Raport Tehnic

Mihai Razvan-Ionut

Facultatea de Informatica Iasi, Strada General Henri Mathias Berthelot 16, Iași 700483

Abstract. Acest document prezinta proiectul yaDns, un prototip de sistem bazat pe protocolul Domain Name Space [DNS] dezvoltat de Mihai Razvan-Ionut in cadrul Facultatii de Informatica Iasi pentru cursul de Retele de Calculatoare.

Keywords: DNS · Multithreading · Database · Transport Protocol

1 Introducere

Protocolul DNS a fost dezvoltat in urma expansiunii exponentiale ale utilizatorilor internetului. Acest protocol a fost proiectat cu mare atentie si pune accent pe scalabilitate, viteza de acces si mentenanta usor de efectuat.

Fiecare utlizator al internetului foloseste acest sistem de fiecare data cand acceseaza o resursa pe internet. Desi toti utilizatorii folosesc protocolul DNS, foarte putin dintre acestia isi dau seama de existenta acestuia. Fapt datorat de maniera in care protocolul DNS a fost proiectat. Acest protocol prezinta multe optimizari care il fac sa fie ascuns intr-o oarecare maniera de catre utilizatorul de rand.

Proiectul yaDns este un prototip de sistem DNS care urmareste sa dezvolte o arhitectura de sistem DNS scalabila si usor de inteles care reflecta conform protocolului DNS descris in RFC1034, RFC1035 interactiunea intre toate componentele ce alcatuiesc sistemele DNS.

Obiectivul principal al acestui proiect este de a fi un model educational pentru autor si pentru cititorii acestui document de a intelege in profunzime cum functioneaza protocolul DNS si toate aspectele ale acestuia.

2 Tehnologii Aplicate

2.1 Protocol de transport:

Datorita obiectivelor proiectului yaDns de a urmari in exactitate protocolul DNS toate serverele care compun acest sistem for folosi drept protocol de transport implicit protocolul UDP. Aceasta alegere este motivata de faptul ca mesajele transmise intre componentele acestui sistem sunt mici din punctul de vedere al marimii. Protocolul UDP permite sistemului DNS un transfer rapid fara latenta, care urmareste sa rezolve cerintele utilizatorilor fara intarzieri majore.

Mesajele transmise intre componentele protocolului DNS pot avea maxim 512 octeti atunci cand este utilizat UDP. In ciuda acestei limite, in cadrul mesajului exista un bit numit TC (Truncated Message) care daca este setat pe 1 comunica celui ce primeste acest mesaj faptul ca mesajul a fost trunchiat.

Desi protocolul DNS utilizeaza implicit UDP, acesta poate folosi TCP in cazul in care atat serverul cat si resolverul care proceseaza cerinta utilizatorului au implementate si protocolul de transport TCP. Acest protocol de transport poate fi utilizat atunci cand mesajul primit a fost trunchiat si nu reflecta in totalitate nevoile utilizatorului.

2.2 Modul de procesare a interogarilor

Protocolul DNS nu impune o anumita arhitectura precisa de gestionare a interogarilor, dar impune procesarea acestora intr-o maniera concurenta. Orice server ce se gaseste a fi o componenta intr-un sistem DNS va gestiona o multime de interogari intr-un timp foarte scurt (la nivel de secunde, milisecunde). Fapt datorat de complexitatea modului in care internetul se doreste a fi accesat de o multime de utilizatori la nivel global.

