Vysoké učení technické v Brně
Fakulta informačních technologií

Síťové aplikace a správa sítí – monitorování DHCP komunikace

Manuál k programu dhcp-stats

18. listopadu 2023 Ondřej Koumar (xkouma02)

Obsah

Úvod

Ne vždy má uživatel možnost sledovat statistiku vytížení síťových prefixů. Některé DHCP servery tuto statistiku poskytují samy, případně se mohou parsovat adresy z jejich logů. Pokud ale takováto možnost není, nezbývá moc jiných možností, než sledovat DHCP provoz odchytáváním paketů a manuálním parsováním IP adres, které jsou klientským stanicím přidělovány. To je use-case, pro který byl program `dhcp-stats` vytvořen.

Program `dhcp-stats` je schopen monitorovat DHCP provoz na vybraném rozhraní, ať už se jedná o ethernetové či bezdrátové, a generovat statistiku pro uživatelem zadaný síťový prefix. Samotné monitorování spočívá v odchytávání a filtraci paketů, které na daném rozhraní projdou, a hledání podstatných informací v nich. V paketu, který je odeslán DHCP serverem, případně klientem, který DHCP službu využívá, jsou umístěny veškeré informace, které jsou potřebné pro generování statistiky pro síťový prefix.

Pokud uživatel nemá potřebu přímo monitorovat DHCP provoz naživo, má možnost využít zpracování *.pcap* nebo *.pcapng* souborů (které si může vygenerovat například pomocí programu Wireshark[1]). To se hodí například v případě, že má již síť zmonitorovanou a nepotřebuje real-time statistiku. V tomto případě je ale počítat s tím, že možnosti programu jsou omezené; vypíše se pouze finální podoba sítě, tedy vytížení prefixů v době, kdy skončilo zachytávání paketů. Je ale možné do programu uměle vložit zpoždění tak, aby šlo vidět zpracování paketů po jednom. Více o této možnosti v kapitole 3.

[1] O programu se dozvíte na https://www.wireshark.org/about.html.

1 Uvedení do problematiky

DHCP (*Dynamic Host Configuration Protocol*) je popsán v rámci RFC 2131. DHCP se skládá za dvou částí – z protokolu zajišťujícího doručování parametrů od serveru ke klientovi a systém alokací IP adres. Je postaven na modelu klient-server, kde server je zařízení, které poskytuje parametry přes přenosovou část protokolu a klient je zařízení, které o tyto parametry DHCP server žádá.

1.1 DHCP pakety

Položky DHCP paketu a jejich vysvětlení:

Pole	**Velikost (B)**	**Popis**
`op`	1	Kód operace zprávy.
`htype`	1	Typ hardwarové adresy.
`hlen`	1	Délka hardwarové adresy.
`hops`	1	Počet relay agentů[2], přes které šla DHCP zpráva.
`xid`	4	ID transakce, které používají jak server, tak klient k identifikaci zpráv.
`secs`	2	Počet sekund od začátku transakce (žádost o adresu nebo obnovení *lease time*).
`flags`	2	Příznak broadcastu.
`ciaddr`	4	IP adresa klienta.
`yiaddr`	4	Nová klientova IP adresa (použito při nabízení a potvrzování adres od serveru).
`siaddr`	4	IP adresa serveru, který má klient kontaktovat při další zprávě.
`giaddr`	4	IP adresa relay agenta, přes kterého se posílají DHCP zprávy.
`chaddr`	16	MAC adresa klientovy síťové karty.
`sname`	64	Jméno serveru (volitelné).
`file`	128	Název souboru, který klient použije pro bootstrapping[3].
`options`	n	Pole volitelných parametrů.

