Implementazione traceroute socket con DNS

Sicurezza dei sistemi e delle reti - Classe L31

Gabriele Bottani e Fabio Ferraris

Contents

1 Introduzione

1.1 DNS

Il Domain Name System, o DNS, è il sistema che consente di tradurre i nomi di dominio, come "unimia.unimi.it", in indirizzi IP, come "192.0.2.1", rendendo possibile l'accesso alle risorse su Internet in modo più user-friendly. Il DNS opera attraverso due modalità operative principali che includono l'uso della delega, la quale permette una gestione gerarchica ed efficiente dei nomi di dominio.
In entrambe le modalità il server DNS che riceve una richiesta fornisce la migliore risposta possibile basandosi sulle informazioni nella sua cache locale o sulla conoscenza diretta. In caso contrario ci sono due possibilità a seconda della modalità scelta:

- Iterativa: in modalità iterativa, se il DNS non può fornire una risposta completa, restituirà l'indirizzo di un server DNS più appropriato da consultare. Questo processo continua fino a quando si raggiunge un server autoritativo.

- Ricorsiva: in modalità ricorsiva, se non ha la risposta in cache, inizierà una catena di richieste ai server DNS necessari, risalendo lungo la gerarchia DNS fino a raggiungere il server autoritativo appropriato.

1.2 Traceroute

Il traceroute è un comando utilizzato per "controllare" il percorso attraverso cui i dati viaggiano da un computer a un altro su una rete IP, come Internet, allo scopo di consentire agli utenti di identificare i punti di congestione o i ritardi lungo il percorso. Ecco come funziona:

1. Il computer di origine invia un numero variabile di pacchetti di dati (ad esempio 3) verso la destinazione desiderata, inserendo un valore TTL (Time-to-Live) iniziale nel pacchetto. Il TTL svolge la funzione di contatore venendo inizialmente impostato su un valore abbastanza alto (ad esempio 30).

2. Il pacchetto inizia il suo viaggio attraverso la rete. Ogni router lungo il percorso riduce il TTL del pacchetto di 1 unità ogni volta che lo inoltra.

3. Quando il TTL raggiunge zero, il router scarta il pacchetto e invia una risposta ICMP (Internet Control Message Protocol) al mittente per informarlo della scadenza del TTL.

4. Il mittente riceve questa risposta ICMP, che include l'indirizzo IP del router che ha scartato il pacchetto. Questo fornisce informazioni sul percorso che il pacchetto ha seguito fino a quel punto.

5. Il mittente quindi invia un nuovo pacchetto con un TTL incrementato di uno e ripete il processo. Questo nuovo pacchetto raggiunge il router precedente e si ferma al router successivo lungo il percorso.

6. Ripete questo processo in modo iterativo, incrementando gradualmente il TTL fino a quando il pacchetto raggiunge la destinazione desiderata.

7. Il mittente raccoglie le risposte ICMP lungo il percorso e le visualizza all'utente, fornendo un elenco dei router attraverso cui è passato il pacchetto, insieme ai tempi di ritardo (ping) per ciascun hop.

1.3 Socket

Una socket è una API (Application Programming Interface) che attraverso una combinazione di un indirizzo IP e un numero di porta consente a computer diversi di stabilire una connessione e scambiare dati tra di loro attraverso una rete, come Internet. Le socket vengono comunemente utilizzate per implementare comunicazioni su protocolli come TCP (Transmission Control Protocol) per connessioni affidabili o UDP (User Datagram Protocol) per connessioni senza stato.

1.3.1 TCP

Una socket TCP (Transmission Control Protocol) è una tipologia di socket utilizzata per la trasmissione di dati in maniera affidabile su una rete. La nostra implementazione (Figure 1) segue quella fatta in laboratorio e ne abbiamo dedicato l'utilizzo per la creazione del DNS simulato. Per l'invio e la ricezione abbiamo utilizzato le funzioni `read()` e `write()`.

```c
int socketdescriptor, new_socket;
struct sockaddr_in bind_ip_port, client_ip_port;
int bind_ip_port_length = sizeof(bind_ip_port);
int client_ip_port_length = sizeof(client_ip_port);
socketdescriptor = socket(AF_INET, SOCK_STREAM, 0);

if (setsockopt(socketdescriptor, SOL_SOCKET, SO_REUSEADDR, &optval, sizeof(optval)) == -1)
{
    perror("setsockopt");
    exit(1);
}

if (socketdescriptor < 0)
    die("client socket() error");

bind_ip_port.sin_family = AF_INET;
bind_ip_port.sin_addr.s_addr = inet_addr("127.0.0.1");
bind_ip_port.sin_port = htons(PORT);

if (bind(socketdescriptor, (struct sockaddr *)&bind_ip_port, bind_ip_port_length) < 0)
    die("client bind() error");

if ((listen(socketdescriptor, 5)) != 0)
    die("client listen() error");

new_socket = accept(socketdescriptor,
                    (struct sockaddr *)&client_ip_port, &client_ip_port_length);

if (new_socket < 0)
{
    die("client accept() error");
}
```

Figure 1: Implementazione di socket TCP

1.3.2 UDP

Le socket UDP forniscono un modo senza connessione, non affidabile e leggero per scambiare dati tra dispositivi in una rete. L'implementazione segue quella del TCP con la differenza di una flag `SOCK_DGRAM` al posto di `SOCK_STREAM` nella dichiarazione e nell'uso, generalmente, di `recvfrom()` e `sendto()`.

