

# Monitorování DNS komunikace

Matúš Janek

November 18, 2024

## Obsah

1. Úvod
2. Teória
3. Vstupné argumenty
4. Implementácia
  - 4.1 Štruktúra repozitára
  - 4.2 Spracovanie vstupných argumentov
  - 4.3 Vytvorenie a spustenie filtru
  - 4.4 Výpis zachytených pakiet
5. Ilustrovaná funkcionality
6. UML
7. Testovanie
8. Bibliografia

## 1. Úvod

Táto dokumentácia slúži ako podrobný manuál k projektu **Monitorování DNS komunikace**, ktorý sa zameriava na implementáciu monitorovania DNS komunikácie. Projekt umožňuje zachytávanie DNS sieťových paketov a ich následujúce spracovanie a písanie na výstup.

Dokumentácia obsahuje technické detaily implementácie, spôsoby použitia aplikácie a jej funkcionality, ako aj postupy testovania a validácie implementovaných funkcií. Okrem toho sa tu nachádzajú aj informácie o doplnkových funkciách a prípadne zdroje, ktoré boli využité pri vytváraní projektu. Informácie o zadaní projektu viz. [1].

## 2. Teória

DNS je jedným z základných kameňov internetu, ktorý umožňuje prevod doménových mien na IP adresy a tým aj efektívnu komunikáciu medzi zariadeniami v sieti. DNS komunikácia prebieha najčastejšie cez protokol UDP, pričom samotné DNS správy obsahujú informácie o dotazoch (query) a odpovediach (response) na zadané doménové mená. DNS správy sa skladajú z niekoľkých sekcií, ktoré obsahujú rôzne informácie potrebné na spracovanie dotazov a odpovedí, tieto sekcie sú Header, Question, Authority a Additional. Pre viac informácií o týchto sekciách viz. [3].

## 3. Vstupné argumenty

Program je spúšťaný z príkazového riadka s nasledujúcimi parametrami:

```
./dns-monitor (-i <interface> | -p <pcapfile>) [-v] [-d <domainsfile>]  
[-t <translationsfile>]
```

Význam jednotlivých vstupných argumentov je špecifikovaný v zadaní viz. [1].

Tento program taktiež podporuje **-help** ktorý vypíše nápovedu, tento argument nesmie byť zadaný s akýmkoľvek iným vstupným argumentom.

## 4. Implementácia

### 4.1 Štruktúra repozitára

Tento program obsahuje súbory **argument\_parser.cpp/hpp** v ktorej je implementovaná trieda **Parser** ktorá obsahuje atribúty zodpovedajúce jednotlivých vstupných argumentom a metódy a ich spracovanie. **packet\_capturing.cpp/hpp** obsahuje triedu **Sniffer** ktorá obsahuje metódy na vytvorenie filtra. Súbory **packet\_processing.cpp/hpp** obsahuje triedu **PacketProcessing** v ktorej sa nachádzajú statické metódy na spracovanie pakety. **utils.cpp/hpp** obsahuje pomocné funkcie pre výpis pakiet a niekoľko

ďalších funkcionalít. Súbor `terminators.cpp/hpp` implementuje triedu `terminators` ktorá zachytáva signále prerušenie a následne manuálne dealokuje všetky potrebné zdroje [4].

## 4.2 Spracovanie vstupných argumentov

Vo funkcii `main` sa vytvorí inštancia triedy `Parser` ktorá obsahuje metódu pre spracovanie vstupných argumentov. Následne sa zavolá jej metóda `parser.parse(argc, argv)` ktorá berie ako parametre pole argumentov a ich počet. Táto metóda spracuje argumenty. Vstupné argumenty sa následne ukladajú do atribútov inštancnej metódy `parser`.

## 4.3 Vytvorenie a spustenie filtru

Po úspešnom spracovaní vstupných argumentov sa vytvorí inštancia triedy `Sniffer` a následne sa zavolá metóda `void run_sniffer(parser &parser)` alebo `void run_pcap(parser &parser)` (záleží či spracovávame pakety zo vstupného súboru alebo rozhrania), ktorá berie ako parameter inštancnú triedu `parser`. AK pakety zaznamenávame z rozhrania tak sa inicializuje sniffer pomocou metódy `pcap_t* init_sniffer(parser& parser)` ktorá zahŕňa otvorenie sieťového rozhrania pomocou funkcie `pcap_open_live`. Po inicializácii sniffera sa volá metóda `void build_filter(parser& parser, pcap_t* handle)`, ktorá slúži na vytvorenie a nastavenie filtru pre zachytávanie DNS paketov. Pomocou `pcap_compile` a `pcap_setfilter` sa aplikuje filter na `handle`. Po úspešnej inicializácii a nastavení filtra sa spúšťa zachytávanie sieťových paketov volaním metódy `void capture_packets(parser &parser, pcap_t *handle)`. Táto metóda používa funkciu `pcap_loop`, ktorá kontinuálne zachytáva pakety. Pre viac popísaný význam jednotlivých funkcií filtru viz. [2].

