Práctica 2: Cliente DNS

Introducción

El protocolo DNS (*Domain Name System protocol*, RFC 1035) es el principal protocolo de resolución de nombres usado en redes TCP/IP. Es decir, su misión principal será convertir nombres de máquinas (*hosts*) de Internet a sus respectivas direcciones IP.

Tabla de Contenidos de la práctica

1) Fundamentos del protocolo DNS

Espacio de 'Nombres de Dominio' (Domain Name Space):

Todas las máquinas (*hosts*) en Internet que quieran usar el protocolo DNS, deberán pertenecer a un 'dominio', normalmente llamado 'Nombre de Dominio' (*Domain Name*). A su vez, los dominios tienen una dependencia jerárquica entre sí, conformando un gran árbol que es el 'Espacio de Nombres de Dominio'. La 'dependencia jerárquica' quiere decir que los dominios son como conjuntos, de modo que un dominio puede englobar a otros dominios, llamados comúnmente 'subdominios'.

Los dominios se identifican con nombres de longitud variable, de modo que para identificar un dominio desde la raíz del árbol del 'Espacio de Nombres de Dominio', se separa cada subdomino del dominio al que pertenece por un punto '.'

Ejemplos de nombres de dominio: 'uam.es' ó 'eps.uam.es'.

Recursos (Resources)

Son los elementos, servicios o grupos que puede haber dentro un dominio. Se diferencian varias 'clases' de recursos que se identifican por unas pocas letras:

- 'Direcciones de Máquinas' (conocidos como recursos de clase 'A')
- 'Servidores de Nombres' (conocidos como recursos de clase 'NS')
- 'Servidores de Correo' (conocidos como recursos de clase 'MX')
- · ... y muchos otros especificados más abajo y en la RFC

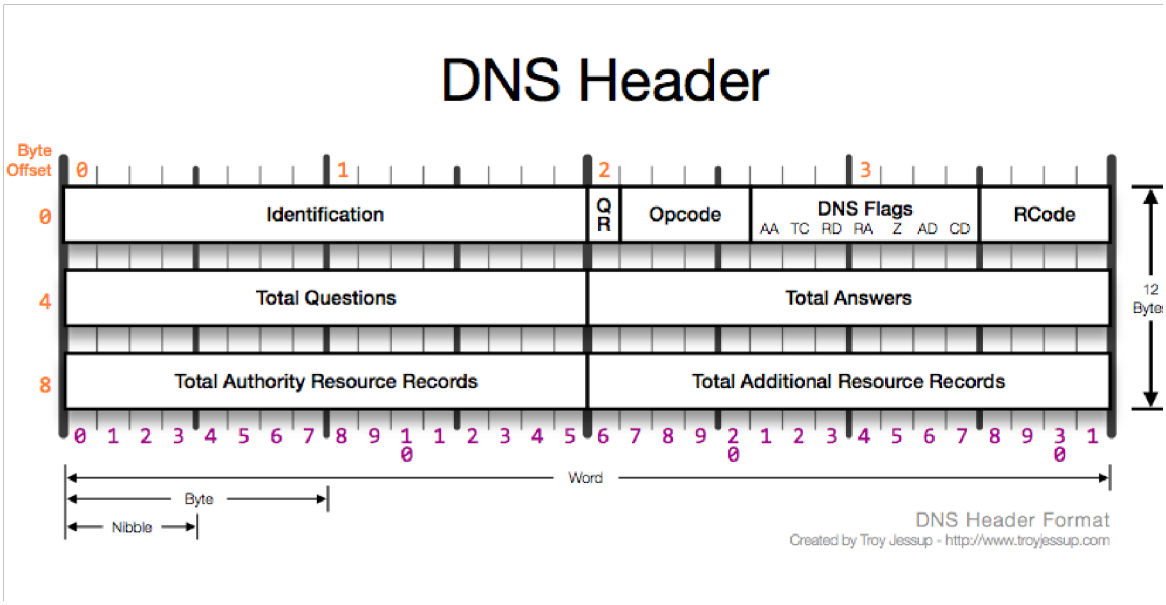
Cada recurso, al igual que los dominios, tiene un nombre de longitud variable. Un *host* común que posea un nombre y una dirección IP será considerado como un recurso de clase 'A'. Un ejemplo muy común de recurso de clase 'A' son los servidores web, a los que se les suele asignar el nombre 'www'.

Ejemplo: La cadena 'www.eps.uam.es' hace referencia a un recurso clase 'A' llamado 'www' que se encuentra en el dominio 'eps.uam.es' (en realidad existen 3 dominios: 'eps' que es un subdominio de 'uam.es', el dominio 'uam' que es un subdominio de 'es' y el propio dominio 'es').