2 DNS simulato

2.0.1 Architettura

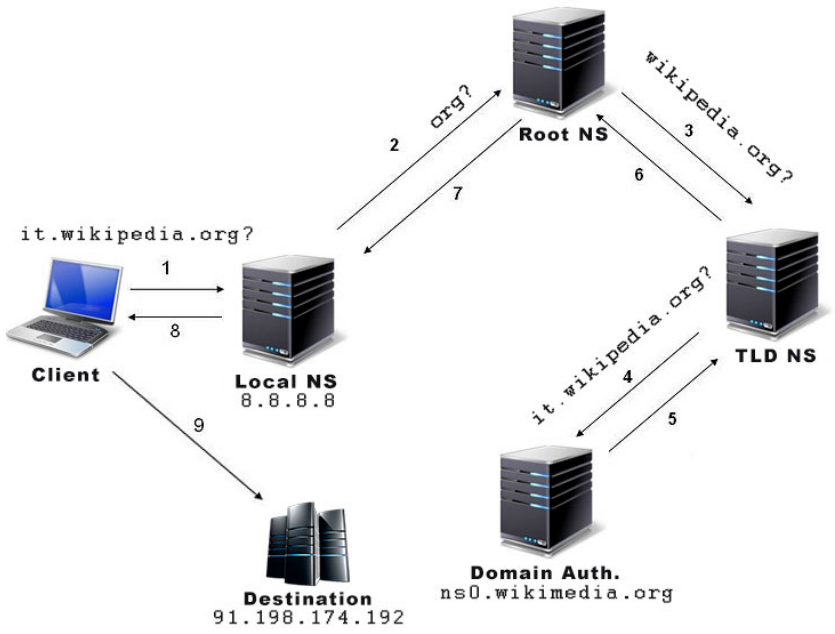


Figure 2: Schema del funzionamento del DNS simulato

Abbiamo deciso di implementare l'architettura del nostro DNS Simulato seguendo l'immagine Figure 2 usando la modalità ricorsiva e gerarchica come richiesto. Ogni NS, ad eccezione del DNS Authoritative, è realizzato tramite una socket TCP di tipo fork sia per la ricezione che l'inoltro dei dati. In particolare ogni server deve essere in grado di ricevere una richiesta e di gestirla tramite un processo `fork()` che crea un'altra socket per inoltrare la richiesta ad un ulteriore socket nella linea gerarchica. DNS Authoritative è differenziato dal fatto che, essendo l'ultimo nodo della rete, riceve e invia utilizzando la stessa socket. Il meccanismo di delega che abbiamo simulato non utilizza dei file, ma bensì delle strutture dati contenute in ogni server e riempite su inizializzazione della rete a partire dal DNS Root. Lo scambio di delega avviene tramite funzioni apposite che trasmettono dati sottoforma di stringhe formattate e poi analizzate da chi le riceve.

Abbiamo infine implementato un meccanismo di Logging (Figure 3-4) su file che utilizza lock per gestire accessi concorretti e condiviso tra tutti i nodi della rete, compreso il client. Sul file di log vengono registrate tutte le fasi di inizializzazione della rete e di gestione delle richieste del client.

```c
typedef struct {
    FILE *file;
    pthread_mutex_t mutex;
} LogFile;
```

Figure 3: Struttura per la gestione del file

```c
void *writeToFile(char* string) {

    time_t currentTime;
    char timeString[30],finalString[150];

    time(&currentTime);

    printf("%s", string);

    strftime(timeString, sizeof(timeString),
             "[%Y-%m-%d %H:%M:%S]", localtime(&currentTime));

    snprintf(finalString, sizeof(finalString), "%s %s", timeString, string);

    pthread_mutex_lock(&data->mutex); // Blocca il mutex
    fprintf(data->file, finalString);
    fflush(data->file);
    pthread_mutex_unlock(&data->mutex); // Sblocca il mutex
}
```

Figure 4: Funzione per la scrittura su file

2.0.2 DNS Locale

Il DNS Locale è il primo server DNS che viene contattato quando il client richiede di risolvere un dominio in indirizzo ip. Al suo interno è contenuta una cache, inizialmente vuota, nella quale sono contenute tutte le coppie domain-address che sono state richieste più recentemente. Una volta contattato, il DNS Locale controlla quindi che il dominio non sia già presente nella cache, in caso contrario procede con il contattare il suo DNS Root (Figure 5).

```
getClientResponse(new_socket);

found = false;
for (i = 0; i <= locaDatabase->length; i++)
{
    //printf("%s con %s\n", bufferInput, locaDatabase->domains[i]);
    if (strcmp(bufferInput, locaDatabase->domains[i]) == 0)
    {
        strcpy(bufferOutput, locaDatabase->addresses[i]);
        found = true;
        sendClientResponse(new_socket);
    }
}

if (!found)
{
    snprintf(logoutput, sizeof(logoutput),
             "DNSLocale didn't find the address: %s in its database\n", bufferInput);
    writeToFile(logoutput);

    forwardToAnotherServer();

    if (strcmp(bufferOutput, "NOT_EXISTS") != 0)
    {
        strcpy(locaDatabase->domains[locaDatabase->length + 1], bufferInput);
        strcpy(locaDatabase->addresses[locaDatabase->length + 1], bufferOutput);
        locaDatabase->length = locaDatabase->length + 1;
    }

    sendClientResponse(new_socket);
}
```

