Vysoké učení technické v Brně

Fakulta informačních technologií

Filtrující DNS resolver

Dokumentace k projektu

ISA – Síťové aplikace a správa sítí

Autor: [Tomáš Zavadil]
Login: [xzavadt00]
Datum: 17. listopadu 2025
Akademický rok: 2025/2026

Obsah

1 Úvod

Tento dokument popisuje implementaci filtrujícího DNS resolveru vytvořeného jako projekt do předmětu ISA. Cílem projektu bylo vytvořit funkční DNS resolver, který dokáže filtrovat dotazy typu A směřující na nežádoucí domény a jejich subdomény, přičemž ostatní dotazy přeposílá specifikovanému resolveru.

1.1 Zadání projektu

Napište program `dns`, který bude filtrovat dotazy typu A směřující na domény v rámci dodaného seznamu a jejich poddomény. Ostatní dotazy bude přeposílat v nezměněné podobě specifikovanému resolveru. Odpovědi na dříve přeposlané dotazy bude program předávat původnímu tazateli.

1.2 Rozsah implementace

Implementace zahrnuje:

- Parsování DNS dotazů a sestavování DNS odpovědí
- Filtrování domén na základě seznamu blokovaných domén
- Podporu wildcard záznamů (`*.example.com`)
- Přeposílání dotazů na upstream DNS server
- Podporu IPv4 i IPv6 transportního protokolu (dual-stack)
- Odpovědi s příslušnými chybovými kódy (NXDOMAIN, NOTIMP, SERVFAIL)

2 Teoretický základ

2.1 DNS Protokol

Domain Name System (DNS) je hierarchický a decentralizovaný systém doménových jmen používaný především k překladu doménových jmen na IP adresy [1]. DNS protokol je definován v RFC 1035 a pracuje typicky na UDP portu 53.

2.1.1 Struktura DNS zprávy

DNS zpráva se skládá z následujících částí:

1. **Header** (12 bytů) – obsahuje transaction ID, příznaky a počty sekcí
2. **Question section** – obsahuje dotazy
3. **Answer section** – obsahuje odpovědi na dotazy
4. **Authority section** – obsahuje autoritativní záznamy
5. **Additional section** – obsahuje další informace

2.1.2 Typy DNS záznamů

- **A** – mapuje doménové jméno na IPv4 adresu
- **AAAA** – mapuje doménové jméno na IPv6 adresu
- **MX** – určuje poštovní servery
- **CNAME** – alias pro jinou doménu
- **TXT** – volitelný text pro ověření nebo zabezpečení, např. SPF
- **NS** – určuje autoritativní DNS servery

Pro tento projekt jsou relevantní pouze záznamy typu A

2.2 DNS Proxy

DNS proxy je server, který přijímá DNS dotazy od klientů a přeposílá je upstream DNS serveru. Může provádět různé operace jako cachování, filtrování nebo úpravu dotazů a odpovědí. Tenhle projekt svým zadáním a funkčností odpovídá spíše výrazu DNS proxy než DNS resolver, proto se zde o tomhle termínu zmiňuji a pravděpodobně ho budu často užívat.

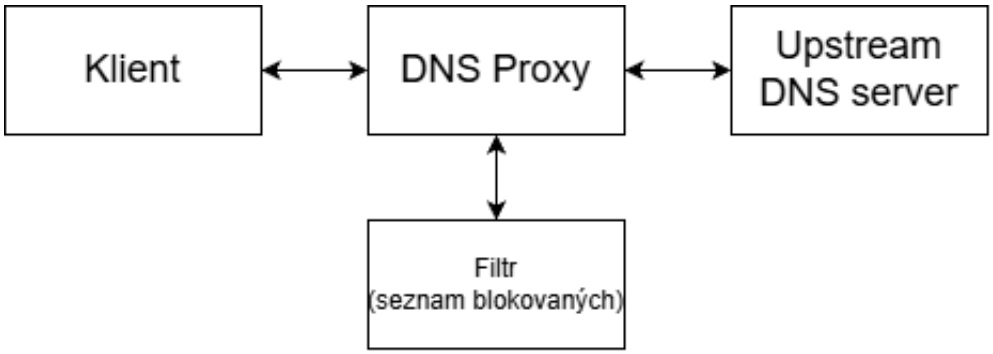
3 Návrh řešení

3.1 Architektura aplikace

Aplikace je rozdělena do následujících modulů:

- **main.c** – hlavní smyčka programu, síťová komunikace
- **args.c/h** – parsování argumentů příkazové řádky
- **filter.c/h** – načítání a kontrola filtrovacích pravidel
- **dns.c/h** – parsování a sestavování DNS zpráv
- **forwarder.c/h** – komunikace s upstream DNS serverem

3.2 Diagram architektury



3.3 Průběh zpracování dotazu

1. Klient pošle DNS dotaz na proxy
2. Proxy přijme dotaz a parsuje ho
3. Kontrola typu dotazu:
   - Pokud není typu A → vrátit NOTIMP
4. Kontrola filtru:
   - Pokud je doména blokována → vrátit NXDOMAIN
5. Přeposlání dotazu na upstream server
6. Přijetí odpovědi od upstream serveru
7. Úprava transaction ID v odpovědi
8. Odeslání odpovědi zpět klientovi

4 Implementace

4.1 Použité technologie a knihovny

Aplikace je implementována v jazyce C s využitím následujících knihoven:

- `<sys/socket.h>` – BSD sockets API
- `<netinet/in.h>` – struktury pro síťovou komunikaci
- `<arpa/inet.h>` – převody síťových adres
- `<netdb.h>` – funkce pro DNS resolver (`getaddrinfo`)
- Standardní C knihovna

4.2 Procházení seznamu blokovaných domén

4.2.1 Datová struktura

Pro uložení seznamu blokovaných domén byl zvolen jednoduchý přístup s polem ukazatelů na řetězce:

```c
#define MAX_FILTER_DOMAINS 1024

typedef struct {
    char *domains[MAX_FILTER_DOMAINS];
    int count;
} FilterList;
```

Listing 1: Struktura FilterList

Tato struktura byla zvolena z následujících důvodů:

- **Jednoduchost implementace** – přímočarý přístup bez složitých datových struktur
- **Dostatečná kapacita** – limit 1024 domén je pro většinu případů použití dostatečný
- **Rychlé načítání** – celý seznam se načte do paměti při startu
- **Přijatelná paměťová náročnost** – při průměrné délce domény 30 znaků jde o cca 30 KB

4.2.2 Algoritmus vyhledávání

Pro kontrolu, zda je doména blokována, je použit **lineární průchod** seznamem s porovnáním každého záznamu:

```
bool filter_is_blocked(const FilterList *list,
                       const char *domain) {
    char tmp[strlen(domain) + 1];
    strcpy(tmp, domain);

    // Normalizace - odstraneni trailing tecky, lowercase
    size_t len = strlen(tmp);
    if (len > 0 && tmp[len - 1] == '.') tmp[len - 1] = '\0';

    strtolower_inplace(tmp);

    for (int i = 0; i < list->count; i++) {
        const char *f = list->domains[i];
        size_t flen = strlen(f);
        if (flen == 0) continue;

        // Wildcard symbol * na zacatku
        if (f[0] == '*') {
            const char *suffix = f + 1;
            size_t slen = flen - 1;
            size_t dlen = strlen(tmp);
            // Domena konci suffixem
            if (dlen >= slen &&
                strcasecmp(tmp + dlen - slen, suffix) == 0) {
                return 1;
        } else {
            // Presna shoda + subdomeny
            size_t dlen = strlen(tmp);

            // Presna shoda
            if (strcasecmp(tmp, f) == 0) {
                return 1;
            }

            // Shoda se subdomenou
            if (dlen > flen + 1) {  // +1 for the dot
                if (tmp[dlen - flen - 1] == '.' &&
                    strcasecmp(tmp + dlen - flen, f) == 0) {
                    return 1;
                }
            }
        }
    }
    return false;
}
```

