Vysoké učení technické v Brně
Fakulta informačních technologií

Síťové aplikace a správa sítí

DHCPv6 relay s podporou vložení MAC adresy

18. listopadu 2019 Martin Chládek (xchlad16)

Obsah

1 Zadání

Naší úlohou bylo vytvořit DHCPv6 relay pro IPv6 adresy. Tento program si dává za úkol zprostředkovávat DHCPv6 komunikaci mezi klientem a serverem. Rozhraní, na kterém bude relay naslouchat, bude specifikováno uživatelem, případně pokud uživatel rozhraní neurčí, budou použita všechny validní rozhraní na zařízení. IPv6 adresu serveru určí uživatel. Nad rámec základního relay požadované řešení specifikuje vkládání MAC adresy klienta do přeposílaného packetu. Úloha mohla být řešena v jazyce `C` nebo `C++`. Já jsem si pro realizaci svého projektu zvolil programovací jazyk `C`.

2 Teorie řešení

2.1 Protokol DHCP

2.1.1 Obecný význam

DHCP (Dynamic Host Configuration Protocol) je protokol, který umožňuje serveru dynamicky distribuovat adresy IP a informace o konfiguraci klientům. Běžně DHCP server poskytuje klientovi tyto základní informace:

- IP adresa
- Maska podsítě
- Výchozí brána

2.1.2 DHCPv6

DHCPv6 pracuje s IPv6 adresami. Princip je podobný jako u přidělování IPv4 adres, ale existuje několik rozdílů.

Jedním z nich je, že IPv6 nepodporuje broadcast. Klient tedy neposílá požadavek všesměrovým vysíláním jako je tomu u IPv4, ale své zprávy směruje do multicastové skupiny `ff02::1:2`.

Pro DHCPv6 komunikaci mezi klientem a serverem existují vyhrazené porty. Jedná se konkrétně o port 546 na straně klienta a port 547 na straně serveru.

Stanice mezi sebou komunikují zasíláním tzv. packetů. K zobrazení odeslaných a přijímaných packetů lze použít počítačovou aplikaci Wireshark[1].

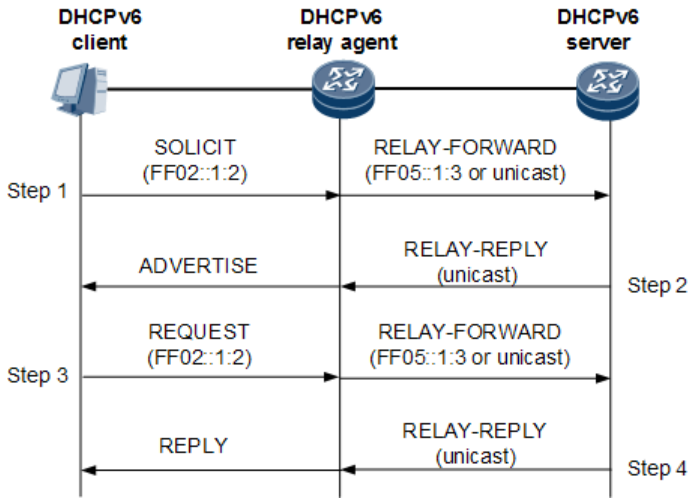
Pro komunikaci je využito UDP (User Datagram Protocol) protokolu, který nenavazuje předem spojení a neposkytuje záruku doručení packetu.

2.2 DHCPv6 relay agent

DHCPv6 relay slouží jako prostředník mezi klientem a serverem. Je to uzel, který předává zprávy DHCPv6 mezi servery a klienty, kteří nejsou na stejné fyzické podsíti.[3].

Komunikaci inicializuje klient zasláním žádosti typu `Solicit` na adresu všech DHCPv6 serverů a agentů (`ff02::1:2`) s dosahem jedné linky. Server odpoví zprávou typu `Advertise`, ve které jsou nabízeny všechny konfigurační parametry. Klient si vybírá nejvhodnější z ohlášených nabídek. Poté co klient vybere, zasílá zprávu `Request`, ve které žádá o potvrzení konfiguračních parametrů. Server nakonec odpovídá zprávou typu `Reply`, ve které potvrzuje, že klient může určené parametry použít a nakonfigurovat podle nich své rozhraní [5].