Especificación del protocolo

A continuación, se dan las especificaciones de las tramas usadas en el protocolo DNS:

Cabecera DNS:



Identification. 16 bits.

Used to match request/reply packets.

QR, Query/Response. 1 bit.

QR	Description
0	Query.
1	Response.

Opcode. 4 bits.

Opcode	Description	References
0	QUERY, Standard query.	RFC 1035
1	IQUERY, Inverse query.	RFC 1035
2	STATUS, Server status request.	RFC 1035
3	Reserved.	
4	Notify.	RFC 1996
5	Update.	RFC 2136
6 - 15	Reserved.	

AA, Authoritative Answer. 1 bit.

Specifies that the responding name server is an authority for the domain name in question section. Note that the contents of the answer section may have multiple owner names because of aliases. This bit corresponds to the name which matches the query name, or the first owner name in the answer section.

AA	Description
0	Not authoritative.
1	Is authoritative.

TC, Truncated. 1 bit.

Indicates that only the first 512 bytes of the reply was returned.

TC	Description
0	Not truncated.
1	Message truncated.

RD, Recursion Desired. 1 bit.

May be set in a query and is copied into the response. If set, the name server is directed to pursue the query recursively. Recursive query support is optional.

RD	Description

0	Recursion not desired.
1	Recursion desired.

RA, Recursion Available. 1 bit.

Indicates if recursive query support is available in the name server.

RA	Description
0	Recursive query support not available.
1	Recursive query support available.

Z. 1 bit.

It must be set to zero (0).

AD. 1 bit.

It must be set to zero (0).

CD. 1 bit.

It must be set to zero (0).

Rcode, Return code. 4 bits.

Rcode	Description	References
0	**No error.** The request completed successfully.	RFC 1035
1	**Format error.** The name server was unable to interpret the query.	RFC 1035
2	**Server failure.** The name server was unable to process this query due to a problem with the name server.	RFC 1035
3	**Name Error.** Meaningful only for responses from an authoritative name server, this code signifies that the domain name referenced in the query does not exist.	RFC 1035
4	**Not Implemented.** The name server does not support the requested kind of query.	RFC 1035
5	**Refused.** The name server refuses to perform the specified operation for policy reasons. For example, a name server may not wish to provide the information to the particular requester, or a name server may not wish to perform a particular operation (e.g., zone transfer) for particular data.	RFC 1035

6	**YXDomain.** Name Exists when it should not.	RFC 2136
7	**YXRRSet.** RR Set Exists when it should not.	RFC 2136
8	**NXRRSet.** RR Set that should exist does not.	RFC 2136
9	**NotAuth.** Server Not Authoritative for zone.	RFC 2136
10	**NotZone.** Name not contained in zone.	RFC 2136
11 - 15	Reserved.	

Total Questions. 16 bits, unsigned.

Number of entries in the question list that were returned.

Total Answer RRs. 16 bits, unsigned.

Number of entries in the answer resource record list that were returned.

Total Authority RRs. 16 bits, unsigned.

Number of entries in the authority resource record list that were returned.

Total Additional RRs. 16 bits, unsigned.

Number of entries in the additional resource record list that were returned.

Questions[]. Variable length.

A list of zero or more *Query* structures.

Answer RRs[]. Variable length.

A list of zero or more Answer *Resource Record* structures.

Authority RRs[]. Variable length.

A list of zero or more Authority *Resource Record* structures.

Additional RRs[]. Variable length.

A list of zero or more Additional *Resource Record* structures.

Query. Variable length.

00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31	
Query Name :::	
Type	Class

Query Name. Variable length.

The query name corresponds to an Internet name where dots ('.') are suppressed and each word is preceded by one byte containing its length:

Example: The name 'www.ii.uam.es' must be written in the field as following bytes:

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14
3	w	w	w	2	i	i	3	u	a	m	2	e	s	0

Please note that a zero (0) is added at the end, symbolizing the root at the Domain Name Space. The maximum size for each domain name (word) is 63 bytes.

Type. 16 bits, unsigned.

Type	Description	References
1	**A.** IPv4 address.	RFC 1035
5	**CNAME.** Canonical Name	RFC 1035

Class. 16 bits, unsigned.

Class	Description	References
1	IN, Internet.	RFC 1035.

Resource Record. Variable length.

00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31	
Name :::	
Type	Class
TTL	

Rdata Length	Rdata :::

Name. Variable Length.