Tabulka 1: Formát zprávy DHCP

Poznámky:

- `op` Může nabývat hodnot BOOTREQUEST a BOOTREPLY.
- Výčet typů hardwarových adres zde.
- `xid` je náhodné číslo generované klientem, používá se po celou dobu komunikace.
- `flags` je ve formátu:

Bits	**Value**
0	B
1-15	MBZ

Tabulka 2: `flags` pole

kde *B* je příznak, že zpráva byla poslána na broadcastovou adresu sítě, *MBZ* je 15 bitů, které jsou vždy nastaveny na 0.

[2] V kontextu DHCP se jedná o proces získání potřebných informací od DHCP serveru.
[3] Relay agenti mají na starost přeposílání paketů mezi klientem a serverem při DHCP komunikaci.

- `options` je proměnné velikosti, klient ale musí být připraven přijmout DHCP zprávu, která má velikost `options` alespoň 312 B. Ne všechny parametry se do pole `options` musí vlézt; pokud je třeba, DHCP server může využít pole `sname` případně pole `file` pro uložení zbývajících parametrů. Pro výčet parametrů, které se v tomto poli mohou vyskytnout, vizte zde.

1.2 DHCP komunikace

1.2.1 DHCP zprávy

Před vysvětlením samotné komunikace je potřeba základní výčet zpráv, které si klient a server v rámci DHCP posílají.

Zpráva	**Využití**
DHCPDISCOVER	Klientovo vysílání pro nalezení dostupných serverů.
DHCPOFFER	Odpověď serveru na DHCPDISCOVER s nabídkou konfiguračních parametrů.
DHCPREQUEST	Může mít několik různých významů: • Klient nemá problém s nabídnutými parametry v DHCPOFFER, zažádá o ně tedy v rámci této zprávy. • Klient se ptá na správnost dříve přidělené adresy po např. restartu systému. • Klient chce prodloužit pronájem (*lease time*) již používané síťové adresy. Kompletní přehled a více informací zde.
DHCPACK	Posílá server, obsahuje konfigurační parametry (1.1) pro klienta do sítě.
DHCPNAK	Posílá server, oznamuje, že server nesouhlasí s parametry, o které si klient zažádal v DHCPREQUEST.
DHCPDECLINE	Posílá klient a oznamuje, že IP adresa je již používána.
DHCPRELEASE	Posílá klient, ruší pronájem IP adresy (z různých důvodů).
DHCPINFORM	Posílá klient a ptá se, jaké má lokální konfigurační parametry, přičemž adresa mu již byla přidělena externě.

Tabulka 3: Typy DHCP zpráv a jejich využití

1.2.2 Posloupnost zpráv

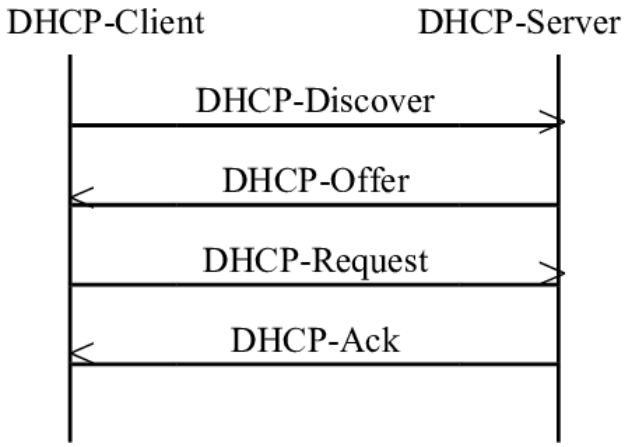
Komunikaci začíná klient tím, že na obecný broadcast[4] vyšle DHCPDISCOVER zprávu. DHCP server tuto zprávu přečte, podívá se na adresy, co má k dispozici a zprávou DHCPOFFER nabídne novému klientovi síťovou konfiguraci. Také do této zprávy na políčko `siaddr` vloží svoji IP adresu, aby mohl být server identifikován po zaslání DHCPREQUEST na broadcast.