## 4.4 Výpis zachytených pakiet

Funkcia `PacketProcessing::parse_packet` je volaná v cykle, ktorý kontinuálne zachytáva pakety na základe nastavených parametrov sniffera. Najprv sa z hlavičky paketu získa časová značka, ktorá označuje čas, keď bol paket zachytený. Časová značka je formátovaná vo funkcii `void print_timestamp(const struct pcap_pkthdr *header, parser *parse)`, ktorá konvertuje čas z formátu štruktúry `pcap_pkthdr` do reťazca čitateľného pre človeka.

**Získanie IP adresy** Počas spracovania paketu funkcia `void print_ip(const u_char *frame, parser *parse)` extrahuje zdrojovú a cieľovú IP adresu. V závislosti od verzie IP protokolu sa vyberie zodpovedajúca hlavička a IP adresy sú premenené na reťazec pomocou funkcie `inet_ntop`. Pre IPv4 sa používa štruktúra `ip`, zatiaľ čo pre IPv6 sa používa štruktúra `ip6_hdr`. Výsledné IP adresy sú následne vypísané, buď podrobne (`verbose` mód), alebo v skrátenom formáte.

**Spracovanie portov** Funkcie `process_ipv4_port(const u_char *frame)` a `process_ipv6_port(const u_char *frame)` spracovávajú informácie o zdrojovom a cieľovom porte pre UDP pakety. Na základe verzie IP protokolu vypisujú tieto informácie, ak je zapnutý **verbose** mód. Tieto funkcie vypisujú zdrojový a cieľový port pre UDP. [5]

**Identifikátor a príznaky DNS** Funkcia `std::pair<const u_char*, uint8_t> print_identfier_and_flags(const u_char *frame, u_int16_t type, parser *parse)` extrahuje DNS hlavičku z UDP rámca a následne vypíše identifikátor DNS transakcie a jednotlivé príznaky. Ak je povolený **verbose** mód, vypisujú sa detaily ako QR, počet sekcií, a podobne. viz [3] a [5].

**DNS informácie** Funkcia `void print_dns_information(const u_char *frame, const u_char *pointer, parser *parse, uint8_t qr)` V tejto funkcii sa kontroluje či bol zadaný paramter **verbous**, ak áno tak sa vypíšu informácie v špecifikovanom formáte viz [2], následne sa volajú pomocné funkcie na zpracovanie DNS sections pomocou funkcií `print_question_sections` a `print_other_sections`. Každá paketa je oddelená formátovacou čiarou (=====) pre lepšiu čitateľnosť výpisu, ak je zapnutý **verbose** mód.

**Spracovanie DNS otázok** Funkcia `print_question_sections` spracováva DNS otázky, kde vypíše doménové meno, typ záznamu a triedu záznamu, ak je povolený **verbose** mód. Funkcia taktiež pridáva doménové mená do súboru, ak je to nastavené pomocou parametra `domains_file`, pre parsovanie mena sa volá funkcia `parse_data`. V prípade zadania parametru `-d` sú doménové mená vypísané do výstupného súboru ktorého názov je špecifikovaný pri spustení. V prípade zadania `-t` parametru pri spustení sú doménové mená a ich preklady vypisované do výstupného súboru ktorý obsahuje preklady doménových mien na ip adresy. [7]

**Spracovanie ďalších sekcií** Funkcia `print_other_sections` spracováva záznamy v sekciách `Answer`, `Authority`, a `Additional`. Pre každý záznam vypíše meno, typ záznamu, triedu, TTL a ďalšie detaily. Pomocou funkcie `parse_rdata_and_print` sa následne spracuje obsah záznamov, čo zahŕňa rôzne typy DNS záznamov, ako napríklad `A`, `AAAA`, `NS`, `MX` a iné. [5] V prípade zadania parametru `-d` sú doménové mená vypísané do výstupného súboru ktorého názov je špecifikovaný pri spustení. V prípade zadania `-t` parametru pri spustení sú doménové mená a ich preklady vypisované do výstupného súboru ktorý obsahuje preklady doménových mien na ip adresy.

