Riassumendo - name space vs name servers

Questo albero riporta, oltre ai TLD, alcune etichette di livello più basso, che fanno riferimento ad applicazioni che fanno riferimento a TLD.

E' riportato anche il nodo foglia **www** che rappresenta un servizio all'interno dell'organizzazione DING; leggeremo quindi `www.ding.unisannio.it.` .

E' possibile creare un albero di name server; i name server vengono associati ai nodi dell'albero, vi è quindi un mapping tra i name server ed i nodi dell'albero precedente.

Un possibile mapping potrebbe essere:

Abbiamo un server che gestisce il nodo root e che è a conoscenza dei server TLD; ad esempio abbiamo anche un server che gestisce un server TLD: **it**. In questo troveremo le informazioni relative alle macchine che fanno riferimento a questo dominio (it).
Successivamente abbiamo un server che gestisce il dominio unisannio.

Spazio dei server

Se ignoriamo completamente le etichette, otteniamo la rappresentazione grafica dello **spazio dei server**, dove abbiamo i vari server DNS che sono collegati tra loro tramite un canale trasmissivo, dove ogni server può contattare un altro server di cui conosce l'indirizzo di trasporto.

DNS Query - Esempio

Per questo esempio facciamo finta che `virgilio.unisannio.it` voglia comunicare con `gaia.cs.umass.edu`, per farlo deve **chiedere** l'indirizzo IP ad un server DNS. Quindi effettua una richiesta al server locale `dns.unisannio.it` e questo server, non avendo la risposta, interroga un **name server di livello superiore**, quindi i name server locali si rivolgono ai name server root per poi attraversare l'albero dalla radice verso la foglia.

Una volta che la richiesta ha seguito la catena fino ad arrivare al name server autorevole (quello che cerchiamo), torna sui suoi passi fino al client che ha effettuato la richiesta.

Tipi di query DNS

Query Ricorsive

Questo tipo di query è il tipo che abbiamo visto nell'esempio precedente; ricorsivo vuol dire che una volta che il client effettua una richiesta per un name server, questo deve rispondere o con la risposta effettiva o con una risposta che ci dice che la risorsa non è stata trovata.

Sarà quindi responsabilità del name server contatto, contattare il name server suo superiore, attraverso chiamate ricorsive.

Query Iterative

Sono iterative quelle query nelle quali il name server contattato non si assume la responsabilità di risolvere direttamente la query, ma restituisce la risposta se ne è in possesso, altrimenti restituisce il successivo indirizzo IP del name server da contattare per poter ricevere la risposta.

Essenzialmente il name server contattato se non ha l'indirizzo IP dell'indirizzo richiesto, se ne lava le mani e dice al client di contattare il successivo name server.

Quale usare?

E' meglio usare una query iterativa piuttosto che delegare il name server, per motivi di **scalabilità**; questo perchè in questo modo il root server non deve tenere traccia dei messaggi di richiesta che ha ricevuto (perché lavorando ricorsivamente deve tenere traccia della query in corso).

Caching

E' opportuno che una volta che è stata effettuata una risoluzione ed una volta che un server lungo la catena ha ottenuto la risposta, mantenga questa risposta, in modo tale da rendere più veloce la risposta qualora la richiesta venga rifatta.

Alcuni name server salvano altri indirizzi IP di name server di "grado superiore", in modo tale da contattare prima quelli, così da poter caricare il meno possibile il server radice.

Come tutti i meccanismi di caching, la risorsa che viene propagata viene mantenuta in cache per un tempo fissato, stabilito da un parametro chiamato **time-to-live** che è associato ad ogni risorsa presente in ogni server.

Protocollo DNS

Dal punto di vista della rete le interrogazioni sono effettuate attraverso uno **scambio di messaggi.**

I messaggi usati sono di due tipi:

- **Messaggi di richiesta**
- **Messaggi di risposta**

Sia i messaggi di richiesta presentano un'intestazione (header) che è strutturata nel seguente modo:

Abbiamo 6 campi da 16 bit ognuno, posti prima della parte del messaggio vero e proprio.