May be a name with same format as 'Query Name', or a 'Pointer' if a compression scheme is being used. A compression scheme can be used if a same name is repeated several times inside a DNS response. In such case (compression scheme begins to work), a 'pointer' will be used instead of the name.
Important Note: If two highest bits for the first byte are set to 1 (values between 0xC0 ~ 0xFF), it is a 'Pointer', otherwise it is a name.
A pointer has a fixed size of 16 bits, but 4 highest bits must be set to 0 to work with it.
A pointer is the offset inside the DNS response, where the name can be found.
Names have same format as specified in 'Query Name' and have a variable length.

TTL. 32 bits.

When a "foreign" DNS record is kept in a DNS server's cache, the record's TTL is continuously reduced as time go by, and when the TTL finally reaches zero the record is removed from the cache.
The returned TTL is current TTL of the DNS record on its cache.

Rdata Length. 16 bits.

Size for next field ('Rdata') in bytes.

Rdata. Size, in bytes, defined on 'Rdata Length'.

Response data depending on the 'query'.
This field, when queries are referred to internet address, will contain the IP address as 4 bytes.

Notas Importantes:

- Para un único nombre, el servidor de nombres puede devolver varias respuestas válidas. Esto es debido a los 'alias' que puedan existir para un mismo nombre.

  En el caso del host 'www.eps.uam.es', el host 'www' es un alias del host 'afrodita' (ya que los dos son el mismo host), por lo que probablemente tengamos dos respuestas, siendo el campo 'Name' de la segunda respuesta, el nombre 'afrodita.ii.uam.es'.

- La primera respuesta contiene directamente un puntero al nombre de la 'query' que va delante de ella, con ello aplica el procedimiento de compresión y evita repetir el nombre. El valor del puntero será 12, que coincide con el tamaño de la cabecera DNS.

- Dependiendo del servidor de nombres al que se acceda, podría ocurrir que en la respuesta no nos incluyera el 'query' que le hicimos, por lo que la primera respuesta tendría en su primer campo el 'nombre' (con el formato adecuado) que solicitamos y no habría un puntero.

2) APIs

SOCKETS DATAGRAM (UDP)

Las primitivas básicas de la API para usarlas en el cliente son:

Crear socket:

int socket(int *socket_family***, int** *socket_type***, int** *protocol***);**
Ej: sock = socket(AF_INET, SOCK_DGRAM, 0);

Enviar:

ssize_t sendto(int *s***, const void ****buf***, size_t** *len***, int** *flags***, const struct sockaddr *dest_addr,socklen_t addrlen);**
Ej: c = sendto(sock,data, strlen(data), 0,to_addr,sockaddrlen);

Esperar datos:

ssize_t select(int *nfds***, fd_set ****readfs***,fd_set ****writefds***,fd_set ****exceptfds***, struct timeval** **timeout***);**

Ej: c = select(FD_SETSIZE, &rfds, (fd_set *)NULL, (fd_set *)NULL, &tv);

Recibir:
ssize_t recv(int *s***, void ****buf***, size_t** *len***, int** *flags***);**

Ej: c = recv(sock, buffer, sizeof(buffer), 0);

Cerrar socket:
int close(int fd);

Ej: close(sock);

Para más información, consultad el manual de las funciones.

Se pueden encontrar multitud de manuales del uso de sockets. Por ejemplo, se puede consultar un ejemplo de uso de sockets UDP como cliente en el moodle de la asignatura.

Otras funciones útiles

- htons / ntohs:

El protocolo DNS ya no es en modo texto como lo era HTTP. Por tanto, los campos que corresponden a números los tenemos que leer como tal (no como si se trataran de cadenas de texto como hacíamos en la práctica anterior).
Además, hay que tener en cuenta que el formato en el que se guardan los números en los PCs puede ser distinto (en particular, en las arquitecturas basadas en Intel, lo es) del formato en el que vienen en los paquetes por la red. Por eso debemos usar las funciones htons para pasar enteros de 16 bits del formato de la máquina al formato de la red, o ntohs para pasar enteros de 16 bits del formato de la red al formato de la máquina.
Por ejemplo, si suponemos que en el puntero buffer estamos construyendo el paquete de petición DNS, para escribir el campo de la cabecera número de preguntas debemos hacerlo del siguiente modo:

```
u_int16_t nquestions=htons(1);
memcpy(buffer+4,&nquestions,sizeof(u_int16_t))
```

3) Objetivo

El objeto de esta práctica es implementar un pequeño **cliente DNS** que permita obtener direcciones IP a partir de su nombre y peticiones de servidor de correo.