Klient se na konfiguraci podívá a pokud mu vyhovuje, oficiálně o tuto konfiguraci zažádá zprávou DHCPREQUEST. Server může přijmout žádost, a tedy poslat potvrzovací DHCPACK zprávu, nebo odmítnout a poslat DHCPNAK zprávu.

V prvním případě má klient ještě možnost odmítnout zprávou DHCPDECLINE, pokud adresu již některé zařízení v síti používá. V tomto případě se klient vrací do stavu, kdy komunikace se serverem probíhá od začátku zprávou DHCPDISCOVER. Pokud ne, adresu přijme a může ji používat v lokální síti. Další komunikace proběhne až klient bude rušit pronájem adresy (případně při zprávě DHCPINFORM).

[4]Protože klient nezná prefix lokální sítě, je zvolena adresa, které rozumí všechna zařízení, a to 255.255.255.255.

V druhém případě se klient vrací do fáze, kdy znovu vysílá DHCPDISCOVER zprávu a komunikace se serverem probíhá od začátku.



Obrázek 1: Typická posloupnost DHCP zpráv s jedním serverem.[2]

Nezapomeňme, že v dosahu se DHCP serverů může vyskytovat více; klient si v tom případě zvolí jeden z nich a jeho adresu vloží do pole `siaddr`, aby se server mohl identifikovat. DHCPREQUEST je také posílán na obecný broadcast a servery musí vědět, komu zpráva patří. Pokud DHCPREQUEST není určen serveru, pak jeho komunikace v tomto bodě končí.

Po vypršení *lease time*, jenž je v DHCP packetu v poli `options` s číslem 51, případně pokud se klient sám rozhodne z jiných důvodů zrušit pronájem IP adresy, klient posílá zprávu DHCPRELEASE. Typicky klient kontaktuje stejný server[5] zprávou DHCPREQUEST se stejnou adresou, jako používal do této chvíle. Server může přijmout, a tedy zaslat DHCPACK, případně odmítnout a klient započíná proces získávání IP adresy od začátku zprávou DHCPDISCOVER.

2 Návrh aplikace

Aplikace je objektově orientovaná, a proto je implementována v C++. V aplikaci je 5 tříd:

- **ArgumentProcessor** má za úkol zpracovávat argumenty příkazové řádky, které byly programu předány. Kontroluje pouze správnost přepínačů a návratovou hodnotu od IpAddressParseru.
- **IpAddressParser** je pomocníkem ArgumentProcessoru. Z Argumentů příkazové řádky kontroluje správnost IP adres, které byly programu předány.
- **PacketSniffer** uchovává informace o použitém rozhraní, případně souboru, ze kterého se načítá, dále data a hlavičky paketů a další pomocné struktury potřebné pro čtení paketů. Extrahuje IP adresy z paketu a předává je IpAddressManageru.
- **IpAddressManager** je hlavní datovou strukturou programu. Uchovává si informace o každé síti, která byla předána jako argument příkazové řádky. Podporuje přidávání i odebírání adres ze sítí.
- **ListInsertable** je třída, ze které dědí všechny ostatní. Je to kvůli tomu, aby všechny objekty, které budou vytvořeny, mohly být přidány do globálního seznamu referencí na objekty.

[5] Opět na broadcast, tentokrát ale lokální sítě.

Dále se v aplikaci nachází datová struktura **NetworkData**. V této struktuře jsou uchovávány veškeré parametry sítě, s vektorem těchto datových struktur pracuje IpAddressManager.

Program je uzpůsoben tomu, aby při přijmutí signálů *SIGINT* a *SIGTERM* byla uvolněna veškerá používaná paměť programu. To se řeší přes globální seznam referencí na objekty, přidá se tam reference na každý vytvořený objekt. Díky tomu, že všechny objekty jsou konstruovány pouze jednou, není vůbec třeba řešit uvolňování paměti po použití objektu, stačí všechnu paměť uvolnit až na konci, paměti je zabíráno minimum.