[1] Více informací k packetovému analyzátoru Wireshark je k dispozici na `<https://www.wireshark.org/>`



Obrázek 1: Schéma DHCPv6 komunikace (zdroj: [2])

DHCPv6 relay původní packet od klienta nijak nemodifikuje. Pouze jej obalí do tzv. „option“, který obsahuje typ `Relay Message`, informaci o délce a obsah původního packetu. Kromě tohoto můžou být do zprávy vloženy ještě jiné „options“, v této implementaci se jedná o `Interface-id` a `Client Link-Layer Address`. Druhá zmíněná nese informaci o MAC adrese klienta, což zadání vyžaduje. Celá tato zpráva je ještě obalena o hlavičku DHCP relay a packet je přeposlán. V případě přijatého packetu od serveru je analyzován a je z něj vyjmuta option typu `Relay Message` a tato část je přeposílána klientovi.

3 Popis implementace

3.1 Zpracování argumentů

Zpracování argumentů je pod taktovkou funkce `parse_arguments`, která kontroluje formát vstupních parametrů a zajišťuje jejich načtení. Jejím srdcem je funkce `getopt_long_only` z knihovny `getopt.h`. Tato funkce načítá pouze parametry začínající znakem '-'[1], případně zda kromě daného přepínače má být zadán i příslušný povinný argument (více o vstupních parametrech v sekci 5). Argumenty a nastavené přepínače jsou načítány do struktury `Arguments`, se kterou je poté pracováno napříč celým programem.

3.2 Výběr rozhraní

Nalezení validních rozhraní pro DHCPv6 relay, případně ověření existence uživatelem zadaného rozhraní zajišťuje v programu funkce `find_interface`. Načtení rozhraní a všech jeho příslušných parametrů je prováděno pomocí funkce `getifaddrs` z knihovny `ifaddrs.h`. Chování závisí na vstupních parametrech:

- Uživatel zadal rozhraní, na kterém má relay naslouchat. V takovémto případě je ověřena existence tohoto rozhraní a v případě úspěchu je zavolána funkce `fork_to_listen_everywhere`, jejíž chování bude blíže vysvětleno v následující kapitole.
- Rozhraní není specifikováno. V takovémto případě je zadáním vyžadováno, že relay má naslouchat na všech rozhraních. Vznikla tedy potřeba v programu uchovat názvy validních rozhraní. Validním rozhraním se rozumí rozhraní, které má lokální IPv6 adresu pro komunikace s klientem v lokální síti a globální IPv6 adresu pro komunikaci se serverem. Protože programovací jazyk `C` nenabízí nic jako je pole řetězců, rozhodl jsem se uchovávání názvů rozhraní implementovat jako 2D pole znaků. V tomto poli je pro každé validní rozhraní vyhrazen jeden řádek pole, kam je název rozhraní nakopírován užitím funkce `strcpy`. Protože v programovacím jazyce `C` nutné při deklaraci specifikovat velikost pole, zvolil jsem hodnotu 20, která by měla běžnému použití ve většině případů stačit. Program je tedy kvůli implementaci omezen na 20 rozhraní.

3.3 Zachytávání packetů

Zachytávání packetů od klienta je úkolem funkce `capture_packet`, které jé předáván mj. i název rozhraní, na kterém má packet zachytávat. Aby bylo možné pochopit jaký význam má tato funkce v programu, je nutné se trochu vrátit a zjistit kdo tuto funkci volá.

Volání funkce `capture_packet` je předmětem funkce `fork_to_listen_everywhere`. Jak už název napovídá, jejím hlavním smyslem je umožnit naslouchání na všech rozhraních. Funkcionalita je následující:

- V případě, že uživatel zadal rozhraní, funkce `capture_packet` je zavolána pro toto již ověřené rozhraní a následující funkcionalita není vykonána.
- Pokud ale uživatel rozhraní nezadal, relay musí naslouchat (odchytávat packety) na všech rozhraních současně. Aby tohle mohlo být realizováno, je potřeba program paralelizovat do několika procesů běžících v jednom čase pro každé rozhraní. Toho je docíleno pomocí funkce `fork`, která proces programu rozštěpí na otcovský a synovský proces[4]. Ve funkci `fork_to_listen_everywhere` je po řádku procházeno pole s názvy rozhraní a pro každé rozhraní (vyjma posledního) je vytvořen synovský proces. V nově vytvořeném procesu je zavolána funkce `capture_packet` a jako parametr je jí předáno příslušné rozhraní. Pro poslední rozhraní v poli se této činnosti ujme sám půvní otcovský proces. Tímto je tedy zajištěno, že funkce `capture_packet` bude zavolána pro každé validní rozhraní paralelně.