*Funkcie su bližšie definované v hlavičkových súboroch.*

## 5. Ilustrovaná funkcionalita

### Vytvorenie filteru

```
void Sniffer::build_filter(parser &parser, pcap_t *handle)
{
    // Filter expression for DNS over UDP (port 53)
    std::string filter = "udp port 53";

    bpf_u_int32 net;
    bpf_u_int32 mask;
    struct bpf_program bpf_prog;

    if(!parser.interface.empty())
    {
        // Lookup network details (netmask, IP range, etc.) for the given interface
        if (pcap_lookupnet(parser.interface.c_str(), &net, &mask, errbuf) == PCAP_ERROR)
        {
            std::cerr << "Error: Looking up network: " << errbuf << std::endl;
            exit(EXIT_FAILURE);
        }
    }

    // Compile the filter expression
    if (pcap_compile(handle, &bpf_prog, filter.c_str(), 0, mask) == PCAP_ERROR)
    {
        std::cerr << "Error: Filter compiling: " << pcap_geterr(handle) << std::endl;
        exit(EXIT_FAILURE);
    }

    // Set the compiled filter
    if (pcap_setfilter(handle, &bpf_prog) == PCAP_ERROR)
    {
        std::cerr << "Error: Setting filter: " << pcap_geterr(handle) << std::endl;
        pcap_freecode(&bpf_prog); // Free the filter code if an error occurs
        exit(EXIT_FAILURE);
    }

    // Free the compiled filter after it's set
    pcap_freecode(&bpf_prog);
}
```

### Parsovanie dát ako napríklad domain name

```
std::pair<std::string, int> Utils::parse_data(const u_char *beginning_of_section,
const u_char *packet_start)
{
    std::string data;
    const u_char *current_ptr = beginning_of_section;
    int lenght = 0;
    int offset = 0;
    // Get lenght of data that is goin to be parsed
    lenght = get_domain_name_length(current_ptr);

    // Loop till 0 value is occured
    while (*current_ptr != 0)
    {
        // Found reference
        if ((*current_ptr & 0xC0) == 0xC0)
        {
            // Get the next byte for the offset
            offset = ((*current_ptr & 0x3F) << 8);
            current_ptr += 1;
            offset |= *current_ptr;
            current_ptr = packet_start + offset;
        }
        else // Append the bytes into domain_name
        {
            int label_length = *current_ptr;
            current_ptr++;
            data.append((const char *)current_ptr, label_length);
            current_ptr += label_length;
            if (*current_ptr != 0)
            {
                data.append(".");
            }
        }
    }

    // Add the final dot if it's not there yet
    if (data.back() != '.')
    {
        data.append(".");
    }
    return std::make_pair(data, lenght);
}
```

## 6. UML



## 7. Testovanie

Testovacie prostredie: WSL.

Program bol úspešne testovaný pomocou vstupných .pcap súborov a ich následovné porovnávanie v aplikácii wireshark.

### Record type A (Question section)

2024-09-20 12:03:45 SrcIP: 2001:930:107:86a3:7c20:b664:8a1b:f323 -> DstIP: 2001:4860:4860::8888 (Q 1/0/0/0)

verbose

Timestamp: 2024-09-20 12:03:45

SrcIP: 2001:930:107:86a3:7c20:b664:8a1b:f323

DstIP: 2001:4860:4860::8888

SrcPort: UDP/52561

DstPort: UDP/53

Identifier: 0xbe0b

Flags: QR=0, OPCODE=0, AA=0, TC=0, RD=1, RA=0, AD=0, CD=0, RCODE=0

[Question Section]



instagram.fist2-4.fna.fbcdn.net. IN A

=====

```

Domain Name System (query)
  Transaction ID: 0x4be0b
  > Flags: 0x0100 Standard query
  Questions: 1
  Answer RRs: 0
  Authority RRs: 0
  Additional RRs: 0
  > Queries
    > instagram.fist2-4.fna.fbcdn.net: type A, class IN

```

## AAAA (Question and Answer section)

2024-09-20 13:34:53 SrcIP: 2001:4860:4860::8888 -> DstIP: 2001:930:107:86a3:7c20:b664:8a1b:f323  
(R 1/1/0/0)

### verbous

Timestamp: 2024-09-20 13:34:53  
SrcIP: 2001:4860:4860::8888  
DstIP: 2001:930:107:86a3:7c20:b664:8a1b:f323  
SrcPort: UDP/53  
DstPort: UDP/54218  
Identifier: 0x4be4  
Flags: QR=1, OPCODE=0, AA=0, TC=0, RD=1, RA=1, AD=0, CD=0, RCODE=0