Vypisování informací o sítích je řešeno knihovnou ncurses. Pokud je překročeno 50 % vytížení prefixu, standardní logovací rutinou `syslog` se tato informace zaznamenává do systémového logu a zároveň je vypsána na standardní výstup. Detailnější popis v sekci 3.

3 Popis implementace

Program je spuštěn funkcí `main`, která se nachází v souboru *dhcp-stats.cpp*. Volá ArgumentProcessor pro zpracování argumentů, poté PacketSniffer, aby začal zpracovávat pakety na rozhraní, případně v souboru. Je tu také obslužná rutina pro signály *SIGINT* a *SIGTERM*, která uvolní všechny alokované zdroje.

3.1 ArgumentProcessor

Třída zodpovědná za zpracování argumentů příkazové řádky. Nejdříve se v metodě `processOptions` zpracují přepínače včetně případných argumentů. To je implementováno pomocí funkce `getopt`. Po zpracování přepínačů se zavolá metoda `processIpPrefixes`, která cyklem prochází zbytek argumentů příkazové řádky, předává je IpAddressParseru na zkontrolování a přidává do vektoru `ipPrefixes`, jsou-li v korektním tvaru.

3.2 IpAddressParser

Zpracuje IP prefix a vrátí pouze příznak *SUCCESS* nebo *FAIL* podle toho, jestli byla IP adresa a maska ve správném formátu. Vstupní metodou je `parseIpAddress`, která má vstupní parametr adresu jako řetězec. Postupně zkontroluje byte po bytu, jestli sedí hodnoty a nakonec i masku. Pro kontrolu samotných bytů a masky pak slouží metody `parseMask` a `parseByte`.

3.3 PacketSniffer

Třída, která zachytává pakety jdoucí přes rozhraní/načítá je ze souboru a extrahuje z nich data, která potřebuje IpAddressManager k tomu, aby si dokázal uchovávat informace o prefixech sítí.

Pro otevření zachytávání paketů jsou z knihovny *libpcap* použity standardně používáné funkce, a to `pcap_open_live` pro živé zachytávání packetů na rozhraní, `pcap_open_offline` pro čtení paketů ze souboru. Další konfigurační funkce, které jsou v kódu použity, jsou `pcap_compile`, která slouží pro přeložení filtru z textové podoby do podoby, kterou může použít funkce `pcap_setfilter`. Ta má za úkol aplikovat filter na aktuální zachytávací session.

Pro samotné zpracování paketů jsou dvě možnosti. Lze použít funkci `pcap_loop` a jako parametr jí předat obslužnou rutinu, která bude pakety zpracovávat; případně v nekonečné smyčce čekat na další pakety a volat rutinu manuálně. V této aplikaci, díky použití objektové orientace[6], byl zvolen přístup s `pcap_next_ex` ve while smyčce. Navíc to má výhodu možnosti lépe kontrolovat chyby.

V obslužné rutině `processPacket` se přeskakují nám nepotřebná data, tedy všechny hlavičky. Uchovávají se dva ukazatele do dat paketu, a to ukazatel na začátek DHCP dat a na začátek DHCP parametrů. Samotná metoda je na obrázku 2.