Datorita acestui fapt, yaDns va impune pentru toate serverele incluse in acest sistem o arhitectura bazata pe multithreading. Fiecare server va utiliza POSIX Threads pentru a gestiona interogari concurent. Aceste threaduri vor fi utilizate sub forma unui pool de threaduri. Aceasta grupare de threaduri va fi utilizata in acelasi timp cu o coada de interogari. Atunci cand serverul primeste o interogare el o va plasa in coada alaturi de informatii legate de cel care a plasat aceasta interogare. Fiecare thread asteapta ca aceasta coada de interogari sa primeasca o interogare, iar atunci cand una este plasata in coada unul dintre threaduri va incerca sa o rezolve. In dezvoltarea sistemului a fost aleasa utilizarea threadurilor in locul proceselor doarece atat in cazul unui resolver cat si a unui server exista mai multe resurse partajate precum un cache pentru resolver, un master file care contine componenta bazei de date pentru un anumit nod din aceasta gestionat de un name server, lista de name servere la care are acces un resolver, etc. O buna parte din aceste componente vor fi stocate atat pe disk cat si in memoria serverului pentru a realiza rezolvarea interogarilor intr-un timp cat mai scurt.

3 Structura Aplicatiei

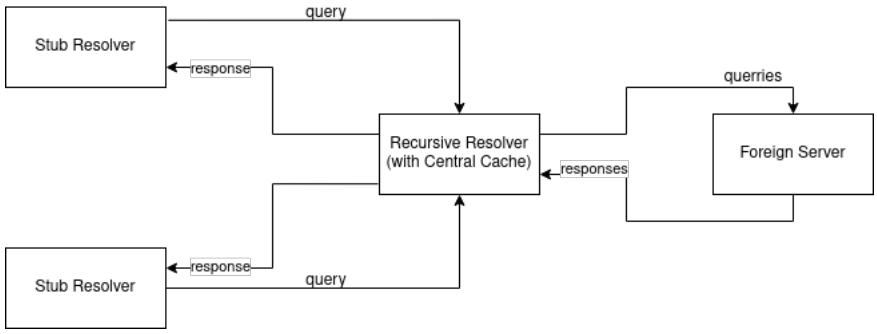
Un sistem DNS complet prezinta o arhitectura complexa ce are in componenta sa clienti, resolvere si name servere. La baza un sistem DNS este o baza de date distribuita care are mecanisme de interogare si de raspuns la aceste interogari. Sistemul yaDns este structurat dupa aceasta arhitectura dezolvtata de protocolul DNS.

Un resolver primeste o interogare de la un client pe care o va incerca sa o rezolve in mod concurent si recursiv. Acest resolver are acces la mai multe name servere din sistemul yaDns.

Baza de date a sistemului yaDns va fi proiectata sub forma unui arbore care va incepe de la radacina mentinuta de un root server si se va bifurca apoi in top domain levels. In proiectarea acestei baze de date se urmareste redundanta impusa de protocolul DNS pentru a asigura constant accesul la aceasta baza de data fara intreruperi, deci pentru acest nod radacina vor exista macar doua root servere care vor avea acces la informatia acestui nod. Deocamdata aceasta baza de date este rudimentara si contine un root server care va permite mai apoi accesul la noduri de nivel mai mic.

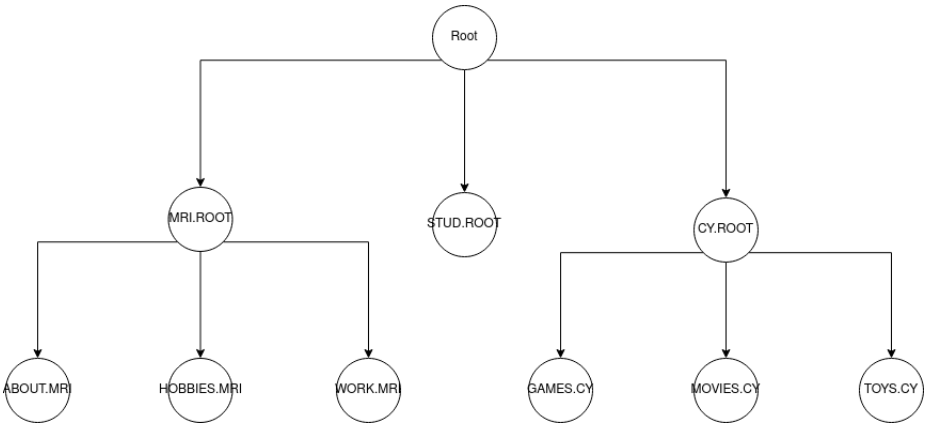
Fiecare nod din domain space va avea informatii despre el insusi si despre nodurile care sunt sub nivelul acestuia sub forma de resource records. Aceste resource records vor fi componentele ce vor ajuta in rezolvarea unei interogari.