Listing 2: Kontrola blokované domény

4.2.3 Časová složitost

- **Vyhledávání**: O(n × m), kde:
  - n = počet domén ve filtru
  - m = průměrná délka doménového jména
- **Nejhorší případ**: Pro 1024 domén a průměrnou délku 30 znaků je to přibližně 30 000 operací porovnání znaků

- **Praktický dopad**: Pro běžné použití (stovky domén) je lineární průchod dostatečně rychlý, overhead je zanedbatelný oproti síťové komunikaci

4.2.4 Alternativní přístupy

Byly zvažovány i jiné datové struktury:

Hash tabulka

- **Výhody**: O(1) průměrná složitost vyhledávání
- **Nevýhody**: Složitější implementace, problematická podpora wildcard a subdomain matching
- **Důvod odmítnutí**: Složitost převažuje nad výhodou u malých seznamů

Trie (prefix strom)

- **Výhody**: Efektivní pro prefix matching, vhodné pro DNS hierarchii
- **Nevýhody**: Značná paměťová náročnost, složitá implementace
- **Důvod odmítnutí**: Overhead pro malé až střední seznamy domén

Binární vyhledávací strom

- **Výhody**: O(log n) složitost vyhledávání
- **Nevýhody**: Nutnost seřazení, složitější wildcard matching
- **Důvod odmítnutí**: Lineární průchod je pro typické použití dostatečný

4.2.5 Optimalizace

V implementaci byly použity následující optimalizace:

1. **Case-insensitive porovnání**: Domény jsou převedeny na lowercase již při načítání, aby se konverze nemusela provádět při každém dotazu
2. **Early exit**: Jakmile je nalezena shoda, funkce okamžitě vrací výsledek bez kontroly zbytku seznamu
3. **Jednorázové načtení**: Seznam se načte do paměti při startu a zůstane tam po celou dobu běhu programu

4.2.6 Matching pravidla

Program podporuje dva typy pravidel:

Přesná shoda + subdomény Pro záznam `example.com` v filtru:

`example.com` – přesná shoda `sub.example.com` – subdoména `deep.sub.example.com` – vnořená subdoména `notexample.com` – jiná doména

Wildcard záznamy Pro záznam `*.ads.com` v filtru:

`tracker.ads.com` – wildcard match `analytics.ads.com` – wildcard match `ads.com` – root doména není blokována `myads.com` – jiná doména

4.3 Klíčové funkce

4.3.1 Parsování DNS dotazu

Funkce `dns_parse_question()` parsuje DNS dotaz a extrahuje doménové jméno a typ dotazu:

```
bool dns_parse_question(const uint8_t *buf, int len,
                        DnsQuestion *out) {
    if (len < DNS_HEADER_SIZE + 5)
        return false;

    int pos = DNS_HEADER_SIZE;
    int qname_len = 0;

    // Zpracovani labels
    while (pos < len) {
        uint8_t label_len = buf[pos++];
        if (label_len == 0) break;
        // ... kopirovani label do qname
    }

    // Extrakce QTYPE a QCLASS
    out->qtype  = (buf[pos] << 8) | buf[pos + 1];
    out->qclass = (buf[pos + 2] << 8) | buf[pos + 3];

    return true;
}
```

Listing 3: Parsování DNS dotazu

4.3.2 Komunikace s upstream serverem

Funkce `resolver_forward_query()` přeposílá dotaz na upstream DNS server:

```
bool resolver_forward_query(const char *server,
                            const uint8_t *query, int query_len,
                            uint8_t *response, int *response_len,
                            int timeout_sec) {
    // Resolve server name/IP pomoci getaddrinfo
    struct addrinfo hints, *result;
    memset(&hints, 0, sizeof(hints));
    hints.ai_family = AF_INET;
    hints.ai_socktype = SOCK_DGRAM;
    hints.ai_protocol = IPPROTO_UDP;
    getaddrinfo(server, "53", &hints, &result);
    // Vytvoreni socketu a nastaveni timeoutu
    int sock = socket(result->ai_family,
                      result->ai_socktype,
                      result->ai_protocol);

    // Odeslani dotazu a prijem odpovedi
    sendto(sock, query, query_len, 0,
           result->ai_addr, result->ai_addrlen);
    recvfrom(sock, response, DNS_MAX_PACKET_SIZE, 0,
             NULL, NULL);
    // Uprava TXID
    response[0] = query[0];
    response[1] = query[1];

    return true;
}
```