- Identificatore: contiene l'identificatore della richiesta, questo identificatore consente di associare un messaggio di richiesta con uno di risposta; di conseguenza la coppia di messaggi richiesta-risposta dovranno avere lo stesso ID.
- Flag: `QR, OpCode, AA, TC, RD, RA, 000, rCode`
  - QR serve a specificare se il messaggio è di richiesta o di risposta, avremo un valore 0/1 a seconda se il messaggio sia di richiesta o di risposta
  - OpCode: solitamente è 0, e viene usato per caratterizzare il tipo di operazione, che possono essere **normali o inverse** ; le interrogazioni inverse prevedono di arrivare ad un FQDN partendo da un indirizzo IP.
  - AA: ci dice se la risposta che viene fornita è autorevole; è significativo nei messaggi di risopsta: se troviamo un bit == 1 vuol dire che la riposta è stata fornita da un server autorevole.
    La riposta potrebbe essere restituita dalla cache di uno dei server lungo la catena.
  - TC: viene usato per indicare se il messaggio è stato troncato oppure no; siccome il DNS usa un protocollo UDP, che come sappiamo confina i messaggi

all'interno di datagram, potrebbe "spezzettare" (**troncato**) il messaggio. **Quando questo avviene, viene usato il protocollo TCP** insieme all'UDP.
    - RD: questo flag indica che vogliamo usare la ricorsione con la nostra richiesta.
    - RA: questo flag viene usato nel messaggio di risposta per indicare che la ricorsione è stata accettata.
    - 000: bit a zero
    - rCode: codice che ci dice se ci sono stati eventuali errori
- **Numero di record di richiesta** contengono il numero di record di richiesta
- **Numero dei record di risposta** numero di record che fanno riferimento al server autorevole - 0 nei messaggi di richiesta.
- **Numero di recors supplementari**: informazioni supplementari fornite dal server contattato per evitare che il client faccia ulteriori richieste. - 0 nei messaggi di richiesta.
- **Numero dei record autorevoli** - 0 nei messaggi di richiesta

Richieste e risposte DNS

*Ricordiamo che tutto ciò che è stato visto nella sezione precedente è nell'**header**, all'interno di una struttura di grandezza prefissata.*

Per quanto riguarda i messaggi di richiesta, abbiamo solo la sezione delle richieste, sezione caratterizzata dal **numero di record di richieste** (visto nella sezione precedente), che ci dice quante query abbiamo nel messaggio.

Nel caso del messaggio di risposta abbiamo diverse sezioni che fanno riferimento ai 4 campi numerici (anche questi visti prima).

Richiesta

Ognuna di queste sezioni ha una struttura: la sezione richieste è strutturata con dei record del tipo:

- Query name: nome di supporto all'interrogazione, parametro dell'interrogazione.
- Query type: specifica che tipo di interrogazione voglio effettuare
- Query class: a quale classe appartiene l'interrogazione.

In **Query name** troveremo **l'FQDN** di cui vogliamo l'indirizzo IP (ad esempio www.unisannio.it). La struttura è tale per cui non è possibile specificare la dimensione del Query Name, quindi possono essere di dimensione variabile. Dobbiamo prevedere una soluzione per indicare al server che legge il messaggio di richiesta, come recuperare da questa sezione il query name; se non sa quanti caratteri deve leggere, leggerà in modo errato.

La soluzione usata è quella di scrivere l'FQDN all'interno della struttura, anticipando ciascuna label con un numero che caratterizza la dimensione della label.

Quindi: `3www9unisannio2it0` lo zero indica la fine della stringa.

Risposta

La risposta ha questa struttura:

Abbiamo diverse informazioni, come ad esempio il tempo di vita dell'informazione qualora essa venisse mantenuta in cache, ed a differenza della richiesta, questa volta viene specificata la lunghezza esplicita della risorsa, successivamente abbiamo i byte della risorsa.

DNS records - query

Il sistema DNS non è solo usato per effettuare risoluzioni semplici da un FQDN al suo indirizzo IP, ma possiamo anche effettuare interrogazioni più complesse:

Interrogazione Type = NS

Possiamo sapere qual è il name server che opera all'interno di un dominio; in questo caso l'interrogazione diventa di tipo **NS**; nella sezione di risposta troveremo l'FQDN del name server:

Quando usiamo **NS** il nome **query name** non è più il nome dell'host ma il nome di dominio.