Urmatoarea diagrama prezinta structura unui resolver:



Un foreign server este orice name server la care un resolver are acces. Fiecare name server gestioneaza o zona din structura de date arborescenta ce reprezinta intreaga baza de date.

Urmatoarea diagrama prezinta structura bazei de date prototip:



4 Aspecte de Implementare

In implementarea acestui proiect am mapat fiecare structura de informatii legate de protocolul DNS pentru a face gestionarea acestora cat mai lizibila.

Fiecare nod din domain space va avea o eticheta de maxim 63 de octeti. Parcurgand din nodul curent catre radacina si concatenand aceste etichete vom obtine numele domeniului. Acest nume este limitat la 255 de octeti. Toate aceste limitari sunt impuse de protocolul DNS.

Urmatoarea sectiune de cod mapeaza tipurile de interogari, clasele si resource records conform protocolului DNS:

```
typedef enum QType {
    A        = 1,   //  host adress
    NS       = 2,   //  authoritative name server
    CNAME    = 5,   //  the canonical name for an alias
    SOA      = 6,   //  marks the start of a zone of authority
    PTR      = 12,  //  domain name pointer
    MX       = 15,  //  mail exchange
    TXT      = 16,  //  text strings
    AXFR     = 252, //  request for a transfer of an entire zone
    ALLTYPES = 255, //  request for all records
} QType;

typedef enum QClass {
    IN  = 1,    // internet
    ALLCLASSES = 255   // any class
} QClass;

typedef struct ResourceRecord {
    char domainName[MAX_NAME_SIZE];
    QType rrType;
    QClass rrClass; // Class always set to 1
    int timeToLive;
    char* rData;
    unsigned short int rdLength;
} ResourceRecord;
```

Intrebarile primite de la utilizatori pot fi de trei tipuri conform protocolului DNS, de accea am ales aceasta mapare pentru diferitele tipuri de interogari posibile:

```
typedef struct QuestionInfo {
    int questionCategory;
    char domainName[MAX_NAME_SIZE];
    enum QType qType;
    enum QClass qClass;
} QuestionInfo;
```

Cum am mentionat in sectiunea ce documenteaza protocolul de transport in cadrul protocolului DNS, mesajele transmise de la un server la alt server sunt limitate la 512 octeti. Acest sitem mapeaza aceste mesaje pentru a facilita rezolvarea mesajelor intr-un mod cat mai lizibil. Un mesaj arbitrar va avea un

header, o lista de intrebari (deobicei numai una), o lista de rapunsuri, o lista de servere autoritare peste informatia primita si o lista de resurse aditionale care pot ajuta un utilizator in cazul in care mesajul primit nu este in totalitate conformat la cerintele acestuia.

Urmatoarea sectiune de code mapeaza componenta header din cadrul mesajului:

```
typedef struct MessageHeader {
    unsigned short id;
    /**
     * This member is responsible for a bunch of message flags
     * QR       Query Response 1 bit: 0 for queries, 1 for responses.
     * OPCODE   Operation Code 4 bits: Typically always 0, see RFC1035 for details.
     * AA       Authoritative Answer 1 bit: Set to 1 if the responding server is authoritative.
     * TC       Truncated Message 1 bit: Set to 1 if the message length exceeds 512 bytes.
     * RD       Recursion Desired 1 bit: Set by the sender of the request if the server should attempt to resolve the query
     * RA       Recursion Available 1 bit: Set by the server to indicate whether or not recursive queries are allowed.
     * Z        Reserved 3 bits: Originally reserved for later use, but now used for DNSSEC queries.
     * RCODE    Response Code 4 bits: Set by the server to indicate the status of the response, i.e. whether or not it was
     */
    unsigned short queryInfo;
    unsigned short qdCount;
    unsigned short anCount;
    unsigned short nsCount;
    unsigned short arCount;
} MessageHeader;
```

