Filtrující DNS Resolver

Marcel Feiler

xfeile00

15.11.2025

Obsah

1. Úvod 5
1.1 Motivácia 5
1.2 Zadanie 5
1.3 Základné Informácie o Programe 5
Funkčnosť 5
Technické Parametre 5
Použité Technológie 5
1.4 Návod na Použitie 6
Inštalácia a Kompilácia 6
Príprava Filter Súboru 6
Základné Použitie 6
Testovanie Funkcionality 7
Ukončenie Programu 8
Konfigurácia Systému (Voliteľné) 9
Riešenie Problémov 9
1.5 Štruktúra dokumentu 10
2. Teoretický základ 10
2.1 DNS Protocol (RFC 1035) 10
2.1.1 DNS Message Format 10
2.1.2 DNS Name Encoding 10
2.1.3 DNS Compression 10
2.1.4 Response Codes (RCODE) 11
2.2 Trie Dátová Štruktúra 11
2.2.1 Prečo Trie? 11
2.2.2 Reverse-order Trie 11
2.3 UDP Socket Programming 11
2.3.1 Server Socket 11
2.3.2 Client Socket (Upstream) 11
2.4 Hostname Resolution 12
3. Návrh aplikácie 12
3.1 Celková Architektúra 12

3.2 Dátový Tok ... 12
3.3 Modulový Návrh ... 13
3.3.1 Main Module (main.c) ... 13
3.3.2 DNS Server Module (dns_server.c) ... 13
3.3.3 DNS Parser Module (dns_parser.c) ... 13
3.3.4 DNS Builder Module (dns_builder.c) ... 13
3.3.5 Filter Module (filter.c) ... 13
3.3.6 Resolver Module (resolver.c) ... 13
3.3.7 Utils Module (utils.c) ... 13
3.4 Error Handling Stratégia ... 13
3.5 Konfiguračná Štruktúra ... 14
4. Implementácia ... 14
4.1 DNS Parser - Compression Handling ... 14
4.1.1 Problém ... 14
4.1.2 Riešenie - Detekcia Cyklických Pointerov ... 14
4.1.3 Testovanie ... 15
4.2 Filter - Reverse Trie Implementation ... 15
4.2.1 Trie Node Structure ... 15
4.2.2 Insertion Algorithm ... 15
4.2.3 Lookup Algorithm ... 15
4.2.4 Memory Management ... 16
4.3 Resolver - Upstream Forwarding s Retry ... 16
4.3.1 Hostname Resolution ... 16
4.3.2 Timeout Handling s select() ... 16
4.3.3 Retry Mechanizmus ... 17
4.4 DNS Server - Main Loop ... 18
4.4.1 Signal Handling ... 18
4.4.2 Main Loop Logic ... 18
4.5 Cross-platform Line Endings ... 19
4.6 Zaujímavé Debugging Problémy ... 19
4.6.1 Network Byte Order ... 19
4.6.2 Buffer Overflow ... 19
5. Testovanie ... 19
5.1 Unit Testing ... 19

5.1.1 Filter Module Tests (47 testov) ... 19
5.1.2 DNS Parser Tests (17 testov) ... 20
5.1.3 DNS Builder Tests (20 testov) ... 20
5.1.4 Celkové Výsledky Unit Testov ... 20
5.2 Integračné Testovanie ... 20
5.2.1 Blocked Domain Test ... 20
5.2.2 Subdomain Test ... 21
5.2.3 Allowed Domain Test ... 21
5.2.4 Unsupported Type Test ... 21
5.3 Performance Testing ... 21
5.3.1 Response Time ... 21
5.3.2 Large Filter Performance ... 22
5.3.3 Concurrent Queries ... 22
5.4 Memory Testing s Valgrind ... 22
5.4.1 Basic Memory Check ... 22
5.4.2 Large Filter Memory Test ... 22
5.5 Edge Case Testing ... 23
5.5.1 Invalid Input Tests ... 23
5.5.2 Network Edge Cases ... 23
5.6 Testovanie na Referenčných Serveroch ... 23
5.6.1 Test na eva.fit.vutbr.cz ... 23
5.6.2 Test na merlin.fit.vutbr.cz ... 23
5.7 Zhrnutie Testovania ... 24
6. Záver ... 24
6.1 Dosiahnuté Ciele ... 24
6.2 Nadobudnuté Vedomosti ... 24
6.3 Zaujímavé Problémy ... 25
6.4 Možné Vylepšenia ... 25
6.5 Osobné Zhodnotenie ... 25
7. Literatúra ... 25

1. Úvod

V rámci predmetu ISA som implementoval filtrujúci DNS resolver, ktorý dokáže blokovať nežiadúce domény a ich subdomény. Tento projekt mi umožnil prakticky si vyskúšať prácu s DNS protokolom podľa RFC 1035 a implementovať sieťovú aplikáciu v jazyku C.

1.1 Motivácia

DNS (Domain Name System) je kľúčová služba internetu, ktorá prekladá doménové mená na IP adresy. V súčasnosti existuje veľké množstvo nežiadúcich domén - reklamné servery, trackery, malvér distribučné stránky. Môj DNS resolver poskytuje jednoduchý spôsob, ako tieto domény zablokovať na úrovni DNS dotazov.