Listing 4: Přeposlání dotazu

4.4 Dual-stack podpora (IPv4 + IPv6)

Aplikace podporuje příjem dotazů přes IPv4 i IPv6 pomocí dual-stack socketu:

```
// Vytvoreni IPv6 socketu
int sock = socket(AF_INET6, SOCK_DGRAM, 0);

// Vypnuti IPv6-only modu pro dual-stack
int ipv6only = 0;
setsockopt(sock, IPPROTO_IPV6, IPV6_V6ONLY,
           &ipv6only, sizeof(ipv6only));

// Bind na vsechny rozhrani (IPv4 i IPv6)
struct sockaddr_in6 local_addr;
local_addr.sin6_family = AF_INET6;
local_addr.sin6_port = htons(port);
local_addr.sin6_addr = in6addr_any;

bind(sock, (struct sockaddr*)&local_addr,
     sizeof(local_addr));
```

Listing 5: Dual-stack socket

5 Chybové stavy a ošetření chyb

5.1 DNS Chybové kódy (RCODE)

Program generuje následující DNS chybové kódy v závislosti na situaci:

5.1.1 NXDOMAIN (RCODE = 3)

Situace: Dotazovaná doména je v seznamu blokovaných domén.

Popis: Tento chybový kód indikuje, že dotazovaná doména neexistuje. V kontextu DNS proxy je použit pro blokování nežádoucích domén.

Příklad:

```
if (filter_is_blocked(&filters, q.qname)) {
    if (args.verbose)
        fprintf(stderr, "Blocked domain: %s\n", q.qname);

    dns_build_error_response(buf, r, response,
                             &response_len, 3); // NXDOMAIN
    sendto(sock, response, response_len, 0,
           (struct sockaddr*)&client_addr, client_len);
    continue;
}
```

Testování:

```
$ dig @127.0.0.1 -p 5353 blocked.com A

;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 12345
```

5.1.2 NOTIMP (RCODE = 4)

Situace: Dotaz není typu A (např. AAAA, MX, TXT, CNAME).

Popis: Chybový kód "Not Implemented"značí, že server nepodporuje daný typ dotazu. Program podporuje pouze dotazy typu A pro IPv4 adresy.

Příklad:

```
// Kontrola typu dotazu
if (q.qtype != DNS_TYPE_A) {
    if (args.verbose)
        fprintf(stderr, "Non-A query received: %s (type %d)\n",
                q.qname, q.qtype);

    dns_build_error_response(buf, r, response,
                             &response_len, 4); // NOTIMP
    sendto(sock, response, response_len, 0,
           (struct sockaddr*)&client_addr, client_len);
    continue;
}
```

Testování:

```
$ dig @127.0.0.1 -p 5353 google.com AAAA

;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOTIMP, id: 12346
```

5.1.3 SERVFAIL (RCODE = 2)

Situace: Nepodařilo se kontaktovat upstream DNS server nebo došlo k chybě při komunikaci.

Popis: Chybový kód "Server Failure"indikuje, že proxy server nemohl dokončit požadavek kvůli vnitřní chybě.

Možné příčiny:

1. Upstream DNS server není dostupný
2. Síťová chyba při komunikaci s upstream serverem
3. Timeout při čekání na odpověď (5 sekund)
4. Chyba při vytváření socketu
5. Chybná odpověď od upstream serveru (příliš krátká)

Příklad:

```
// Preposlani dotazu na upstream
if (!resolver_forward_query(args.server, buf, r,
                            response, &response_len,
                            DEFAULT_TIMEOUT)) {
    if (args.verbose)
        fprintf(stderr,
                "Failed to query upstream for %s\n",
                q.qname);

    dns_build_error_response(buf, r, response,
                             &response_len, 2); // SERVFAIL
    sendto(sock, response, response_len, 0,
           (struct sockaddr*)&client_addr, client_len);
    continue;
}
```