[6] `pcap_loop` očekává handler jako ukazatel na C-style funkci, tedy nelze se odkazovat na metodu v objektu.

```
void PacketSniffer::processPacket(IpAddressManager& manager)
{
    struct in_addr tmpAddress;
    char tmpAddressStr[INET_ADDRSTRLEN];

    u_char* dhcpData = this->skipToDHCPData();
    u_char* options = this->skipToOptions(dhcpData);
    u_char messageType = this->findDHCPMessageType(options);

    if (messageType == DHCPACK)
    {
        memcpy( dest: &tmpAddress,  src: dhcpData + YIADDR_POSITION,  n: sizeof(struct in_addr));
        inet_ntop( af: AF_INET,  cp: &tmpAddress,  buf: tmpAddressStr,  len: INET_ADDRSTRLEN);
        manager.processNewAddress( &: tmpAddress);
    }
    else if (messageType == DHCPRELEASE || messageType == DHCPDECLINE)
    {
        memcpy( dest: &tmpAddress,  src: dhcpData + CLIENT_IPADDR_POSITION,  n: sizeof(struct in_addr));
        inet_ntop( af: AF_INET,  cp: &tmpAddress,  buf: tmpAddressStr,  len: INET_ADDRSTRLEN);
        manager.removeUsedIpAddr( &: tmpAddress);
    }
    manager.printMembers();
//    std::this_thread::sleep_for(std::chrono::milliseconds(200));
}
```

Obrázek 2: metoda PacketSniffer::processPacket()

Povšimněte si komentáře na úplně posledním řádku metody. Tam si uživatel může nastavit čekání mezi jednotlivými zpracováními paketu. Hodí se to především pro offline sniffing, kde celý soubor je zpracován bleskovou rychlostí. Díky nastavení zpoždění se dá sledovat, jak probíhá vytížení prefixů z *.pcap* souborů průběžně.

Klientova adresa nalezená v paketech se předá IpAddressManageru, ať už se jedná o DHCPACK, DHCPDECLINE či DHCPRELEASE paket.

3.4 IpAddressManager

Důležitou součástí této třídy je datová struktura `NetworkData`. Ta obsahuje tyto informace o síti:

- binární adresa prefixu,
- decimální maska prefixu,
- binární broadcastová adresa,
- počet zabraných adres,
- vektor zabraných adres (binárně),
- maximální počet klientů,
- využití sítě v %,
- logovací příznak,
- adresa jako C-style string.

Na počátku, po zpracování argumentů příkazové řádky, se vytvoří vektor s těmito strukturami, je jich stejný počet, jako počet prefixů dodaných programu. Prozatím se nastaví pouze adresa prefixu binárně i řetězec, maska prefixu, z čehož se zároveň spočítá maximální počet klientů a nakonec broadcast.

V průběhu programu se tyto informace průběžně aktualizují. Po zachytnutí paketu je zavolána metoda `processNewAddress`. Pro každou síť zkontroluje, jestli do ní patří, a pokud ano, tak o tom

```
void IpAddressManager::processNewAddress(struct in_addr& clientAddress)
{
    uint32_t clientAddr = (uint32_t)(ntohl( netlong: clientAddress.s_addr));
    for (NetworkData& network : this->networks)
    {
        if (this->belongsToNetwork( clientAddressShifted: clientAddr,  &: network) &&
            !this->isTaken( clientAddress: clientAddr,  &: network))
        {
            network.numberOfTakenAddresses++;
            network.takenAddresses.push_back(clientAddr);
            this->updateUtilization( &: network);
        }
    }
}
```

Obrázek 3: Metoda IpAddressManager::processNewAddress()

vloží do daného prefixu informace. Kontroluje se také, že adresa ještě není zabraná. Celá metoda na obrázku 3.

Podobně funguje i odstraňování informace o adrese. Prochází se vektory adres ve všech prefixech a pokud se tam nachází, odstraní se a aktualizují se informace v daném prefixu. Celá metoda pak vypadá takto:

```
void IpAddressManager::removeUsedIpAddr(struct in_addr &clientAddress)
{
    uint32_t addrToRemove = (uint32_t)(ntohl( netlong: clientAddress.s_addr));
    for (NetworkData& network : this->networks)
    {
        for (auto takenAddress : iterator<...> = network.takenAddresses.begin();
                  takenAddress < network.takenAddresses.end();
                  takenAddress++)
        {
            if (*takenAddress == addrToRemove)
            {
                network.takenAddresses.erase( position: takenAddress);
                network.numberOfTakenAddresses--;
                this->updateUtilization( &: network);
                return;
            }
        }
    }
}
```

Obrázek 4: Metoda IpAddressManager::removeUsedIpAddr()

Metoda `updateUtilization` funguje na principu jednoduché matematiky. Víme, že maximální počet připojených v podsíti je $2^{32-\text{Maska sítě}} - 2$. Dále $\frac{\text{Počet použitých adres}}{\text{Maximální počet adres}}$ nám dá normalizované využití prefixu. Na % lze převést vynásobením 100.