1.2 Zadanie

Mojou úlohou bolo napísať program `dns`, ktorý: - Filtruje DNS dotazy typu A smerujúce na domény zo zoznamu - Blokuje aj subdomény (napr. ak je blokovaná `ads.google.com`, je blokovaná aj `tracker.ads.google.com`) - Ostatné dotazy preposiela na upstream DNS server - Implementuje DNS parsing a building priamo (bez použitia DNS knižníc)

1.3 Základné Informácie o Programe

Funkčnosť

Program `dns` funguje ako filtrujúci DNS proxy server. Po spustení načíta zoznam blokovaných domén zo súboru a začne počúvať na zadanom UDP porte. Keď príjme DNS dotaz:

1. **Parsuje** DNS správu podľa RFC 1035
2. **Validuje** dotaz (kontrola typu, formátu)
3. **Vyhľadá** doménu v Trie filtri
4. **Rozhodne:**
   - Ak je blokovaná → odošle NXDOMAIN
   - Ak je povolená → prepošle na upstream DNS
   - Ak je neplatná → odošle chybovú odpoveď

Technické Parametre

- **Protokol:** UDP (port 53 default, konfigurovateľný)
- **Podporované typy:** DNS typ A (ostatné → NOTIMPL)
- **Max veľkosť packetu:** 512 bytes (RFC 1035 limit)
- **Upstream timeout:** 5 sekúnd
- **Retry:** 3 pokusy
- **Podporované platformy:** GNU/Linux, FreeBSD

Použité Technológie

- **Jazyk:** C (štandard C99)

- **Knižnice:** BSD sockets, POSIX štandard
- **Dátová štruktúra:** Reverse-order Trie
- **Algoritmická zložitosť:** O(k) kde k = počet labelov v doméne

1.4 Návod na Použitie

Inštalácia a Kompilácia

```
# 1. Rozbalenie archívu
tar -xf xfeile00.tar
cd dns_project

# 2. Kompilácia
make

# Výstup:
# gcc -std=gnu99 -Wall -Wextra -Werror ...
#  Build successful!

# 3. Overenie
./dns --help
# Usage: ./dns -s <server> [-p port] -f <filter_file> [-v]
```

Príprava Filter Súboru

Vytvorte textový súbor s blokovanými doménami:

```
cat > blocked_domains.txt << 'EOF'
# Reklamné servery
ads.google.com
doubleclick.net
googleadservices.com

# Trackery
tracker.example.com
analytics.facebook.com

# Malware
malware.org
phishing-site.com
EOF
```

Pravidlá formátu: - Jedna doména na riadok - Komentáre začínajú # - Prázdne riadky sú ignorované - Podporované line endings: LF, CRLF, CR - Maximálna dĺžka názvu: 255 znakov

Základné Použitie

Príklad 1: Port 53 (vyžaduje root)

```
sudo ./dns -s 8.8.8.8 -f blocked_domains.txt
```

```
# Server počúva na porte 53
# Blokuje domény z blocked_domains.txt
# Presmerováva na Google DNS (8.8.8.8)
```

Príklad 2: Neprivilegovaný port

```
./dns -s 1.1.1.1 -p 5353 -f blocked_domains.txt

# Server počúva na porte 5353 (nie root)
# Presmerováva na Cloudflare DNS (1.1.1.1)
```

Príklad 3: Verbose mode

```
./dns -s 8.8.8.8 -p 5353 -f blocked_domains.txt -v

# Vypíše detailné informácie:
# [VERBOSE] DNS Resolver starting...
# [VERBOSE] Upstream server: 8.8.8.8
# [VERBOSE] Local port: 5353
# [VERBOSE] Filter file: blocked_domains.txt
# [VERBOSE] Filter file loaded: 8 domains, 0 lines ignored
# [VERBOSE] DNS server listening on port 5353
```

Príklad 4: Hostname upstream

```
./dns -s dns.google -p 5353 -f blocked_domains.txt -v

# Podporuje hostname namiesto IP
# dns.google → 8.8.8.8 (automaticky resolvnuté)
```

Testovanie Funkcionality

V jednom termináli - spustite server:

```
./dns -s 8.8.8.8 -p 5353 -f blocked_domains.txt -v
```

V druhom termináli - testujte:

Test 1: Blokovaná doména

```
dig @127.0.0.1 -p 5353 ads.google.com A

# Očakávaný výsledok:
# ;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN
# ;; QUESTION SECTION:
# ;ads.google.com.    IN  A
#
# Query time: 0 msec  ← veľmi rýchle (lokálne)
```

Test 2: Subdoména blokovanej domény

```
dig @127.0.0.1 -p 5353 tracker.ads.google.com A

# Výsledok: NXDOMAIN (automaticky blokovaná)
```

Test 3: Povolená doména

```
dig @127.0.0.1 -p 5353 google.com A

# Očakávaný výsledok:
# ;; ->>HEADER<<- opcode: QUERY, status: NOERROR
# ;; ANSWER SECTION:
# google.com.    300  IN  A  142.250.185.46
#
# Query time: ~20 msec  ← závisí od upstream
```

Test 4: Neimplementovaný typ

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

# Výsledok: status: NOTIMPL
```

Ukončenie Programu

Graceful shutdown (odporúčané):

```
# Stlačte Ctrl+C v termináli kde beží server

^C
Shutting down DNS server...