Interrogazione Type = MX

Questa interrogazione prevede come query name il dominio (come NS), e consente di risalire all'FQDN del mail server che opera in un dominio; si tratta del server che viene usato per trasportare il messaggio di posta elettronica verso il destinatario.

Nel momento in cui inviamo un messaggio di posta elettronica attraverso il client specificando il destinatario, come fa il client a consegnare quel messaggio? O meglio, **quale server contatta?**

Il client ovviamente non conosce il server che gestisce il dominio di posta elettronica dell'email selezionata, ma conosce solo il server locale (come accade con la risoluzione dei domain name); quindi è configurato per contattare un server di ingresso per la gestione della posta elettronica.

Quel server che il client contatta, deve risalire all'indirizzo IP del mail server di destinazione, che è quello che poi gestisce effettivamente la mail box in cui deve essere inserito il messaggio da noi scritto.

Il client usa l'indirizzo di posta elettronica del destinatario per estrarre il dominio del server che hosta la posta elettronica dell'email a cui vogliamo inviare il messaggio, in modo da recuperare l'indirizzo IP del server che opera in quel dominio.

Test pratico

🏁 00:51

dig

Usiamo il comando `dig` per effettuare delle interrogazioni:

Troviamo l'indirizzo IP di www.unisannio.it

Se digitiamo `dig A www.unisannio.it` otteniamo:

Possiamo notare il nome, tipo e classe. Il tempo di vita dell'informazione è 137934, in secondi.

La sezione di richiesta è labeled come **QUESTION SECTION:**.

Possiamo inoltre notare i flag di cui abbiamo parlato prima: **qr, rd, ra**;

Troviamo il name server del dominio unsannio.it

Digitando `dig NS unisannio.it` otteniamo:

La sezione supplementare ci permette non solo di scoprire il name server del dominio unisannio.it, ma nella sezione addizionale ci mostra anche l'indirizzo IP (non so perchè con questa query non è stata mostrata la sezione addizionale).

Scopriamo il mail server del servizio unisannio.it

Digitando `dig MX unisannio.it` otteniamo:

il Mail server è **mailgw.unisannio.it**; se volessimo sapere l'indirizzo IP di questo server ci basterebbe digitare, come prima, `dig A mailgw.unisannio.it`, ottenendo `;; ANSWER SECTION: mailgw.unisannio.it.  172674  IN  A 193.206.108.11`

E su windows?

Su windows ci si attacca.

A parte gli scherzi, su windows è possibile usare **nslookup**, anche se windows andrebbe, a parere di molti, lasciato alle segretarie/segretari.

Ad ogni modo, è possibile usare **nslookup** anche su UNIX:

WWW ed HTTP

🏁 1:03

WWW

Cosa è il WWW dal punto di vista strutturale?

Il servizio www è implementato da due programmi, un **client ed un server**, che vengono eseguiti su alcune macchine sull'internet, e da un protocollo applicativo: **l'HTTP**. L'HTTP definisce la struttura e la semantica dei messaggi scambiati tra client e server.

I client sono rappresentati dai **browser**, e contattano i server per ottenere delle risorse. Queste richieste possono essere richieste di file HTML, immagini, audio, video, ecc. Possiamo avere quindi sul server dei files che vengono poi inviati come risposta ai client.

Queste risorse vengono trasferite dal server al client, ed una volta che il client le ha ricevute le **decodifica**, con l'obbiettivo di **interpretare** le risorse ricevute. Il termine usato in questi casi è **"rendering"**, ovvero il browser renderizza le risorse ricevute dal server.

Il programma server, spesso chiamato **server web** ascolta sulla **porta 80** e gestisce le risorse web remote accessibili usando una **URI**; una URI (Uniform Resource Identifier), ed è un'estensione del concetto **FQDN** visto prima. Ancora una volta è un "nome", non usato per identificare una macchina ma per identificare una **risorsa specifica (file)**.