S adresami se manipuluje v binární formě. Při hledání, zda adresa patří do prefixu, stačí vyshiftovat kontrolovanou adresu o (Maximální maska − Maska sítě) míst a porovnat s adresou prefixu. Hledání broadcastové adresy je zajímavější. Na bitech, které se mohou měnit v rámci prefixu, potřebujeme jedničku. Tam, kde jsou bity pevné, nesmíme měnit. Už to nás vede na bitovou operaci *or*, jen potřebujeme získat tu správnou bitovou masku. Stačí vzít 32 bitů s logickou '1', vyshiftovat

```
void IpAddressManager::setBroadcastAddress()
{
    NetworkData& lastNetwork = this->networks.back();
    uint32_t binaryMask = 0xFFFFFFFF << (MAX_MASK_NUMBER - lastNetwork.decimalMask);
    lastNetwork.broadcast = lastNetwork.address | ~binaryMask;
}
```

Obrázek 5: Metoda IpAddressManager::setBroadcastAddress()

zprava potřebný počet nul a invertovat. Na stejném principu je to v programu implementováno, vizte obrázek 5.

3.5 ListInsertable

Třída ListInsertable implementuje velmi jednoduchý mechanismus podobný garbage collectoru, který uklidí dosud neodklizenou paměť programu, což se hodí nejvíce při neočekávaném ukončení programu. Spočívá v tom, že všechny ostatní třídy dědí od této třídy, což zaručuje, že všechny třídy mohou nabývat typu `ListInsertable`.

Dále je v programu implementován globální seznam, který obsahuje reference na objekty typu `ListInsertable`. Při konstrukci každého objektu si objekt přidá referenci na sebe do globálního seznamu. Díky využití polymorfismu a virtuálnímu destruktoru třídy `ListInsertable` se při odstraňování ze seznamu spustí správný destruktor a objekt se ze seznamu korektně vymaže.

Tento mechanismus dvě výhody:

1. korektní uvolnění paměti při všech situacích, stačí jen zavolat funkci `deleteAll`,
2. odpadá potřeba mazat objekt ihned po jeho použití[7].

4 Testování

4.1 Unit testy

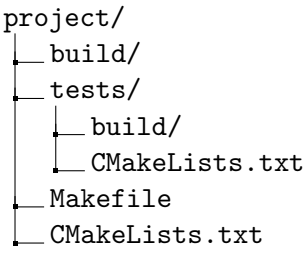
Všechny třídy kromě `packetSnifferu` mají implementovány svoje unit testy. Jako testovací framework byl zvolen googletest. Unit testy naleznete ve složce *tests/*.

Testy pokrývají jen základní funkcionalitu programu, neslouží k verifikaci funkčnosti celého celku, nicméně dávají alespoň trochu jistotu, že program je minimálně zčásti funkční.

- `IpAddressParser`: Testy kontrolují, jestli se správně parsují IP adresy formou úspěch/neúspěch. Objektu se dodávají různě znehodnocené IP adresy.
- `ArgumentProcessor`: Všechny testy spočívají v tom, že se tento objektu pošle vektor řetězců a kontroluje se, jestli tento vektor byl zpracován korektně či ne.
- `IpAddressManager`: Nejdříve se kontroluje správnost vkládání a mazání IP adres, o kterých si uchovává informace, poté se zkouší složitější operace typu vložení mnoha adres, kontrolování log příznaku, atd.
- `AllocList`: V několika málo testech se zkouší, zda se seznam správně plní a poté i maže.