[Question Section]

accounts.google.com. IN AAAA

[Answer Section]

accounts.google.com. 20 IN AAAA 2a00:1450:4013:c14::54

=====

```

> Frame 1: 127 bytes on wire (1016 bits), 127 bytes captured (1016 bits)
> Ethernet II, Src: ZyxelCommuni_5b:76:98 (48:ed:e6:5b:76:98), Dst: Intel_38:82:81 (c0:a5:e8:38:82:81)
> Internet Protocol Version 6, Src: 2001:4860:4860::8888, Dst: 2001:930:107:86a3:7c20:b664:8a1b:f323
> User Datagram Protocol, Src Port: 53, Dst Port: 54218
  > Domain Name System (response)
    Transaction ID: 0x4be4
    > Flags: 0x8100 Standard query response, No error
    Questions: 1
    Answer RRs: 1
    Authority RRs: 0
    Additional RRs: 0
    > Queries
      > accounts.google.com: type AAAA, class IN
    > Answers
      > accounts.google.com: type AAAA, class IN, addr 2a00:1450:4013:c14::54

```

## SOA (Authority section) A (Question section)

2024-09-20 12:04:33 SrcIP: 2001:4860:4860::8888 -> DstIP: 2001:930:107:86a3:7c20:b664:8a1b:f323  
(R 1/0/1/0)

### verbous

Timestamp: 2024-09-20 12:04:33  
SrcIP: 2001:4860:4860::8888

DstIP: 2001:930:107:86a3:7c20:b664:8a1b:f323  
SrcPort: UDP/53  
DstPort: UDP/53250  
Identifier: 0x4256  
Flags: QR=1, OPCODE=0, AA=0, TC=0, RD=1, RA=1, AD=0, CD=0, RCODE=3

[Question Section]

wpad.hgw.local. IN A

[Authority Section]

. 86399 IN SOA a.root-servers.net. nstld.verisign-grs.com. 2024092000 1800 900 604800 86400  
=====

```

  ▾ Queries
    > wpad.hgw.local: type A, class IN
  ▾ Authoritative nameservers
    ▾ <Root>: type SOA, class IN, mname a.root-servers.net
      Name: <Root>
      Type: SOA (6) (Start Of a zone of Authority)
      Class: IN (0x0001)
      Time to live: 86399 (23 hours, 59 minutes, 59 seconds)
      Data length: 64
      Primary name server: a.root-servers.net
      Responsible authority's mailbox: nstld.verisign-grs.com
      Serial Number: 2024092000
      Refresh Interval: 1800 (30 minutes)
      Retry Interval: 900 (15 minutes)
      Expire limit: 604800 (7 days)
      Minimum TTL: 86400 (1 day)
```

NS (Authority section), A, AAAA (Additional, Answer, Question section)

*2024-09-25 11:55:31 SrcIP: 147.229.191.143 -> DstIP: 147.229.193.87 (R 1/1/2/4)*

verbose

Timestamp: 2024-09-25 11:55:31  
SrcIP: 147.229.191.143  
DstIP: 147.229.193.87  
SrcPort: UDP/53  
DstPort: UDP/53760  
Identifier: 0xb4e1  
Flags: QR=1, OPCODE=0, AA=0, TC=0, RD=1, RA=1, AD=0, CD=0, RCODE=0

[Question Section]

www.vut.cz. IN A

[Answer Section]

www.vut.cz. 68 IN A 147.229.2.90

[Authority Section]

vut.cz. 68 IN NS rhino.cis.vutbr.cz.  
vut.cz. 68 IN NS pipit.cis.vutbr.cz.

```
[Additional Section]
pipit.cis.vutbr.cz. 283 IN A 77.93.219.110
pipit.cis.vutbr.cz. 283 IN AAAA 2a01:430:120::4d5d:db6e
rhino.cis.vutbr.cz. 68 IN A 147.229.3.10
rhino.cis.vutbr.cz. 68 IN AAAA 2001:67c:1220:e000::93e5:30a
=====
```

```

  ▾ Queries
    > www.vut.cz: type A, class IN
  ▾ Answers
    > www.vut.cz: type A, class IN, addr 147.229.2.90
  ▾ Authoritative nameservers
    > vut.cz: type NS, class IN, ns rhino.cis.vutbr.cz
    > vut.cz: type NS, class IN, ns pipit.cis.vutbr.cz
  ▾ Additional records
    > pipit.cis.vutbr.cz: type A, class IN, addr 77.93.219.110
    > pipit.cis.vutbr.cz: type AAAA, class IN, addr 2a01:430:120::4d5d:db6e
    > rhino.cis.vutbr.cz: type A, class IN, addr 147.229.3.10
    > rhino.cis.vutbr.cz: type AAAA, class IN, addr 2001:67c:1220:e000::93e5:30a