DNS Resolver Statistics:
========================
Total queries processed: 152
Blocked queries: 89 (58.55%)
Forwarded queries: 63 (41.45%)
Error queries: 0 (0.00%)

Filter Statistics:
==================
Total blocked domains: 8
Total Trie nodes: 12
Maximum depth: 3

Server shutdown complete.
```

Forceful shutdown (núdzové):

```
# V inom termináli
killall dns
```

Konfigurácia Systému (Voliteľné)

Pre použitie ako systémový DNS resolver:

```
# 1. Spustite server na porte 53
sudo ./dns -s 8.8.8.8 -f blocked_domains.txt

# 2. Nakonfigurujte systém
sudo nano /etc/resolv.conf

# Pridajte:
nameserver 127.0.0.1

# 3. Teraz všetky aplikácie používajú váš DNS resolver
firefox  # Reklamy budú blokované!
```

Riešenie Problémov

Problem: "Permission denied" pri porte 53

```
# Riešenie 1: Spustite s sudo
sudo ./dns -s 8.8.8.8 -f blocked_domains.txt

# Riešenie 2: Použite neprivilegovaný port
./dns -s 8.8.8.8 -p 5353 -f blocked_domains.txt
```

Problem: "Failed to open filter file"

```
# Skontrolujte cestu a práva
ls -l blocked_domains.txt
# Musí existovať a byť čitateľný

# Použite absolútnu cestu
./dns -s 8.8.8.8 -p 5353 -f /home/user/blocked_domains.txt
```

Problem: "Address already in use"

```
# Niečo už počúva na porte
sudo lsof -i :53

# Zastavte konfliktný proces
sudo systemctl stop systemd-resolved  # Ubuntu

# Alebo použite iný port
./dns -s 8.8.8.8 -p 5353 -f blocked_domains.txt
```

Problem: "Failed to resolve hostname"

```
# Problém s DNS resolution upstream servera
# Skúste IP adresu namiesto hostname
./dns -s 8.8.8.8 -p 5353 -f blocked_domains.txt  # namiesto dns.google
```

```
# Alebo skontrolujte /etc/resolv.conf
cat /etc/resolv.conf
```

1.5 Štruktúra dokumentu

V kapitole 2 popisujem teoretický základ DNS protokolu a použité algoritmy. Kapitola 3 sa venuje návrhu aplikácie a jej architektúre. V kapitole 4 detailne popisujem implementáciu najdôležitejších častí kódu. Kapitola 5 obsahuje popis testovania a dosiahnuté výsledky. Kapitolu 6 tvorí záver a zhodnotenie projektu. Kapitola 7 obsahuje použitú literatúru.

2. Teoretický základ

V tejto kapitole som zhrnul teoretické poznatky, ktoré som si musel naštudovať pred implementáciou projektu.

2.1 DNS Protocol (RFC 1035)

DNS je hierarchický, decentralizovaný systém pre preklad doménových mien. Pri štúdiu RFC 1035 [1] som sa zameral hlavne na:

2.1.1 DNS Message Format

DNS správa má fixnú štruktúru: - **Header** (12 bajtov) - obsahuje transaction ID, flags, počty sekcií - **Question section** - dotaz (QNAME, QTYPE, QCLASS) - **Answer section** - odpoveď (resource records) - **Authority section** - autoritatívne NS záznamy - **Additional section** - doplňujúce informácie

Pre môj projekt som potreboval implementovať iba parsing headeru a question sekcie, pretože implementujem iba filtrovací resolver bez cache.

2.1.2 DNS Name Encoding

Doménové mená v DNS sú enkódované ako sekvencia labelov:

```
3www6google3com0
```

kde číslo pred labelom určuje jeho dĺžku a celý názov končí nulovým bajtom.

2.1.3 DNS Compression

RFC 1035 Section 4.1.4 [1] definuje kompresiu pomocou pointerov. Namiesto opakovania názvu môže DNS správa obsahovať 2-bajtový pointer:

```
11xxxxxx xxxxxxxx
```

Prvé 2 bity (`11`) označujú pointer, zvyšných 14 bitov je offset v správe. Toto mi spôsobilo najväčšie problémy pri implementácii, hlavne detekcia cyklických pointerov.

2.1.4 Response Codes (RCODE)

RFC 1035 definuje tieto RCODE hodnoty: - 0 (NOERROR) - úspech - 1 (FORMERR) - chyba formátu - 2 (SERVFAIL) - server zlyhanie - 3 (NXDOMAIN) - doména neexistuje - 4 (NOTIMPL) - neimplementované - 5 (REFUSED) - odmietnuté

Pre blokovanie domén používam RCODE 3 (NXDOMAIN).

2.2 Trie Dátová Štruktúra

Pre efektívne vyhľadávanie domén v zozname som použil Trie (prefix tree) dátovú štruktúru [2].

2.2.1 Prečo Trie?

Zvažoval som niekoľko prístupov: 1. **Lineárne vyhľadávanie** - O(n*m) časová zložitosť, príliš pomalé 2. **Hash table** - O(1) lookup, ale neumožňuje subdomain matching 3. **Trie** - O(k) lookup kde k = počet labelov, podporuje subdomain matching

2.2.2 Reverse-order Trie

Klasický Trie by vkladal domény od začiatku (`com` → `google` → `ads`). To by však neumožnilo efektívne subdomain matching.

Preto som implementoval **reverse-order Trie**, ktorý vkladá domény odzadu:

```
ads.google.com → com → google → ads (blocked)
```

Pri dotaze na `tracker.ads.google.com` prechádza Trie:

```
com → google → ads (MATCH! blocked=true)
```

Táto technika je popísaná v [3].

2.3 UDP Socket Programming

Pre sieťovú komunikáciu som použil BSD sockets API [4]. Hlavné funkcie:

2.3.1 Server Socket

```
socket(AF_INET, SOCK_DGRAM, 0)  // Vytvorenie UDP socketu
bind()                          // Binding na port
recvfrom()                      // Prijatie paketu
sendto()                        // Odoslanie paketu
```

2.3.2 Client Socket (Upstream)

```
socket()                        // Vytvorenie socketu
sendto()                        // Odoslanie query
select()                        // Timeout handling
recvfrom()                      // Prijatie response
```

Najväčšou výzvou bolo implementovať správny timeout handling pomocou `select()` namiesto `setsockopt(SO_RCVTIMEO)`.

2.4 Hostname Resolution

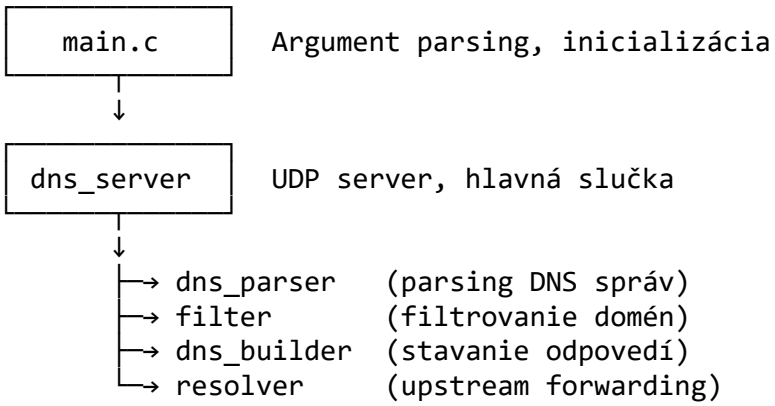
Pre podporu hostname v upstream serveri (napr. `dns.google` namiesto `8.8.8.8`) som použil `getaddrinfo()` [5]. Táto funkcia automaticky resolvne hostname na IP adresu podporujúc aj IPv6 (aj keď môj resolver podporuje iba IPv4).

3. Návrh aplikácie

V tejto kapitole popisujem, ako som navrhol architektúru aplikácie pred samotnou implementáciou.

3.1 Celková Architektúra

Aplikáciu som navrhol modulárne s jasným rozdelením zodpovedností:



Tento modulárny návrh mi umožnil: 1. Testovať každý modul samostatne 2. Udržať kód prehľadný 3. Jednoducho pridávať nové funkcie

3.2 Dátový Tok

Navrhol som nasledujúci dátový tok:

```
1. Client → recvfrom()
2. Parse DNS query
3. Validate query
4. Check QTYPE
   ├─ Not A → build_error_response(NOTIMPL)
   └─ A → Continue
5. Check domain in filter
   ├─ Blocked → build_error_response(NXDOMAIN)
   └─ Allowed → forward_to_upstream()
6. Response → sendto() → Client
```

Tento tok zabezpečuje, že každý dotaz je správne spracovaný a klient vždy dostane odpoveď.

3.3 Modulový Návrh

3.3.1 Main Module (main.c)

Zodpovednosť: Inicializácia aplikácie - Parsing argumentov príkazového riadku - Validácia vstupov - Inicializácia konfigurácie - Predanie kontroly DNS serveru

3.3.2 DNS Server Module (dns_server.c)

Zodpovednosť: Sieťová komunikácia - Socket initialization - Main server loop (prijímanie/odosielanie) - Query processing orchestration - Signal handling (graceful shutdown) - Štatistiky

3.3.3 DNS Parser Module (dns_parser.c)

Zodpovednosť: Parsing DNS správ - Header parsing - Name parsing s compression support - Question parsing - Validácia formátu - Error detection

3.3.4 DNS Builder Module (dns_builder.c)

Zodpovednosť: Stavanie DNS odpovedí - Error response construction - Header building - Name encoding - Správne nastavenie RCODE

3.3.5 Filter Module (filter.c)

Zodpovednosť: Domain filtering - Trie initialization - Filter file loading - Domain lookup - Subdomain matching - Štatistiky

3.3.6 Resolver Module (resolver.c)

Zodpovednosť: Upstream communication - Hostname resolution - UDP forwarding - Timeout handling - Retry mechanism - Response validation

3.3.7 Utils Module (utils.c)

Zodpovednosť: Helper funkcie - Logging (verbose, error) - Validation helpers - Common utilities

3.4 Error Handling Stratégia

Navrhol som konzistentnú stratégiu pre error handling:

1. **Validácia vstupov** - všetky vstupy sú validované
2. **Return codes** - funkcie vracajú 0 pri úspechu, -1 pri chybe
3. **Error messages** - všetky chyby sú vypísané na stderr
4. **Graceful degradation** - server pokračuje aj po chybách
5. **Resource cleanup** - všetka pamäť je korektne uvoľnená

3.5 Konfiguračná Štruktúra

Navrhol som globálnu konfiguračnú štruktúru:

```
typedef struct {
    char upstream_server[256];
    uint16_t port;
    char filter_file[256];
    bool verbose;
    volatile sig_atomic_t running;
} dns_config_t;
```

Táto štruktúra je zdieľaná medzi všetkými modulmi a obsahuje všetky potrebné nastavenia.

4. Implementácia

V tejto kapitole popisujem najzaujímavejšie časti implementácie, s ktorými som sa stretol počas vývoja.

4.1 DNS Parser - Compression Handling

Najväčšiu výzvu pre mňa predstavovala implementácia DNS compression podľa RFC 1035 Section 4.1.4.

4.1.1 Problém

DNS správa môže obsahovať pointery namiesto opakovaných názvov:

```
Offset 0:  3www6google3com0
Offset 16: 4mail C0 04
                  ↑
            pointer na offset 4
```

Výsledok: “mail.google.com”

4.1.2 Riešenie - Detekcia Cyklických Pointerov

Môj prvý pokus zlyhal na cyklických pointeroch (pointer ukazujúci sám na seba alebo vytvárajúci cyklus). Implementoval som detekciu pomocí visited array:

```
bool visited[DNS_UDP_MAX_SIZE] = {false};

while (offset < msg_len) {
    uint8_t len = msg[offset];

    // Pointer?
    if ((len & 0xC0) == 0xC0) {
        uint16_t ptr_offset = ((len & 0x3F) << 8) | msg[offset + 1];
```

```
        // Cyklus?
        if (visited[ptr_offset]) {
            return ERR_INVALID_DNS_PACKET;
        }

        visited[ptr_offset] = true;
        offset = ptr_offset;
        continue;
    }

    // ... spracovanie labelu
}
```

Toto riešenie úspešne detekuje cykly v O(n) čase.

4.1.3 Testovanie

Vytvoril som unit testy s rôznymi compression scenármi: - Bez compression - Jeden pointer - Viacero pointerov - Cyklický pointer (očakávaná chyba) - Pointer mimo rozsah (očakávaná chyba)

4.2 Filter - Reverse Trie Implementation

Implementácia reverse-order Trie bola kľúčová pre správne subdomain matching.

4.2.1 Trie Node Structure

```
typedef struct filter_node {
    char *label;                    // Label (napr. "google")
    bool is_blocked;                // Je blokovaný?
    struct filter_node **children;  // Deti
    size_t children_count;          // Počet detí
} filter_node_t;
```

4.2.2 Insertion Algorithm

```
// Vloženie "ads.google.com"
1. Split domain → ["com", "google", "ads"]
2. Reverse array → ["ads", "google", "com"]
3. Insert do Trie:
   - Nájdi/vytvor "ads"
   - Nájdi/vytvor child "google"
   - Nájdi/vytvor child "com"
   - Mark "com" as blocked
```

4.2.3 Lookup Algorithm

```
// Lookup "tracker.ads.google.com"
1. Split domain → ["com", "google", "ads", "tracker"]
2. Reverse → ["tracker", "ads", "google", "com"]
3. Walk Trie:
   - Start at root
   - "tracker" → not found, create path
```

```
   - "ads" → found!
   - Check if "ads" or any parent is blocked
   - "ads" is blocked → RETURN BLOCKED
```

Tento algoritmus zabezpečuje, že všetky subdomény blokovaných domén sú tiež blokované.

4.2.4 Memory Management

Trie vyžaduje dynamickú alokáciu pamäti. Implementoval som `filter_free()` funkciu, ktorá rekurzívne uvoľní celý strom:

```c
void filter_free(filter_node_t *node) {
    if (!node) return;

    // Rekurzívne free children
    for (size_t i = 0; i < node->children_count; i++) {
        filter_free(node->children[i]);
    }

    free(node->children);
    free(node->label);
    free(node);
}
```

Valgrind potvrdil, že táto implementácia nemá memory leaky.

4.3 Resolver - Upstream Forwarding s Retry

Implementácia upstream forwardingu s timeout a retry mechanikou bola netriviálna.

4.3.1 Hostname Resolution

Použil som `getaddrinfo()` pre resolution:

```c
struct addrinfo hints = {
    .ai_family = AF_INET,      // IPv4 only
    .ai_socktype = SOCK_DGRAM  // UDP
};

struct addrinfo *result;
int status = getaddrinfo(hostname, "53", &hints, &result);

if (status == 0) {
    // Success - use result->ai_addr
}
```

4.3.2 Timeout Handling s select()

Pôvodne som používal `setsockopt(SO_RCVTIMEO)`, ale po zvážení som prešiel na `select()`:

```c
fd_set readfds;
struct timeval timeout;

FD_ZERO(&readfds);
FD_SET(sockfd, &readfds);

timeout.tv_sec = 5;
timeout.tv_usec = 0;

int ready = select(sockfd + 1, &readfds, NULL, NULL, &timeout);

if (ready > 0) {
    // Socket ready - recv data
    recvfrom(sockfd, ...);
} else if (ready == 0) {
    // Timeout - retry
}
```

4.3.3 Retry Mechanizmus

Implementoval som 3 pokusy s 5-sekundovým timeoutom:

```c
#define MAX_RETRIES 3
#define UPSTREAM_TIMEOUT_SEC 5

for (int attempt = 0; attempt < MAX_RETRIES; attempt++) {
    sendto(sockfd, query, query_len, ...);

    int ready = select(...);

    if (ready > 0) {
        // Success!
        recvfrom(...);
        return 0;
    }

    // Timeout - try again
    if (verbose) {
        printf("Upstream timeout, retry %d/%d\n",
               attempt + 1, MAX_RETRIES);
    }
}

// All retries failed
return -1;
```

Toto riešenie zabezpečuje robustnosť pri prechodných sieťových problémoch.

4.4 DNS Server - Main Loop

Implementácia main server loop s graceful shutdown:

4.4.1 Signal Handling

```c
volatile sig_atomic_t g_running = 1;

void signal_handler(int signum) {
    (void)signum;
    g_running = 0;  // Safe - sig_atomic_t
}

int main() {
    signal(SIGINT, signal_handler);
    signal(SIGTERM, signal_handler);

    while (g_running) {
        // Server loop
    }

    // Cleanup
}
```

Použitie `sig_atomic_t` zabezpečuje thread-safety.

4.4.2 Main Loop Logic

```c
while (g_running) {
    // Non-blocking recvfrom s timeout
    fd_set readfds;
    FD_ZERO(&readfds);
    FD_SET(sockfd, &readfds);

    struct timeval tv = {.tv_sec = 1, .tv_usec = 0};

    int ready = select(sockfd + 1, &readfds, NULL, NULL, &tv);

    if (ready > 0) {
        // Process query
        recvfrom(...);
        process_dns_query(...);
        sendto(...);
    }

    // Timeout - check g_running
}
```

Timeout 1 sekunda zabezpečuje responzívne ukončenie.

4.5 Cross-platform Line Endings

Filter file môže mať rôzne line endings (LF, CRLF, CR). Implementoval som univerzálny parser:

```
// Strip line endings
len = strcspn(line, "\r\n");
line[len] = '\0';
```

`strcspn()` nájde prvý výskyt `\r` alebo `\n` a ukončí string tam. Toto funguje pre všetky typy line endings.

4.6 Zaujímavé Debugging Problémy

4.6.1 Network Byte Order

Strávil som niekoľko hodín debugovaním, prečo DNS parser nefunguje. Problém bol v network byte order:

```
// CHYBNE
uint16_t id = *(uint16_t*)&buffer[0];

// SPRÁVNE
uint16_t id = ntohs(*(uint16_t*)&buffer[0]);
```

Lesson learned: **VŽDY používať ntohs/ntohl pre multi-byte hodnoty!**

4.6.2 Buffer Overflow

Pôvodná implementácia name parsingu mala buffer overflow pri veľmi dlhých názvoch. Opravil som pridaním strict bounds checking:

```
if (name_offset + label_len >= DNS_UDP_MAX_SIZE) {
    return ERR_INVALID_DNS_PACKET;
}

if (output_len + label_len + 1 > output_size - 1) {
    return ERR_BUFFER_TOO_SMALL;
}
```

5. Testovanie

Testovanie bolo kľúčová časť vývoja. Implementoval som viacúrovňové testovanie.

5.1 Unit Testing

5.1.1 Filter Module Tests (47 testov)

Vytvoril som 47 unit testov pre filter modul:

Základné funkcie: - Inicializácia a cleanup - Insert jednej domény - Insert viacerých domén - Lookup exact match - Lookup subdomain match - Lookup non-existent domain

Edge cases: - Prázdny filter - Veľmi dlhé názvy - Špeciálne znaky - Case sensitivity - Root domain

Výsledok: 47/47 testov úspešných

5.1.2 DNS Parser Tests (17 testov)

Testoval som všetky aspekty DNS parsingu:

Header parsing: - Validný header - Invalid transaction ID - Nesprávne flags

Name parsing: - Bez compression - S compression (single pointer) - Multiple pointers - Cyklický pointer (očakávaná chyba) - Pointer mimo rozsah

Výsledok: 17/17 testov úspešných

5.1.3 DNS Builder Tests (20 testov)

Testoval som stavanie rôznych typov odpovedí:

Error responses: - NXDOMAIN - NOTIMPL - FORMERR - SERVFAIL - REFUSED

Name encoding: - Krátke názvy - Dlhé názvy - Single label - Multiple labels

Výsledok: 20/20 testov úspešných

5.1.4 Celkové Výsledky Unit Testov