Figure 5: Funzionamento della cache locale

```
[2023-09-02 11:58:46] DNSLocale client accepted
[2023-09-02 11:58:46] DNSlocale data received from client: www.google.it
[2023-09-02 11:58:46] DNSLocale didn't find the address: www.google.it in its database
[2023-09-02 11:58:46] DNSLocale asking for the address: www.google.it to DNSRoot on port 7778
[2023-09-02 11:58:46] DNSRoot client accepted
[2023-09-02 11:58:46] DNSRoot data received from DNSLocale: www.google.it
```

Figure 6: Log di funzionamento del DNSLocale

2.0.3 DNS Root

Il DNS Root svolge il compito di "indicizzatore" ovvero inoltra una richiesta dove si trova la risposta. Una volta che il DNS Locale contatta il DNS Root, esso sa esattamente a quale TLD (Top Level Domain) server inoltrare i dati per ottenere ciò che ha chiesto il client. Questo grazie al sistema di delega che, su inizializzazione della rete simulata, prende luogo. Preliminarmente il DNS Root ha un Database con tutte le coppie domain-address in possesso della rete, le quali sono immagazzinate all'interno della struttura dati `TLDDatabase` (Figure 7a). Al fine di simulare il funzionamento della delega, abbiamo riempito il database con due tipi di indirizzi differenziati dalla zona di appartenenza: "org" e "it". Viene successivamente effettuata l'operazione di delega, tramite una funzione randomica si assegna, con l'ausilio di un'ulteriore struttura dati `DELDatabase` (Figure 7b), a ciascuna zona una porta di un server DNS TLD, al quale verranno poi delegate tutte le coppie appartenenti.

```
db TLDDatabase = {
// Inizializzazione del primo array di stringhe
{
    "www.wikipedia.org",
    "it.wikipedia.org",
    "www.un.org",
    "it.un.org",
    "www.google.it",
    "ftp.google.it",
    "unimia.unimi.it",
    "ariel.unimi.it"
},
// Inizializzazione del secondo array di stringhe
{
    "185.15.58.224",
    "185.15.58.224",
    "185.15.58.224",
    "192.168.1.4",
    "185.15.58.224",
    "192.168.1.6",
    "159.149.53.172",
    "103.224.182.250"
},
.length = 7
};
```

(a) domain-address database

```
srand(time(NULL));
int *tldPorts = portsDelegator();

DELdb DELDatabase = {
    // Inizializzazione del primo array di stringhe
    {
        "it",
        "org",
    },
    // Inizializzazione del secondo array di stringhe
    {
        tldPorts[0],
        tldPorts[1],
    },
    .length = 1
};
```

(b) delegation database

Figure 7: Strutture dati iniziali del DNS Root

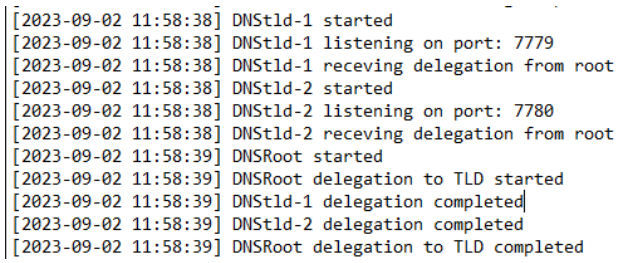


Figure 8: Log di funzionamento della delegazione tra Root e TLD

La fase di delega si compone di due passi fondamentali:

- Delegazione dei Top-Level Domains:
  - Viene creato un elenco di second level domain unici prelevati dal database `TLDDatabase` e ne si salva il conteggio.
  - Per ogni dominio di second level domain, il codice ricerca il corrispondente top level domain nella struttura `DELdatabase` e lo associa alla porta del TLD scelto per quel top level domain.
  - Il dominio viene inoltrato al server TLD sulla porta specifica.
  - Alla fine, viene inviato un messaggio `INIT_DEL_COMPLETED` ai server TLD per segnalare la fine della fase di delegazione.
- Aggiornamento del Database dei TLD:
  - Ogni coppia dominio-indirizzo viene formattata come una stringa e inviata al TLD a cui appartengono tramite la porta specificata nel `DELDatabase`.
  - Dopo averle inviate tutte viene inoltrato un messaggio `INIT_TLD_COMPLETED` a ciascun server TLD.

Finita la fase di delegazione iniziale, ogni qualvolta che il DNS Root riceverà una richiesta, analizzerà il top level domain ricevuto e controllerà sul proprio `DELDatabase` a quale porta, quindi a quale TLD, inoltrare la richiesta. Se il top level domain estratto non risulterà appartenere a nessuno di quelli registrati, allora il DNSRoot procederà a rispondere immediatamente al client con una messaggio `NOT_EXISTS`.

2.0.4 DNS TLD

Il DNS TLD rappresenta il DNS responsabile di zone denotate da top level domain come "it" e "org" occupandosi della traduzione da domain name a ip di tutte le risorse sotto i domini a loro assegnati e delegati ad uno specifico Authoritative DNS. Nell'architettura da noi implementata sono presenti 2 TLD DNS.
All'interno di ciascun DNS TLD sono presenti 2 strutture simili a quelle descritte nel DNS Root con le stesse medesime funzioni: `TLDDatabase` e `DELDatabase` (Figure 23a-23b).

```
struct delegationDB
{
    char domains[MAXLEN][MAXLEN];
    int ports[MAXLEN];
    int length;
};
typedef struct delegationDB DELdb;
```

(a) delegation database

```
struct database
{
    char domains[MAXLEN][MAXLEN];
    char addresses[MAXLEN][16];
    int length;
};
typedef struct database db;
```

(b) domain-address database

Figure 9: Strutture dati del DNS TLD

Il database andrà a contenere le coppie nome di dominio-ip, ottenute dal DNS Root, che verranno inoltrate a ciascun DNS Authoritative. Il delegatioDB contiene, invece, il dominio di cui ciascun DNS Authoritative sottostante è responsabile associandolo tramite la porta attraverso cui il DNS TLD e l'AUTHORITATIVE possono comunicare.