In final aceasta este maparea completa a unui mesaj din cadrul unui sitem DNS:

```
typedef struct Message {
    MessageHeader header;
    char* questionsList;
    ResourceRecord* answersList;
    ResourceRecord* authorityList;
    ResourceRecord* additionalsList;
} Message;
```

Folosind aceste mapari vom dezvolta functii de procesare ale acestor informatii care vor fi folosite in dezvoltarea unui resolver si a mai multor name servere care vor permite unui utilizator sa interogheze baza de date si sa primeasca un raspuns potriv nevoilor acestuia.

Resolverul din cadrul sistemului yaDns este defapt un server ce proceseaza recursiv si concurent interogari prin maniera explicata intr-o sectiune de mai sus. Acesta va avea si o componenta de caching care va fi prezenta atat in memorie cat si pe disk. Aceasta componenta va fi folosita inainte de a incepe procesul de a raspunde la o interogare, iar in cazul in care resolverul a mai raspuns o data la aceasta intrebare acesta va lua din cache raspunsul fara a mai porni un proces de rezolvare a unei interogari.

Concurenta din cadrul acestui resolver sa bazeaza pe urmatoarea arhitectura:

```
void executeResolveQuestion(ResolveQuestion* question) {
    question -> resolverFunction(question -> socketDescriptor, question -> recievedQuestion, question -> questionSender);
}

void submitQuestion(ResolveQuestion question) {
    pthread_mutex_lock(&mutexQuestionsQueue);
    questionsQueue[questionsCount++] = question;
    pthread_mutex_unlock(&mutexQuestionsQueue);
    pthread_cond_signal(&condQuestionsQueue);
}

void* startWorker() {
    for (;;) {
        ResolveQuestion question;
        pthread_mutex_lock(&mutexQuestionsQueue);
        while (questionsCount == 0) {
            pthread_cond_wait(&condQuestionsQueue, &mutexQuestionsQueue);
        }
        question = questionsQueue[0];
        for (int i = 0; i < questionsCount - 1; ++i) {
            questionsQueue[i] = questionsQueue[i + 1];
        }
        --questionsCount;
        pthread_mutex_unlock(&mutexQuestionsQueue);
        executeResolveQuestion(&question);
    }
    return NULL;
}
```

Spatiul de domenii din carul sistemului yaDns va fi impartit intr-o structura arborescenta care va fi impartita pe zone. Fiecare zona va fi asignata unui name server care o va stoca atat intr-un master file cat si in memorie. Se doreste ca fiecare zona sa fie gestionata de macar doua name servere ca in cazul in care unul dintre acestea are probleme, gestionarea raspunsurilor sa nu fie oprita. Baza noastra de date va avea un nod radacina care va avea drept nod copii doua top domain levels [.cy] si [.mri]. Fiecare top domain level din aceste doua noduri vor avea la randul lor alte noduri secundare.

5 Concluzii

Urmatoarele imbunatatiri pot fi aduse acestui proiect:

- Implementarea corecta a unui sistem de caching pentru resolver, care proceseaza adaugarea sau scoaterea resurselor intr-un mod concurent prin asignarea unui thread special care gestioneaza aceste actiuni.
- Structurarea bazei de date intr-o structura arborescenta care ierarhizeaza informatiile pentru a fi mai usor de modificat si intretinut
- Gestionarea fenomenului de data racing atunci cand se lucreaza intr-un mediu multithreaded
- Proiectarea unui algoritm rapid de cautare a informatiilor necesare unei interogari

References

1. RFC1034, https://datatracker.ietf.org/doc/html/rfc1034. Last accessed 11 Dec 2023
2. RFC1035, https://datatracker.ietf.org/doc/html/rfc1035. Last accessed 11 Dec 2023