Para ello, nuestro cliente de DNS deberá ser capaz de mandar 'queries' a un servidor de nombres (DNS) y procesar las respuestas que devuelva. De esta manera nuestro cliente del DNS será capaz de convertir nombres de Internet (nombres de dominio) a direcciones IP.

Para realizar las peticiones ('queries') de dirección, deberemos utilizar el API de sockets para mandar datagramas UDP con las 'queries DNS' al puerto 53 de un DNS cualquiera (por ejemplo, al servidor DNS de Verisign que tiene dirección IP 64.6.64.6).

EJERCICIOS

Ejercicio 1)

Implementar un cliente DNS con una interfaz de uso similar al comando nslookup de Linux, que haga resolución directa (esto es, dado un nombre de host, nos devuelva su IP correspondiente).

Por ejemplo, resolver la dirección 'www.uam.es'.

En particular, una llamada a nuestro programa deberá ser:
$ ./nslookup www.uam.es

Y la salida deberá ser de la forma:
Server: 64.6.64.6
Address: 64.6.64.6#53

Non-authoritative answer:
Name: www.uam.es
Address: 150.244.214.237

Ejercicio 2)

Aumentar la funcionalidad del cliente, para que además resuelva alias y pueda devolver varias IPs.
Por ejemplo, resolver 'smtp.gmail.com'.

En particular, una llamada a nuestro programa deberá ser:
$ ./nslookup smtp.gmail.com

Y la salida deberá ser de la forma:
Server: 64.6.64.6
Address: 64.6.64.6#53

```
Non-authoritative answer:
smtp.gmail.com      canonical name = gmail-smtp-msa.l.google.com.
Name:               gmail-smtp-msa.l.google.com
Address:            74.125.206.108

Name:               gmail-smtp-msa.l.google.com
Address:            74.125.206.109
```

CRITERIOS DE EVALUACIÓN

1. Resolución DNS simple:

Obtención de nombre canónico/IP (e.g. www.uam.es): 4 puntos.

2. Resolución DNS compuesta:

Obtención de alias y varias IPs (e.g. smtp.gmail.com): 5 puntos.

3. Control de errores: 1 punto

¡Atención! No serán corregidas prácticas con errores de compilación o sin Makefile. Todas las pruebas se harán en entorno real. En particular, las solicitudes DNS se harán al servidor 64.6.64.6.

ENTREGAS

Las entregas se harán antes de las 00:00 del día de la siguiente clase (esto es, 23:55 del día anterior).

4) ¿Cómo empezar?

1. Leemos atentamente el enunciado: entendiendo para qué sirve el protocolo DNS, cómo funciona, cuál es la estructura de sus mensajes y qué debe hacer nuestra práctica.

2. Para reforzar el punto 1 (entender cómo funciona el protocolo DNS y cómo funciona nslookup, al cualnuestra práctica deberá "imitar"):

- Abrimos wireshark y ponemos a capturar de eth0. Ponemos el filtro de visualización 'udp.port eq 53' para que sólo nos muestre los paquetes DNS.
- Abrimos un terminal y hacemos varias pruebas con nslookup. Por ejemplo, nslookup www.uam.es, nslookup www.google.es
- Miramos en wireshark qué forman tienen tanto las peticiones que hace nslookup como las respuestas del servidor. Las diferencias que hay tanto en las respuestas como en las peticiones en los distintos tipos de peticiones.

3. Descargamos el código del cliente UDP que se proporciona. Lo leemos atentamente para entender qué hace y cómo funciona.

4. ¿Qué tenemos que modificar para el ejercicio 1?

- Construir el paquete de petición DNS siguiendo el protocolo, y enviar por el socket (en vez de enviarla cadena "Esto es un ejemplo...").
- Procesar la respuesta del servidor. Para este primer ejercicio podemos suponer que sólo habrá una respuesta dentro del paquete DNS y que no habrá alias. Pero sí puede haber compresión en la respuesta.

Se debe tener en cuenta dos aspectos:

- Estamos usando UDP como protocolo de transporte (SOCK_DATAGRAM), por lo que ahora enviamos DATAGRAMAS y no tenemos un FLUJO de datos continuo entre cliente y servidor. Esto es, ahora NO podemos ir escribiendo y leyendo del socket poco a poco como hacíamos en la práctica anterior. Sino que debemos construir toda la petición y luego mandarla.
- El protocolo DNS ya NO es en modo texto (como HTTP), sino que muchos campos de la cabecera sonbinarios y habrá que tener en cuenta las funciones htons()/ntohs() para leer y escribir enteros.