Grazie alla funzione `getDelegationFromRoot()`, ciascun DNS TLD rimane in attesa dei dati di inizializzazione provenienti dal DNS Root. Il processo di delega tra DNS Root e DNS TLD è spiegata nella precedente sezione.
Una volta completata la ricezione di dati dal DNS Root, ciascun DNS TLD si occupa di delegare, tramite la funzione `delegateAuthoritatives()`, a ciascuno dei due DNS Authoritative sottostanti i SLD corrispondenti. Il procedimento è il seguente:

- Tramite un ciclo si scorrono tutti i domini contenuti in `TLDDatabase` e:
  - Si costruisce la stringa contenente la coppia domain-address da inviare all'Authoritative scelto.

  - Con l'ausilio della funzione `extractSecondLevelDomain()` si esamina il second level domain e, tramite il `DELDatabase`, si verifica a quale porta corrisponde.
  - Si procede quindi con l'inviare la stringa tramite la porta trovata e si rimane in attesa di una risposta di conferma.

- Finiti i domini in `TLDDatabase` si procede con l'inviare il messaggio `INIT_COMPLETED` ai propri DNS Authoritative.

Una volta completato questo processo di inizializzazione, ciscun DNS TLD rimane in attesa di richieste trasmesse dal DNS Locale attraverso il DNS Root. Ricevuta una richiesta si procede a controllare il relativo SLD per identificare il DNS Authoritative a cui inoltrarla, rimanendo poi in attesa di risposta da quest'ultimo prima di rispondere al DNS Root.

```
[2023-09-02 11:58:39] DNStld-1 delegation completed
[2023-09-02 11:58:39] DNStld-1 delegation to authoritatives started
[2023-09-02 11:58:39] DNSauthoritative-3  client accepted
[2023-09-02 11:58:39] DNStld-2 delegation to authoritatives started
[2023-09-02 11:58:39] DNSauthoritative-4  client accepted
[2023-09-02 11:58:39] DNSauthoritative-3 receving delegation from TLD
[2023-09-02 11:58:39] DNSauthoritative-4 receving delegation from TLD
[2023-09-02 11:58:39] DNSauthoritative-3  client accepted
[2023-09-02 11:58:39] DNSauthoritative-4  client accepted
[2023-09-02 11:58:39] DNSauthoritative-4 receving delegation from TLD
[2023-09-02 11:58:39] DNSauthoritative-1  client accepted
[2023-09-02 11:58:39] DNSauthoritative-3 receving delegation from TLD
[2023-09-02 11:58:39] DNSauthoritative-2  client accepted
[2023-09-02 11:58:39] DNSauthoritative-1 receving delegation from TLD
[2023-09-02 11:58:39] DNSauthoritative-1  client accepted
[2023-09-02 11:58:39] DNSauthoritative-2 receving delegation from TLD
[2023-09-02 11:58:39] DNSauthoritative-2  client accepted
[2023-09-02 11:58:39] DNSauthoritative-2 receving delegation from TLD
[2023-09-02 11:58:39] DNStld-1 delegation to authoritatives completed
[2023-09-02 11:58:39] DNSauthoritative-3  client accepted
[2023-09-02 11:58:39] DNSauthoritative-1 receving delegation from TLD
[2023-09-02 11:58:39] DNStld-2 delegation to authoritatives completed
[2023-09-02 11:58:39] DNSauthoritative-3 delegation completed
[2023-09-02 11:58:39] DNSauthoritative-4  client accepted
[2023-09-02 11:58:39] DNSauthoritative-2  client accepted
[2023-09-02 11:58:39] DNSauthoritative-2 delegation completed
[2023-09-02 11:58:39] DNSauthoritative-4 delegation completed
[2023-09-02 11:58:39] DNSauthoritative-1  client accepted
[2023-09-02 11:58:39] DNSauthoritative-1 delegation completed
```

Figure 10: Log di funzionamento della delegazione tra TLDs e Authoritatives

2.0.5 DNS Authoritative

I DNS Authoritative rappresenta l'ultimo componente della gerarchia del DNS essendo, gli stessi, i server dedicati alle traduzioni in indirizzi ip di specifiche risorse contenute all'interno della loro zona di delega (es. se il DNS Authoritative si occupa del SLD google.it al proprio interno avrà le corrispettive traduzioni in ip dei domini www.google.it,ftp.google.it, ecc...).

La struttura `localDatabase` (Figure 23b) è l'unica e contiene le coppie di nome di domain-address di cui il DNS Authoritative è responsabile. Quando una richiesta viene inoltrata dal DNS TLD, il DNS Authoritative non fa altro che controllare se esiste il corrispondente indirizzo ip all'interno del proprio database. In caso affermativo risponde con l'indirizzo ip, altrimenti risponde col messaggio `NOT_EXISTS`.
La funzione principale del DNS Authoritative è `HandleRequests()` la quale è utilizzata sia per l'inizializzazione del database sia per ricevere e rispondere alle richieste del DNS Locale giunte attraverso il DNS TLD.

```
void getDelegationFromTLD(db* localDatabase, char* bufferInput){
    int i = localDatabase->length;

    char *domain,*address,*separator = "+";
    domain = strtok(bufferInput, separator);
    address = strtok(NULL, separator);

    strcpy(localDatabase->domains[i], domain);
    strcpy(localDatabase->addresses[i], address);
    localDatabase->length++;

}
```