Testování:

```
# Spusteni s nedostupnym upstream serverem
$ ./dns -s 192.0.2.1 -p 5353 -f filters.txt

# V jinem terminalu
$ dig @127.0.0.1 -p 5353 google.com A

;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: SERVFAIL, id: 12347
```

5.2 Aplikační chybové zprávy

Program vypisuje chybové zprávy na standardní chybový výstup (`stderr`) v následujících situacích:

5.2.1 Chyby při spuštění

Chybějící povinné parametry

```
$ ./dns -p 5353 -f filters.txt
# Vystup: Chyba pri parsovani argumentu
# Exit code: 1
```

Zpráva se zobrazí, pokud:

- Chybí parametr `-s` (upstream server)
- Chybí parametr `-f` (filter soubor)

Neplatný port

```
$ ./dns -s 8.8.8.8 -p 99999 -f filters.txt
# Vystup: Neplatny port: 99999
# Exit code: 1
```

Kontrola rozsahu: port musí být v rozmezí 1-65535.

Nenalezený filter soubor

```
$ ./dns -s 8.8.8.8 -p 5353 -f neexistuje.txt
# Vystup: Chyba: nelze nacist filter file.
# Exit code: 2
```

Chyba při vytváření socketu

```
# Vystup: socket: Permission denied
# Exit code: 3
```

Nastane například při:

- Pokusu o bind na port < 1024 bez root oprávnění
- Systémových omezeních pro vytváření socketů

Chyba při bindování na port

```
# Vystup: bind: Address already in use
# Exit code: 4
```

Nastane, pokud:

- Jiný proces již naslouchá na daném portu
- Aplikace je spuštěna vícekrát současně

5.2.2 Chyby během běhu (verbose mód)

S parametrem -v program vypisuje dodatečné informace:

Malformované DNS dotazy

```c
if (!dns_parse_question(buf, r, &q)) {
    if (args.verbose)
        fprintf(stderr, "Malformed DNS query received\n");
    continue; // dotaz je ignorovan
}
```

Informace o blokování

```c
if (filter_is_blocked(&filters, q.qname)) {
    if (args.verbose)
        fprintf(stderr, "Blocked domain: %s\n", q.qname);
    // ... NXDOMAIN response
}
```

Nepodporované typy dotazů

```c
if (q.qtype != DNS_TYPE_A) {
    if (args.verbose)
        fprintf(stderr, "Non-A query received: %s (type %d)\n",
                q.qname, q.qtype);
    // ... NOTIMP response
}
```

Selhání upstream serveru

```c
if (!resolver_forward_query(...)) {
    if (args.verbose)
        fprintf(stderr,
                "Failed to query upstream resolver for %s\n",
                q.qname);
    // ... SERVFAIL response
}
```

Transaction ID debugging

```c
if (args.verbose) {
    printf("TXID REQ=%02X%02X RESP=%02X%02X\n",
           buf[0], buf[1], response[0], response[1]);
}
```

5.3 Shrnutí exit kódů

Exit kód	Význam
0	Úspěšné ukončení (teoretické, program běží nekonečně)
1	Chyba v argumentech příkazové řádky
2	Nelze načíst filter soubor
3	Chyba při vytváření socketu
4	Chyba při bindování na port

Tabulka 1: Přehled exit kódů programu

5.4 Robustnost implementace

Program je navržen tak, aby:

- Pokračoval v běhu i při příjmu malformovaných paketů
- Logoval chyby pouze v verbose módu, aby nezaplňoval stderr
- Vždy odpověděl klientovi (buď validní odpovědí nebo chybovým kódem)
- Nikdy neselhal při běžných chybových stavech (timeout, nedostupný upstream)
- Ukončil se pouze při fatálních chybách při startu

6 Testování

6.1 Testovací nástroje

Pro testování byla vytvořena automatická testovací sada v Bash skriptu `test_dns.sh`, která obsahuje:

- Test validace argumentů příkazové řádky
- Test různých formátů upstream serveru (IP, doménové jméno)
- Test různých portů
- Test formátů filter souboru
- Test verbose módu
- Test blokování domén
- Test wildcard blokování
- Test case-insensitive matchingu
- Performance test (50 dotazů)
- Test robustnosti (malformed packets)