Identificatore delle risorse : URI

Abbiamo quindi capito che le URI vengono usate per identificare delle **risorse**; abbiamo due tipi di URI:

- **URL** : è la variante nota come **uniform resource locator**

- **URN** : nota come **uniform resource name**

La differenza tra URL ed URN, sta nel fatto che nell'URL è presente anche la posizione all'interno della rete occupata dalla risorsa, oltre che al suo nome.

La struttura generale di un URL è la seguente:

`<schema> :// <nome server> <:porta> / <path> / <nome file> #sezione`

Come possiamo notare, compare la voce <nome server>, di conseguenza dobbiamo specificare l'FQDN, o indirizzo di trasporto del server che hosta la risorsa; il browser utilizza quindi un **resolver** prima di poter contattare il server. Questo perchè nella maggior parte delle volte specifichiamo un FQDN, e non un indirizzo di trasporto. Quindi, come abbiamo visto nella prima parte della lezione, il client deve **risolvere** (tramite un resolver) l'indirizzo di trasporto associato all'FQDN.

URN

A differenza dell'URL l'URN non presenta al suo interno la posizione (path) della risorsa, ma è semplicemente un nome. Ad esempio un URN potrebbe essere l'ISBN di un libro: `urn:isbn:0012313892`.

Questa peculiarità non è per forza una cosa negativa, infatti potremmo recuperare la risorsa da fonti diverse, ad esempio **dal server più vicino**, e non da uno specifico server come accate con l'URL.

A livello di trasporto?

A livello di trasporto il protocollo utilizzato è il TCP; viene usato il TCP perchè la maggior parte delle volte i contenuti trasmessi sono di grandi dimensioni, che di conseguenza superano i limiti consentiti dai datagram.

HTTP

Tutti i protocolli previsti per il funzionamento delle applicazioni su internet, sono documentati dall RFC, quindi possiamo consultare il documento in rete.

L'HTTP sta per **Hyper Text Transfer Protocol**, proprio perchè questo protocollo è stato pensato per il trasferimento di testi ipertestuali, struttura tipica dei file HTML. L'HTTP inoltre non salva **informazioni di stato** associate ad uno specifico client, e viene quindi detto **stateless**. Ogni interazione HTTP per il server è come se fosse nuova, proprio perchè il server non memorizza alcuna informazione. I server dovrebbero essere privi di stato per motivi di affidabilità e di scalabilità.

Formato di messaggio HTTP

I messaggi di richiesta e risposta HTTP hanno un formato base:

Prevedono un'intestazione basata su testo; in particolare abbiamo diverse linee: Una prima linea viene chiamata **start-line** seguita da altre linee che contengono dei formati specifici.

Dopo il CRLF, ovvero una linea vuota, abbiamo il messaggio vero e proprio, ovvero la parte più significativa del messaggio, che contiene le parte di informazioni che vogliamo trasferire.

Per quanto riguarda gli header invece, sono composti da una parte binaria.

Quindi?

Ogni linea che troviamo all'interno del messaggio è strutturata nel seguente modo; per ogni linea abbiamo la struttura `<nome del campo>:<valore>`; quindi sostanzialmente l'intestazione è composta da tante coppie nomecampo:valore.

Abbiamo diversi tipi di linee di intestazione

Le linee di intestazione si differenziano a seconda dell'aspetto dell'interazione che vogliono caratterizzare; se la linea di intestazione è generale, parliamo di **general header**, un esempio è la **data in cui avviene l'interazione**.

Header HTTP

Gli header sono linee di testo **free-format** che possono specificare:

- **Informazioni generali - General Header**
  - Esempio: Date: < transmission date>
- **Informazioni associate alla relazione - Request Header**
  - Esempio: Referer: < URL della risorsa che origina la richiesta>
- **Informazioni associate alla risposta - Response Header**
  - Esempio: Server < una stringa che descrive il server>
- **Informazioni associate alle entità - Entity Header** : content type
  - Esempio: Content-Type: < MIME type of the data in the message body>
  - Esempio di formato mime: text/html

Content type è seguito da un formato (stringa) **MIME - Multipurpose Internet Mail Extensions**; attraverso la lettura di questa stringa il browser è in grado di interpretare il tipo di contenuto che ha ricevuto come risposta. Se ad esempio la risposta usa un MIME del tipo **text/html**, il browser saprà che dovrà interpretare i dati ricevuti come testo formattato in html (che prevede il rendering del contenuto).