Figure 11: Funzione di ricezione della delega

```
void handleResponse(int sd, db* localDatabase, char* bufferInput, short *initialized)
{
    int data,send,i;
    char logoutput[MAXLEN];
    bool found;
    data = read(sd, bufferInput, MAXLEN);
    printf("BufferInput: %s\n", bufferInput);

    if(strcmp(bufferInput, "INIT_COMPLETED") == 0){
        writeToFile("DNSauthoritative-1 delegation completed\n");
        *initialized = 1;
        send = write(sd, "AUTH-1: INIT OK", MAXLEN);
        if (send < 0)
            die("address send() error");
        return;
    }

    if(!*initialized){
        writeToFile("DNSauthoritative-1 receving delegation from TLD\n");
        getDelegationFromTLD(localDatabase, bufferInput);
        send = write(sd, "AUTH-1: Entry added", MAXLEN);
        if (send < 0)
            die("address send() error");
    }
    else{
        found = false;
        snprintf(logoutput, sizeof(logoutput), "DNSauthoritative-1 data received from TLD: %s\n", bufferInput);
        writeToFile(logoutput);
        for(i = 0; i<=localDatabase->length; i++){
            if(strcmp(bufferInput, localDatabase->domains[i]) == 0){
                found = true;
                snprintf(logoutput, sizeof(logoutput),
                            "DNSauthoritative-1 sending response to TLD: %s\n", localDatabase->addresses[i]);
                writeToFile(logoutput);
                send = write(sd, localDatabase->addresses[i], MAXLEN);
                if (send < 0)
                    die("address send() error");
            }
        }
        if(!found){
             send = write(sd, "NOT_EXISTS", MAXLEN);
            writeToFile("DNSauthoritative-1 sending response to DNSRoot: NOT_EXISTS");
            if (send < 0)
                die("NOT_EXISTS send() error");
        }
    }
}
```

Figure 12: Funzione di inizializzazione e di gestione richieste del DNS Authoritative

3 DNS esterno

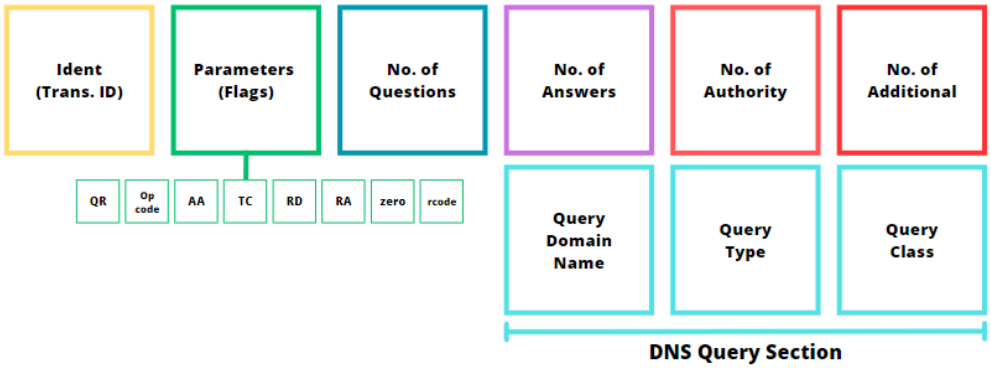


Figure 13: Schema del formato di un messaggio DNS

Per effettuare una richiesta ad un server DNS è necessaria per prima cosa la costruzione del pacchetto da inviare. In particolare, seguendo lo schema in Figure 13, abbiamo composto ed utilizzato delle strutture C con gli stessi medesimi campi (Figure 14-15), per poi inizializzarle seguendo le direttive standard.
`Header`, `Query Domain Name`, `Query Type` e `Query Class` vengono quindi inseriti in un pacchetto, rappresentato da un array di `unsigned char` con grandezza massima 65536. Per una questione di ottimizzazione, il `Query Domain Name` viene convertito tramite la tecnica del DNS name compression la quale viene utilizzata all'interno del protocollo DNS per abbreviare la rappresentazione dei nomi di dominio, riducendo così la quantità di dati necessaria nelle risposte DNS. Per l'ottenimento dell'indirizzo del DNS locale, utilizziamo una funzione che va direttamente a cercare all'interno del sistema operativo il file `/etc/resolv.conf` ed estrae il suddetto indirizzo presente nel file. Dopo aver creato il pacchetto ed ottenuto l'ip del DNS locale, si può procedere con l'invio dei dati tramite socket UDP alla porta standard 53.

```
/* Costruzione della porzione di header del pacchetto DNS */
header = (HEADER *)&packet;
header->id = (unsigned short)htons(getpid());
header->qr = 0;
header->opcode = 0;
header->aa = 0;
header->tc = 0;
header->rd = 1;
header->ra = 0;
header->z = 0;
header->rcode = 0;
header->qdcount = htons((unsigned short)(1)); //1 = numero di domini
header->ancount = 0x0000;
header->nscount = 0x0000;
header->arcount = 0x0000;
```

Figure 14: Inizializzazione dell'Header DNS

```
/* Costruzione delle flags della query nel pacchetto DNS*/
qflags = (Q_FLAGS *)&packet[steps];
qflags->qtype = htons(0x0001);
qflags->qclass = htons(0x0001);
```