6.2 Spuštění testů

Listing 6: Spuštění testů

```
# Prelozeni projektu
make clean
make

# Spusteni testovaci sady
make test
```

6.3 Testovací scénáře

6.3.1 Test blokování domény

```
# Filter soubor obsahuje: blocked.com
$ dig @127.0.0.1 -p 5353 blocked.com A

# Ocekavany vysledek: RCODE: NXDOMAIN
```

6.3.2 Test subdomén

```
# Filter soubor obsahuje: blocked.com
$ dig @127.0.0.1 -p 5353 sub.blocked.com A

# Ocekavany vysledek: RCODE: NXDOMAIN
# (subdomeny jsou take blokovany)
```

6.3.3 Test wildcard

```
# Filter soubor obsahuje: *.ads.com
$ dig @127.0.0.1 -p 5353 tracker.ads.com A

# Ocekavany vysledek: RCODE: NXDOMAIN
$ dig @127.0.0.1 -p 5353 ads.com A

# Ocekavany vysledek: RCODE: NOERROR
# (wildcard neblokuje root domenu)
```

6.4 Výsledky testování

Všechny testy prošly úspěšně. Aplikace správně:

- Validuje vstupní parametry
- Podporuje různé formáty upstream serveru
- Blokuje domény a jejich subdomény
- Podporuje wildcard pravidla
- Funguje case-insensitive
- Odpovídá správnými chybovými kódy
- Zvládá malformované pakety bez pádu

7 Použití programu

7.1 Parametry příkazové řádky

```
Použití: dns -s server [-p port] -f filter_file [-v]

Popis parametrů:
  -s server        IP adresa nebo doménové jméno DNS serveru
  -p port          Port pro naslouchání (volitelné) (výchozí 53)
  -f filter_file   Soubor s nežádoucími doménami
  -v               Verbose mód (volitelné)
```

7.2 Příklady spuštění

7.2.1 Základní použití

```
# Spusteni s Google DNS jako upstream
./dns -s 8.8.8.8 -p 5353 -f filters.txt

# Spustsni s domenovym jmenem
./dns -s dns.google -p 5353 -f filters.txt

# Verbose mod pro debugging
./dns -s 8.8.8.8 -p 5353 -f filters.txt -v
```

7.2.2 Testování funkčnosti

```
# V jinem terminalu - test povoleneho dotazu
dig @127.0.0.1 -p 5353 google.com A

# Test blokovaneho dotazu
dig @127.0.0.1 -p 5353 ads.example.com A

# Test nepodporovaneho typu dotazu
dig @127.0.0.1 -p 5353 google.com AAAA
```

7.3 Formát filter souboru

Listing 7: Příklad filters.txt

```
# Komentare zacinaji #
# Prazdne radky jsou ignorovany

# Blokovani cele domeny a jejich subdomen
ads.example.com
tracker.com

# Wildcard blokovani
*.telemetry.com

# Case-insensitive matching
MALWARE.NET
```

8 Závěr

V rámci projektu byl implementován funkční DNS proxy server splňující všechny požadavky zadání. Aplikace dokáže filtrovat DNS dotazy typu A podle seznamu blokovaných domén, podporuje wildcard pravidla a správně přeposílá povolené dotazy na upstream server.

Implementace byla otestována automatickou testovací sadou obsahující více než 70 test cases, které úspěšně prošly. Program je napsán modulárně s jasným rozdělením zodpovědností mezi jednotlivé moduly.

Během vývoje byla věnována pozornost robustnosti (odolnost proti malformovaným paketům) a správné práci se síťovou komunikací včetně podpory IPv6 transportního protokolu.

Reference

[1] P. Mockapetris, *Domain names - implementation and specification*, RFC 1035, November 1987. https://www.rfc-editor.org/rfc/rfc1035

[2] J. Klensin, *Simple Mail Transfer Protocol*, RFC 5321, October 2008. https://www.rfc-editor.org/rfc/rfc5321

[3] IEEE, *The Open Group Base Specifications Issue 7, getaddrinfo*, 2018. https://pubs.opengroup.org/onlinepubs/9699919799/