```

**MX (Answer and Question section), A, AAAA (Additional section),  
NS (Authority section)**

*2024-09-25 12:39:10 SrcIP: 147.229.191.143 -> DstIP: 147.229.193.87 (R  
1/1/2/5)*

#### verbous

```

Timestamp: 2024-09-25 12:39:10
SrcIP: 147.229.191.143
DstIP: 147.229.193.87
SrcPort: UDP/53
DstPort: UDP/50249
Identifier: 0x1971
Flags: QR=1, OPCODE=0, AA=0, TC=0, RD=1, RA=1, AD=0, CD=0, RCODE=0

```

```
[Question Section]
example.com IN MX
```

```
[Answer Section]
example.com 86400 IN MX 0 <root>
```

```
[Authority Section]
example.com 72806 IN NS b.iana-servers.net
example.com 72806 IN NS a.iana-servers.net
```

```
[Additional Section]
a.iana-servers.net 60 IN A 199.43.135.53
a.iana-servers.net 60 IN AAAA 2001:500:8f::53
b.iana-servers.net 60 IN A 199.43.133.53
b.iana-servers.net 60 IN AAAA 2001:500:8d::53
=====
```

```

▼ Domain Name System (response)
  Transaction ID: 0x1971
  > Flags: 0x8180 Standard query response, No error
  Questions: 1
  Answer RRs: 1
  Authority RRs: 2
  Additional RRs: 5
  ▼ Queries
    > example.com: type MX, class IN
  ▼ Answers
    > example.com: type MX, class IN, preference 0, mx <Root>
  ▼ Authoritative nameservers
    > example.com: type NS, class IN, ns b.iana-servers.net
    > example.com: type NS, class IN, ns a.iana-servers.net
  ▼ Additional records
    > a.iana-servers.net: type A, class IN, addr 199.43.135.53
    > a.iana-servers.net: type AAAA, class IN, addr 2801:500:8f::53
    > b.iana-servers.net: type A, class IN, addr 199.43.133.53
    > b.iana-servers.net: type AAAA, class IN, addr 2801:500:8d::53
    > <Root>: type OPT

```

## 8. Bibliografia

- [1]: Radek Hranický. (2024). Monitorování DNS komunikace , ISA Projects 2024 [online]. Publisher: Brno University of Technology. Retrieved September 20, 2024, [cit. 2024-09-25] Available at: [https://www.vut.cz/studis/student.phtml?script\\_name=zadani\\_detail](https://www.vut.cz/studis/student.phtml?script_name=zadani_detail)
- [2]: ENGRSALMANSHAIKH . (DECEMBER 9, 2014). NETWORK PACKET SNIFFER C++ [online]. Publisher: UNCATEGORIZED . Retrieved September 20, 2024, [cit. 2024-09-25] Available at: <https://engrsalmanshaikh.wordpress.com/2014/12/09/network-packet-sniffer-c/>
- [3]: CS4700/CS5700 . (JANUARY 24, 2011). Simple DNS Client [online]. Publisher: UNCATEGORIZED . Retrieved September 20, 2024, [cit. 2024-09-25] Available at: <https://mislove.org/teaching/cs4700/spring11/handouts/project1-primer.pdf>
- [4]: geeksforgeeks.org . (DECEMBER 07, 2021). Signal Handling in C++ [online]. Publisher: geeksforgeeks.org . Retrieved September 25, 2024, [cit. 2024-09-25] Available at: <https://www.geeksforgeeks.org/signal-handling-in-cpp/>
- [5]: P. Mockapetris . (NOVEMBER 1987). DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION [online]. Publisher: Data Tracker . Retrieved September 25, 2024, [cit. 2024-09-25] Available at: <https://datatracker.ietf.org/doc/html/rfc1035>
- [6]: D. Eastlake, 3rd . (JANUARY 1997). Domain Name System Security Extensions [online]. Publisher: Data Tracker . Retrieved September 25, 2024, [cit. 2024-09-25] Available at: <https://datatracker.ietf.org/doc/html/rfc2065>
- [7]: S. Thomson . (OCTOBER 2003). DNS Extensions to Support IP Version 6 [online]. Publisher: Data Tracker . Retrieved September 25, 2024, [cit. 2024-09-25] Available at: <https://datatracker.ietf.org/doc/html/rfc3596>