Figure 15: Inizializzazione delle QFLAGS DNS

Una volta ricevuta la risposta, si procede con l'analizzare il pacchetto estraendo, in ordine di posizionamento, all'interno di esso, i seguenti campi: `Header`, `Query Domain Name`, `Query Flags` e `Resource Records`. Questi ultimi vengono posizionati in una appropriata struttura dati (Figure 16) per essere ulteriormente analizzati, in quanto essi contengono la risposta del DNS. Tra le informazioni contenute all'interno dei Resource Records ha di particolare importanza l'ip del dominio di cui abbiamo fatto richiesta. Esso, una volta estratto, viene prima posizionato in un `unsigned char array` contenente tutti i dati dei Resource Records e poi prelevato nuovamente e inserito in un `int array` al cui interno ci saranno gli interi dell'indirizzo ip, senza punti divisori. Infine viene processato da un algoritmo che inserisce i punti tra i vari interi per farlo diventare un effettivo indirizzo ip che verrà poi restituito dalla funzione.

```c
//Struct per le flags dei resource records
typedef struct rr_flags {
    unsigned short type;      // Tipo di record (16 bit)
    unsigned short class;     // Classe del record (16 bit)
    unsigned int ttl;         // Tempo di vita del record (32 bit)
    unsigned short rdlength;  // Lunghezza dei dati del record (16 bit)
} RR_FLAGS;
```

Figure 16: Struttura dati per contenere le flags dei Resource Records

4 Traceroute

```c
socketdescriptor = socket(AF_INET, SOCK_RAW, IPPROTO_ICMP);
if (socketdescriptor < 0) {
    die("socket error");
}
```

Figure 17: Socket di tipo RAW ICMP

Dopo l'aver ottenuto un indirizzo ip, ed averne verificato l'integrità, è possibile iniziare il processo di traceroute. Per fare ciò abbiamo quindi istanziato una socket (Figure 17) di tipo `RAW` [1] con protocollo `ICMP` che verrà usata durante tutta l'operazione di traceroute. Essa consiste principalmente in un ciclo (Figure 18) che rimarrà in loop fino a che il `ttl` non raggiungerà un valore uguale agli `HOPSMAX` massimi impostati.
Durante questo ciclo vengono eseguite tre operazioni principali:

- Invio dei pacchetti ICMP: invio di `nqueries` pacchetti ICMP verso la destinazione con un certo valore di `ttl`.
- Attesa delle risposte ICMP: attesa delle risposte ICMP corrispondenti ai pacchetti inviati in precedenza. La funzione tiene traccia del tempo iniziale e del tempo finale per ciascun pacchetto inviato e verifica se sono state ricevute risposte durante l'intervallo di tempo specificato.

[1] Per socket di tipo raw si intende una socket che consente di inviare e ricevere pacchetti a livello IP senza che il livello di trasporto (ad esempio, TCP o UDP) gestisca la comunicazione.

- Verifica dell'host raggiunto: dopo aver atteso le risposte ICMP, il programma verifica se ha raggiunto l'host di destinazione.

```
nqueries = 3;
for (ttl = 1; ttl <= HOPSMAX; ttl++) {
    gettimeofday(&start_time, NULL);
    end_time = start_time;
    end_time.tv_sec++;

    for (i = 0; i < nqueries; i++) {
        send_single_icmp(socketdescriptor, dnsoutput, pid, ttl, ttl);
    }

    host_reached = wait_for_icmps(socketdescriptor, pid, ttl, &start_time, &end_time, nqueries);
    if (host_reached) {
        break;
    }
}
```

Figure 18: Ciclo dedicato al funzionamento dell'operazione di traceroute

Relativamente alla prima operazione, ovvero l'invio dei pacchetti ICMP, essa viene svolta utilizzando la funzione `send_single_icmp()` (Figure 19) che utilizza una struttura C standard per la creazione di pacchetti di tipo ICMP, in particolare per settarne il relativo header. In breve, dopo aver inizializzato la struttura con tutti i parametri adeguati, viene calcolato il checksum con un'altra funzione apposita e viene settato, prima dell'invio del pacchetto, un valore di TTL tramite `setsockopt()`.

```c
void send_single_icmp(int sockfd, const char *ip, uint16_t id, uint16_t sequence, int ttl) {
    struct icmphdr icmp_header;         // Struttura per l'header ICMP
    struct sockaddr_in recipient;      // Struttura per l'indirizzo del destinatario

    icmp_header.type = ICMP_ECHO;       // Tipo di messaggio ICMP (Echo Request)
    icmp_header.code = 0;               // Codice ICMP (0 per Echo Request)
    icmp_header.un.echo.id = id;
    icmp_header.un.echo.sequence = sequence;
    icmp_header.checksum = 0;
    // Calcola e imposta il checksum
    icmp_header.checksum = compute_icmp_checksum((uint16_t *)&icmp_header, sizeof(icmp_header));

    bzero(&recipient, sizeof(recipient));
    recipient.sin_family = AF_INET;
    inet_pton(AF_INET, ip, &recipient.sin_addr);

    if (setsockopt(sockfd, IPPROTO_IP, IP_TTL, &ttl, sizeof(int)) != 0) {
        die("setsockopt error");  // Imposta il valore TTL per il socket
    }

    if (sendto(sockfd, &icmp_header, sizeof(icmp_header), 0,
        (struct sockaddr *)&recipient, sizeof(recipient)) < 0) {
        die("sendto() error");   // Invia il pacchetto ICMP tramite il socket
    }
}
```

Figure 19: Funzione per l'invio di un pacchetto ICMP

```c
struct timeval deltas[nqueries];  // Array per memorizzare i ritardi
struct timeval current_time;

printf("%d. ", ttl);

gettimeofday(&current_time, NULL);
```