Pro unit testy také existují pomocné soubory, které najdete ve složce *tests/aux/*.

[7] Není to nejlepší programátorská technika, ale v takto malém programu s tím nemůže nastat problém s nedostatkem paměti.



Obrázek 6: Build systém projektu

4.2 Testy chování programu

Ve složce *dhcp_files/* se nachází jednoduchý skript v Pythonu, který generuje *.pcap* soubory. Dají se jednoduše nakonfigurovat parametry adres, které chcete vygenerovat a poté vygenerovaný soubor předat programu. Kontrola chování programu probíhala způsobem, kdy program byl spouštěn s vygenerovaným *.pcap* souborem a souběžně byla kontrolována správnost výstupu se strukturou souboru v programu Wireshark.

5 Build systém projektu

Program pro překlad využívá nástroje `make` a `cmake`. *Makefile* v kořenovém adresáři slouží pouze k volání různých souborů *CMakeLists.txt*, které jsou distribuovány napříč složkami. Díky tomuto systému je možné přeložit různé cíle jednotným způsobem. Struktura důležitých složek projektu pro překlad programu je zobrazena na obrázku 6.

5.1 Makefile

Program je přeložitelný pomocí programu `make`. *Makefile* je uložen v kořenovém adresáři projektu a má několik cílů:

- `all`: Vytvoří složku *build/* a invokuje *CMakeLists.txt* v kořenovém adresáři. Spustitelný program je vytvořen také v kořenovém adresáři projektu.
- `run`: Podívá se, zda existuje spustitelný soubor a spustí jej s argumenty příkazové řádky, které jsou uloženy v *args.txt* souboru (opět v kořenu projektu). Pokud neexistuje, informuje uživatele.
- `test`: Vytvoří složku *tests/build/* a invokuje *CMakeLists.txt* ve složce *tests/*. Lokálně stáhne googletest framework a přeloží testy, výsledný spustitelný soubor se jmenuje *testing*, je uložen ve složce *tests/build/*. Je nutno nejdříve přeložit program.
- `runtest`: Zkontroluje, jestli existuje testovací binárka a spustí ji. Pokud není, informuje uživatele.
- `clean`: Smaže všechny nepotřebné soubory a spustitelné binárky. Uvede projekt do původního stavu.
- `valgrind`: Spustí program *valgrind*, který zaobalí program *dhcp-stats*, pro detekci nesprávně uvolněné paměti programu. Pro výstup valgrindu je výchozí soubor *valgrind.txt*. Lze upravit v `Makefile`.
- `man`: Zobrazí v terminálu manuálovou stránku, která je popsána souboru *dhcp-stats.1*.

5.2 Návod na použití

Po rozbalení repozitáře stačí v kořenovém adresáři projektu invokovat sekvenci příkazů `make` pro přeložení projektu a poté `make run` pro spuštění binárky. Nezapomeňte si nastavit argumenty příkazové řádky v souboru *args.txt*. Přepínače a další argumenty příkazové řádky, které můžete pro program použít, naleznete v manuálové stránce.

6 Dodatek

Veškeré informace ohledně provozu DHCP byly převzaty z jediné literatury, a to z *RFC 2131* protokolu, který je citován v [1].

Reference

[1] Ralph Droms. Dynamic Host Configuration Protocol. RFC 2131, March 1997. URL: https://www.rfc-editor.org/info/rfc2131, doi:10.17487/RFC2131.

[2] Lukas Pürgstein and Hans-Joachim Hof. A system to save the internet from the malicious internet of things at home. In *The Eleventh International Conference on Emerging Security Information, Systems and Technologies (SECURWARE 2017)*, page 184, 09 2017. https://www.researchgate.net/figure/Typical-DHCP-sequence_fig2_319932617 [accessed 15 Nov, 2023].