con l'hostname (www.dsi.unimi.it) al DNS server;
3) Il DNS client riceve la risposta in cui è contenuto l'indirizzo IP di www.dsi.unimi.it;
4) Il browser si connette all'indirizzo IP.

DNS introduce un ulteriore ritardo, che può essere limitato tramite l'utilizzo di cache di indirizzi IP.

The Internet Directory Service (DNS) 4: other services

HOST ALIASING → alcuni hosts hanno ALIAS NAMES, uno solo è quello CANONICO; DNS fornisce il nome canonico a partire da un nome alias;

MAIL SERVER ALIASING → anche i mail server possono avere alias names; inoltre web e mail server possono avere lo stesso nome; DNS fornisce il nome canonico del mail server;

LOAD DISTRIBUTION → vi sono REPLICATE WEB SERVER, cioè siti replicati su host diversi (ad esempio CNN → sito affollato!); DNS fornisce allora una lista di IP address ad ogni richiesta, ogni volta in ordine diverso.

The Internet Directory Service (DNS) 5: come lavora DNS

Modello DNS più semplice:

- 1 name server cui accedono tutti i clients.

Problemi:

- Se si rompe il name server…
- Traffic Volume: intensissimo sul name server;
- Distant centralized database: ritardi significativi per clients lontani dal name server;
- Maintenance: frequenti udpate all'inserzione di nuovi hosts in rete.

The Internet Directory Service (DNS) 6: come lavora DNS

I name servers sono GERARCHICAMENTE organizzati in:

1) LOCAL NAME SERVER: negli istituti, università, … Il server si trova all'interno della LAN; l'host manda la sua richiesta al local name server, che contiene inoltre tutti gli indirizzi degli host locali;
2) ROOTNAME SERVER: ce ne sono una dozzina in Internet, perlopiù in USA; il local name server chiede al rootname server quando non può soddisfare una richiesta;
3) AUTHORITATIVE NAME SERVER: ogni host è registrato in almeno 2 authoritative server che si trovano nell'ISP locale; un server si dice authoritative per un host se contiene SEMPRE il suo indirizzo IP; molti local name server sono anche authoritative!

The Internet Directory Service (DNS) 7: come lavora DNS

1) Client: DNS query al local name server;
2) Local Name Server: se può risponde al client (→ fine), altrimenti DNS query al rootname server;
3) Rootname Server: se può risponde al Local Name Server (→ fine), altrimenti cerca tra i suoi indirizzi quello di un authoritative name server che conosce la risposta e fa DNS query;
4) L'authoritative name server risponde al Rootname Server, che risponde al Local Name Server, che risponde al Client.

Di solito vengono utilizzati anche name server intermedi!

The Internet Directory Service (DNS) 8: come lavora DNS

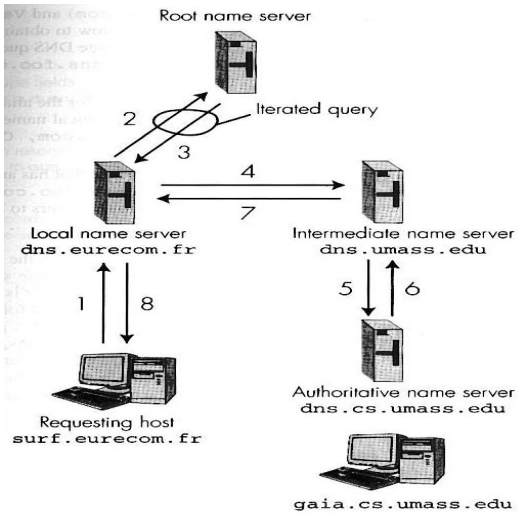
DNS RECURSIVE QUERIES: l'host A fa una richiesta all'host B; l'host B si comporta in modo tale da fornire comunque la risposta ad A.

DNS ITERATIVE QUERIES: se il server B non conosce l'indirizzo richiesto da A, restituisce ad A l'indirizzo IP del name server successivo nella catena.

Di solito: recursive queries, iterative queries solo tra local name server e root name server (per diminuire il lavoro del root name server).

DNS CACHING → i name servers registrano nella RAM o nell'HD gli ultimi indirizzi IP (caching) → diminuzione traffico e tempi di risposta!

The Internet Directory Service (DNS) 9: come lavora DNS



The Internet Directory Service (DNS) 10: DNS record

Resource record: (Name, Value, Type, TTL)

se Type = A	⇒	Name = hostname, Value = IP address;
se Type = NS	⇒	Name = domain, Value = hostname di un authoritative server che conosce il modo di ottenere l'IP dell'host nel domain;
se Type = CNAME	⇒	Name = alias hostname, Value = canonical hostname (web server).
Se Type = MX	⇒	Name = alias hostname, Value = canonical hostname (mail server).

TTL = time to live, indicazione sul tempo di permanenza in cahce.

The Internet Directory Service (DNS) 11: DNS message