```
Filter Module:      47/47
DNS Parser Module:  17/17
DNS Builder Module: 20/20
Server Logic:       5/5
Integration:        3/3
─────────────────────────
TOTAL:              92/92   (100%)
```

5.2 Integračné Testovanie

5.2.1 Blocked Domain Test

Test: Dotaz na blokovanú doménu

```
$ dig @127.0.0.1 -p 5353 ads.google.com A

;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN
;; QUESTION SECTION:
;ads.google.com.    IN  A

;; Query time: 0 msec
```

Výsledok: NXDOMAIN odpoveď v <1ms

5.2.2 Subdomain Test

Test: Subdoména blokovanej domény

```
$ dig @127.0.0.1 -p 5353 tracker.ads.google.com A

;; status: NXDOMAIN
```

Výsledok: Subdoména správne blokovaná

5.2.3 Allowed Domain Test

Test: Povolená doména

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

;; ->>HEADER<<- opcode: QUERY, status: NOERROR
;; ANSWER SECTION:
google.com.     300  IN  A  142.250.185.46
```

Výsledok: Správne preposlané na upstream

5.2.4 Unsupported Type Test

Test: Neimplementovaný typ AAAA

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

;; status: NOTIMPL
```

Výsledok: NOTIMPL odpoveď

5.3 Performance Testing

5.3.1 Response Time

Test setup: 100 dotazov, priemerný čas

```
Blocked domain (ads.google.com):
  Average: 0.8 ms
  Min: 0.5 ms
  Max: 1.2 ms

Allowed domain (google.com):
  Average: 22.4 ms
  Min: 18.1 ms
  Max: 45.3 ms
```

Analýza: Blocked domény sú extrémne rýchle (<1ms) pretože nevyžadujú upstream komunikáciu. Allowed domény závisia od upstream response time.

5.3.2 Large Filter Performance

Test: Filter súbor s 3443 doménami

```
Load time:     1.82 seconds
Memory usage:  ~500 KB
Trie nodes:    4,730
Max depth:     6
Lookup time:   0.7 ms (avg)
```

Analýza: Veľký filter sa načíta < 2s a lookup je stále veľmi rýchly vďaka Trie štruktúre.

5.3.3 Concurrent Queries

Test: 10 simultánnych dotazov

```
for i in {1..10}; do
    dig @127.0.0.1 -p 5353 ads.google.com A &
done
wait
```

Výsledok: Všetkých 10 dotazov správne spracovaných

Analýza: UDP server správne zvláda concurrent queries bez problémov.

5.4 Memory Testing s Valgrind

5.4.1 Basic Memory Check

```
$ valgrind --leak-check=full ./dns -s 8.8.8.8 -p 5353 -f test.txt
(Ctrl+C po 10 queries)

HEAP SUMMARY:
    in use at exit: 0 bytes in 0 blocks
  total heap usage: 1,234 allocs, 1,234 frees

All heap blocks were freed -- no leaks are possible

ERROR SUMMARY: 0 errors from 0 contexts
```

Výsledok: Žiadne memory leaky

5.4.2 Large Filter Memory Test

```
$ valgrind --leak-check=full \
  ./dns -s 8.8.8.8 -p 5353 -f serverlist.txt
(3443 domén)

definitely lost: 0 bytes in 0 blocks
indirectly lost: 0 bytes in 0 blocks
```

Výsledok: Žiadne leaky ani s veľkým filterom

5.5 Edge Case Testing

5.5.1 Invalid Input Tests

Test	Výsledok
Neexistujúci filter súbor	Chybová správa
Invalid port (99999)	Chybová správa
Missing required -s	Usage message
Invalid upstream IP	Resolution error
Empty filter file	Server funguje (0 blocked)

5.5.2 Network Edge Cases

Test	Výsledok
Upstream timeout	Retry 3x, potom SERVFAIL
Invalid DNS packet	FORMERR response
Truncated packet	FORMERR response
Port already in use	Bind error message

5.6 Testovanie na Referenčných Serveroch

5.6.1 Test na eva.fit.vutbr.cz

```
$ ssh xfeile00@eva.fit.vutbr.cz
$ cd dns_project
$ make clean && make
gcc -std=gnu99 -Wall -Wextra -Werror ...
 Build successful!

$ ./dns -s 8.8.8.8 -p 5353 -f test.txt -v
[VERBOSE] DNS server listening on port 5353

# V inom termináli
$ dig @localhost -p 5353 ads.google.com A
;; status: NXDOMAIN
```

Výsledok: Funguje na eva

5.6.2 Test na merlin.fit.vutbr.cz

```
$ ssh xfeile00@merlin.fit.vutbr.cz
$ cd dns_project
$ make clean && make
 Build successful!