Figure 20: Istruzioni per la gestione dei ritardi di ricezione dei pacchetti

Per la ricezione delle risposte ICMP invece utilizziamo la funzione `wait_for_icmps()` che implementa il processo di attesa e acquisizione dei pacchetti ICMP inviati in risposta a richieste precedenti tramite l'uso di una operazione di `select()`. Questa funzione è responsabile di gestire il tempo di ricezione dei pacchetti e per questo motivo fa uso di strutture di tipo `timeval` e della funzione `gettimeofday()` per aspettare i pacchetti o fornire altri risultati (Figure 20). Il ciclo principale all'interno della funzione (Figure 21) viene eseguito finché non sono stati ricevuti un numero specificato di pacchetti ICMP o finisce il tempo massimo di attesa `end_time`. Durante il ciclo, la funzione utilizza diverse chiamate di sistema per gestire la socket, inclusi `select()` per verificare la disponibilità dei dati nel socket e `recvfrom()` per ricevere i pacchetti ICMP in arrivo.

```c
while (!time_passed(packets_received, &current_time, end_time, nqueries)) {
    struct sockaddr_in sender;       // Informazioni sul mittente
    socklen_t sender_len = sizeof(sender);
    uint8_t buffer[IP_MAXPACKET];    // Buffer per i dati del pacchetto

    fd_set descriptors;
    FD_ZERO(&descriptors);
    FD_SET(sockfd, &descriptors);
    struct timeval tv;
    timersub(end_time, &current_time, &tv);

    // Verifica la disponibilita  dei dati
    int ready = select(sockfd + 1, &descriptors, NULL, NULL, &tv);
    if (ready < 0) {
        die("select() error");
    }
    if (ready == 0) {
        break;  // Esci se il timeout è scaduto
    }

    ssize_t packet_len = recvfrom(sockfd, buffer, IP_MAXPACKET, 0,
        (struct sockaddr *)&sender, &sender_len);  // Ricevi il pacchetto ICMP
    if (packet_len < 0) {
       die("recvfrom() error");  // Gestione dell'errore nella ricezione
    }
```

Figure 21: Ciclo principale per la ricezione dei pacchetti

```c
gettimeofday(&current_time, NULL);  // Aggiorna il tempo corrente

char sender_ip_str[20];
const char *inet_ntop_ret = inet_ntop(AF_INET, &(sender.sin_addr),
                                      sender_ip_str, sizeof(sender_ip_str));  // Ottieni l'indirizzo IP del mittente

struct iphdr *ip_header = (struct iphdr *) buffer;
ssize_t ip_header_len = 4 * ip_header->ihl;  // Calcola la lunghezza dell'header IP

struct icmphdr *icmp_ptr = (struct icmphdr *)(buffer + ip_header_len);  // Puntatore all'header ICMP

uint8_t icmp_type = icmp_ptr->type;  // Tipo di messaggio ICMP
int proper_type = icmp_type == ICMP_TIME_EXCEEDED || icmp_type == ICMP_ECHOREPLY;  // Verifica se il tipo è corretto

if (icmp_type == ICMP_TIME_EXCEEDED) {
    struct iphdr *inner_ip_header = (void *) icmp_ptr + 8;  // Header IP interno (nel caso di "Time Exceeded")
    ssize_t inner_ip_header_len = 4 * inner_ip_header->ihl;
    icmp_ptr = (void *)inner_ip_header + inner_ip_header_len;  // Puntatore all'header ICMP interno
}
```

Figure 22: Estrazione dei dati contenuti nell'header ICMP

Nella fase di ricezione delle risposte ICMP (Figure 22) è molto importante dover effetuare un'analisi del pacchetto IP in quanto l'header ICMP è contenuto al suo interno. Dopo aver estrapolato l'ip del mittente, si procede quindi con il calcolare la lunghezza dell'header IP per trovare la posizione di quello

ICMP. Una volta trovata si verifica il tipo di messaggio ICMP e, se è di tipo `ICMP_TIME_EXCEEDED`, sarà necessario effettuare una seconda estrazione di dati. Questo è dovuto al fatto che l'header ICMP "Time Exceeded" contiene l'header IP del pacchetto originale che ha superato il limite di TTL e non è riuscito a raggiungere la destinazione. Questo è chiamato "header IP interno". Esso consente di sapere quale pacchetto originale ha superato il TTL e quale era la destinazione prevista. Una volta verificato il tipo, si controlla anche che l'id e la sequenza siano corretti. Se queste condizioni sono soddisfatte si calcola il ritardo di arrivo del pacchetto e si stampa l'indirizzo IP del mittente.
Infine, se il messaggio ICMP è di tipo `ICMP_ECHO_REPLY` significa che l'host è stato raggiunto e viene quindi aggiornata e ritornata la variabile dedicata in modo da terminare l'operazione di traceroute.