Come è composto un request message?

Request Line

La differenza sta nella **start-line**, a seconda che il messaggio sia di richiesta o di risposta. In particolare, nel messaggio di richeista abbiamo una **request-line** al posto della start-line; in questa linea abbiamo 3 token:

`<metodo> <nome della risorsa> <versione del protocollo>`

Il primo è il **metodo** attraverso il quale avviene la richiesta (POST, GET, ecc), poi abbiamo il **nome della risorsa** sulla quale vogliamo operare. L'ultimo token presente è la versione del protocollo: ci consente di usare i campi previsti dall'intestazione in modo corretto.

Esempio di request line: `GET /index.html HTTP/1.1` nulla ci vieta di avere un intero path al posto di `/index.html`.

Request header lines

- Coppia **chiave-valore** per ogni linea
- Queste coppie sono usate per caratterizzare le richieste
- Esempio: User-Agent: < vendor and version of the client>

- Esempio: Accept: < types of content recognized by the client>

Linea vuota

Data che completa la richiesta

Il body non è sempre presente, infatti non tutti i messaggi di richiesta HTTP1.1 prevedono un body.

Esempio di richiesta HTTP

possiamo usare wireshark per osservare un'interazione web

Possiamo osservare un messaggio di richiesta dove osserviamo la request line: `GET /path/index.html/ HTTP1.1` dove il client ha richiesto `index.html` al server host: `www.unisannio.it`. Successivamente abbiamo una linea vuota ed il body, che in questo caso non è presente.

Perchè specifichiamo l'host?

L'indirizzo di trasporto dell'host è già presente nell'intestazione del segmento, quindi perchè devo specificarlo anche nella richiesta? Host è stato introdotto con l'obbiettivo di supportare **server web virtuali**; quindi è possibile contattare un server web (con l'indirizzo di trasporto), ma il server web a sua volta potrebbe gestire **più siti logici**, ognuno dei quali potrebbe avere un indirizzo diverso.

Questa informazione, quindi, non è usata **per arrivare al server web**, ma sarà **usata dal server web**.

Metodi di richiesta HTTP

- **GET** : richiede una risorsa identificata da un URI
- **HEAD** : richiede informazioni su una risorsa; il messaggio di richiesta è simile al metodo GET.
- **POST** : chiede alla risorsa identificata dalla URI di processare i dati passati dal body del messaggio di richiesta.
- **PUT** : chiede al server di salvare la risorsa identificata dall'URI come una nuova risorsa.
- **DELETE** : chiede al server di eliminare una risorsa identificata dallURI.
- **OPTIONS** : chiede al server di specificare la lista di metodi che possono essere applicati alla risorsa identificata dall'URI.
- **TRACE** : chiede al server di effettuare un **echo** della richiesta per motivi di debugging.
- **CONNECT** : chiede il tunneling di una connessione TCP su HTTP.

Questi metodi sono particolarmente importanti per la programmazione in web; oggi caratterizzano le azioni che possono essere eseguite usando un particolare tipo di servizio

Proprietà dei metodi

- **Safety** : l'esecuzione non cambia la risorsa;
  Esempio : GET è un metodo che produce un'azione safe.
  Esempio : POST modifica lo stato, quindi **non** produce un'azione safe.
- **Idempotenza - Idempotency** : ha a che fare con un'invocazione, e dobbiamo tenere presente che l'interazione fallisca (per via di un malfunzionamento) e che quindi ci possa essere un'interazione non corretta, o senza successo. Per **idempotenza** prevediamo la possibilità di **rieseguire l'interazione** n volte, senza che la riesecuzione produca effetti diversi da quelli prodotti da una singola interazione.

Quindi vogliamo dire che più operazioni eseguite devono produrre lo stesso risultato ottenuto con una singola interazione.

Esempio : GET è un metodo che produce interazione idempotenti, questo perchè la singola invocazione di GET, o la multipla invocazione di GET, producono **sempre la stessa risorsa** .

Esempio : POST è un esempio di metodo non idempotente, perchè la riesecuzione di POST potrebbe dare luogo alla scrittura di diverse risorse sul server.

---

fine lezione 17