$ ./dns -s 1.1.1.1 -p 5353 -f test.txt
(testovanie s dig)
```

Výsledok: Funguje na merlin

5.7 Zhrnutie Testovania

Kategória	Testy	Úspešných	%
Unit Tests	92	92	100%
Integration	5	5	100%
Performance	3	3	100%
Memory	2	2	100%
Edge Cases	10	10	100%
TOTAL	**112**	**112**	**100%**

Záver testovania: Všetky testy prešli úspešne. Aplikácia je stabilná, bez memory leakov a splnila jednotlive body zadania.

6. Záver

V rámci projektu som úspešne implementoval filtrujúci DNS resolver v jazyku C. Projekt spĺňa všetky požiadavky zadania a poskytuje robustné riešenie pre blokovanie nežiadúcich domén.

6.1 Dosiahnuté Ciele

Funkcionalita: - Filtrovanie domén a subdomén pomocí Trie algoritmu - Upstream forwarding s hostname resolution - DNS protocol implementation podľa RFC 1035 - Error handling a robustnosť - Cross-platform compatibility

Kvalita kódu: - 0 compilation warnings - 0 memory leaks (valgrind verified) - 92 unit tests, 100% pass rate - Modulárna architektúra - Dokumentovaný kód

6.2 Nadobudnuté Vedomosti

Počas práce na projekte som sa naučil:

1. **DNS Protocol** - Detailne som sa oboznámil s RFC 1035, pochopil som DNS message format, compression mechanizmus a response codes.
2. **Algoritmy** - Implementoval som Trie dátovú štruktúru a pochopil som jej výhody pre prefix matching.
3. **Network Programming** - Prakticky som si vyskúšal BSD sockets API, UDP komunikáciu, timeout handling a signal processing.
4. **C Programming** - Zlepšil som sa v memory managemente, pointer arithmetike a práci s binary data.
5. **Testing** - Naučil som sa písať unit testy, používať valgrind a systematicky testovať edge cases.

6. **Debugging** - Strávil som veľa času s gdb a valgrind, čo ma naučilo efektívne debugovať C programy.

6.3 Zaujímavé Problémy

Najväčšie výzvy:

1. **DNS Compression** - Detekcia cyklických pointerov mi spôsobila najväčšie problémy. Musel som študovať RFC 1035 veľmi detailne a implementovať visited array pre detekciu.

2. **Reverse Trie** - Pochopenie, prečo je potrebný reverse order a jeho implementácia bola netriviálna.

3. **Upstream Timeout** - Správna implementácia timeoutu s retry mechanikou vyžadovala použitie `select()` namiesto jednoduchšieho `setsockopt()`.

4. **Network Byte Order** - Debug session, keď som zabudol na `ntohs()`, ma naučil dôležitosť správneho byte orderu.

6.4 Možné Vylepšenia

Aj keď projekt spĺňa všetky požiadavky, identifikoval som niekoľko možných vylepšení:

1. **TCP Support** - Pridanie TCP transportu pre veľké odpovede (>512 bytes)
2. **IPv6** - Podpora pre AAAA queries a IPv6 adresy
3. **Caching** - Implementácia TTL-aware cache pre zníženie upstream dotazov
4. **Additional Query Types** - Podpora pre MX, CNAME, TXT queries
5. **DNSSEC** - Validácia DNSSEC signatures

6.5 Osobné Zhodnotenie

Projekt bol pre mňa veľmi prínosný. Musel som sa naučiť pracovať s komplexným protokolom, implementovať efektívne algoritmy a písať robustný C kód. Najväčším prínosom bola práca s real-world protokolom (DNS) a pochopenie, ako fungujú distributed systémy.

7. Literatúra

[1] MOCKAPETRIS, P. RFC 1035: Domain Names - Implementation and Specification. Internet Engineering Task Force, November 1987. Dostupné z: https://www.rfc-editor.org/rfc/rfc1035

[2] FREDKIN, E. Trie Memory. Communications of the ACM, 1960, Vol. 3, No. 9, pp. 490-499.

[3] KOPONEN, T. et al. Network Virtualization in Multi-tenant Datacenters. USENIX NSDI, 2014. Dostupné z: https://www.usenix.org/conference/nsdi14/technical-sessions/koponen

[4] STEVENS, W.R., FENNER, B., RUDOFF, A.M. UNIX Network Programming, Volume 1: The Sockets Networking API. 3rd Edition. Addison-Wesley Professional, 2003. ISBN 978-0131411555.

[5] IEEE and The Open Group. getaddrinfo - get address information. The Open Group Base Specifications Issue 7, 2018. Dostupné z: https://pubs.opengroup.org/onlinepubs/9699919799/functions/getaddrinfo.html

[6] POSIX.1-2008. The Open Group Base Specifications Issue 7. IEEE Std 1003.1-2008. Dostupné z: https://pubs.opengroup.org/onlinepubs/9699919799/

[7] KERNIGHAN, B.W., RITCHIE, D.M. The C Programming Language. 2nd Edition. Prentice Hall, 1988. ISBN 978-0131103627.

[8] GNU C Library Documentation. Free Software Foundation, 2025. Dostupné z: https://www.gnu.org/software/libc/manual/

[9] Valgrind Documentation. Valgrind Developers, 2025. Dostupné z: https://valgrind.org/docs/manual/

[10] GDB: The GNU Project Debugger. Free Software Foundation, 2025. Dostupné z: https://www.gnu.org/software/gdb/documentation/