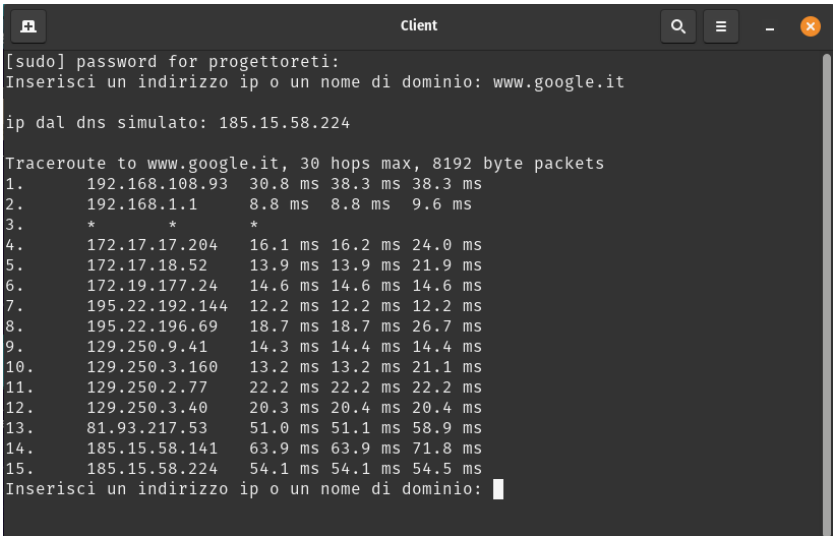
5 Client

Il Client è la parte fondamentale dell'intero applicativo in quanto ingloba:

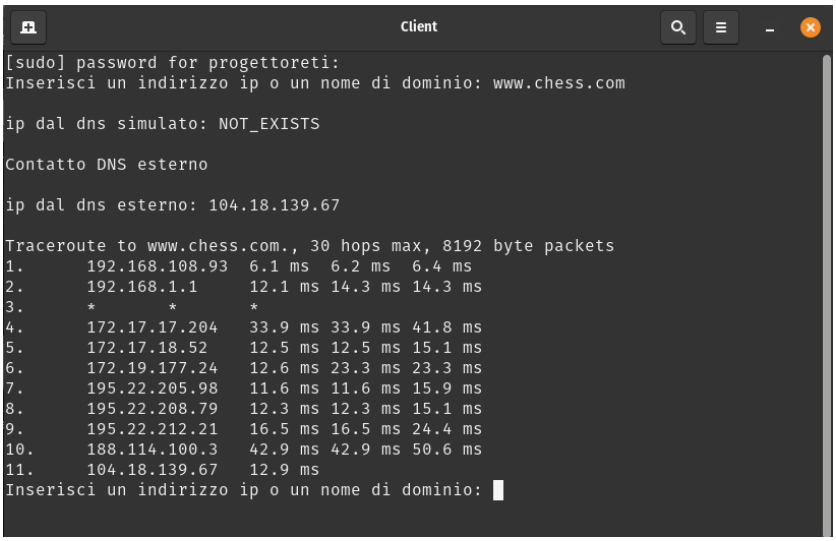
- Richiesta DNS a rete simulata, sezione 2.
- Richiesta DNS al server DNS Locale reale, sezione 3.
- Implementazione traceroute, sezione 4.

Esso è inoltre l'unica interfaccia che viene utilizzata dall'utente e con il quale può interagire. Come già spiegato nella sezione 2.0.1 anche il Client utilizza un sistema di logging che sfrutta il medesimo file utilizzato dai nodi della rete DNS simulata.

L'input ricevuto dall'utente viene analizzato (Figure 24) per verificarne se è necessaria una traduzione in indirizzo ip o meno. In particolare, se il primo carattere è un intero si procederà direttamente con il traceroute verso l'ip specificato, se ritenuto valido. In caso contrario il Client procederà con il richiedere la traduzione alla rete simulata: se il valore ritornato dalla rete è `NOT_EXISTS` allora si procederà con l'effettuare una richiesta al DNS Locale reale, altrimenti si invierà l'indirizzo ip alla funzione dedita al traceroute.



(a) Esempio utilizzo DNS Simulato



(b) Esempio utilizzo DNS Reale

Figure 23: Esecuzione del Client

```
printf("Inserisci un indirizzo ip o un nome di dominio: ");
writeToFile("-------------------------------------\n");

scanf("%s", sendbuff);

if(isFirstCharacterChar(sendbuff)){
    strcpy(dnsoutput, callDNS(sendbuff));

    printf("\nip dal dns simulato: %s\n", dnsoutput);

    if(strcmp(dnsoutput, "NOT_EXISTS") == 0){
        printf("\nContatto DNS esterno\n");
        strcpy(dnsoutput, callEXTDNS(sendbuff));
        printf("\nip dal dns esterno: %s\n", dnsoutput);
    }
}else{
    strcpy(dnsoutput, sendbuff);
}

struct in_addr converted_ip;
if (inet_pton(AF_INET, dnsoutput, &converted_ip) != 1) {
    fprintf(stderr, "IP address not valid!\nUsage: %s <IPv4 address>\n", dnsoutput);
    return EXIT_FAILURE;
}else{
    sendbuff[strlen(sendbuff)] = '\0';
    printf("\nTraceroute to %s, %d hops max, %d byte packets\n", sendbuff, HOPSMAX, IP_MAXPACKET/8);
}
```

Figure 24: Funzionamento Client

6 Conclusioni

Il progetto nasce dall'idea di approfondire il funzionamento delle socket per l'implementazione di DNS, simulato e non, e di un applicativo traceroute. E' stato interessante il dover sviluppare sia un'architettura DNS simulata e sia una richiesta DNS come quelle utilizzate nei sistemi operativi odierni. Non abbiamo riscontrato particolari problematiche durante lo sviluppo eccezion fatta per la condivisione di memoria tra i vari processi padre e figli generati dalla chiamata a sistema `fork()`, ma ciònonostante siamo riusciti a risolverlo sfruttando la libreria `sys/shm.h` che ci ha permesso di allocare strutture dati nello heap per poterle far referenziare tramite dei puntatori usati dai vari processi.
Siamo soddisfatti del risultato in quanto l'applicativo è molto simile alle controparti che si trovano su sistemi come Linux e Windows, anche se con qualche differenza.