U funkce `capture_packet` se tedy očekává, že jako vstupní parametr dostane název rozhraní, na kterém se má naslouchat a nyní už je chování identické pro oba způsoby získání rozhraní. Síťová komunikace zajištěna pomocí funkce `pcap_open_live`, síťový filtr pro zachytávání konkrétních packetů je realizován ve funkci `set_and_compile_filter`, která k jeho nastavení využívá funkcí `pcap_compile` a `pcap_setfilter` z knihovny `pcap.h`. Filtr je nastaven pro DHCPv6 komunikaci od klienta na relay a tedy uvažuje zdrojový port 546 a jako cílový port 547. Připojení do multicast skupiny `ff02::1:2` je zajištěno funkcí `setsockopt`.

Pro samotné zachycení a procházení přijatých packetů je využita funkce `pcap_next`[6], která je volána jako podmínka ve while cyklu. Tělo tohoto cyklu je vyvoláno pro každý zachycený packet, probíhá zde další větvení procesů. Opět je pomocí funkce `fork` vytvořen nový proces, ten vyvolá obslužnou rutinu pro daný packet (více v podsekci 3.4.1) ve formě zavolání funkce `prepare_packet`. Po vykonání rutiny je synovský proces ukončen. Otcovský proces mezitím dále naslouchá, tímto je tedy zajištěno, že na daném rozhraní bude program nepřetržitě obsluhovat příchozí packety.

3.3.1 Životní cyklus obslužné rutiny

Vytvořená implementace uvažuje, že nově vytvoření proces projde následným životním cyklem a na jeho závěru je ukončen. Životní cyklus procesu je následující:

1. Přijetí packetu (zpracování, přidání jednotlivých DHCP options)
2. Odeslání modifikovaného packetu
3. Přijetí odpovědi ze strany serveru (zpracování packetu)
4. Přeposlání jádra packetu klientovi
5. Smrt procesu

Z pohledu posílaných DHCPv6 packetů v síťové komunikaci tedy proces projde jedním ze dvou následujících životních cyklů:

- `Solicit` → `Relay-forward [Solicit]` → `Relay-reply [Advertise]` → `Advertise`
- `Request` → `Relay-forward [Request]` → `Relay-reply [Reply]` → `Reply`

3.4 Zpracování packetů

Přijatý packet je analyzován ve funkci `prepare_packet` užitím struktur hlaviček `ether_header` a `ip6_hdr`, které jsou definovány v příslušných knihovnách. Informace o zdrojové a cílové IP adrese, MAC adrese a názvu rozhraní na kterém byl packet přijat jsou uloženy do struktury `Addresses`, která je dále odkazem předávána mezi funkcemi.

Druhou úlohou této funkce je naplnění hlavičky pro DHCPv6 komunikaci a inicializace struktur pro jednotlivé DHCP options a jejich naplnění validními daty. Všechny struktury byly implementovány mnou, jejich názvy v programu je uveden v závorce. Jedná se o následující:

- **Relay message** (`Opt_relay_message`)
  - Pro tento program nejdůležitější DHCP option. Nese s sebou obsah původní DHCP zprávy přijaté od klienta.
- **Client Link-Layer Address** (`Opt_client_addr`)
  - Nese informaci o MAC adrese klienta
- **Interface-id** (`Opt_interface_id`)
  - Obsahuje informaci o rozhraní, na kterém byl packet přijat

Po naplnění struktur validními daty je volána funkce `Relay_to_Server` a všechny struktury jsou jí odkazem předány.

3.5 Přeposílání packetů

Přeposílání zpráv od klienta a serveru zajišťují 2 funkce, přičemž na sebe sekvenčně navazují. Jako první je potřeba přeposlat packet od klienta serveru, to zajišťuje funkce `Relay_to_Server`. Provoz v druhém směru je úkolem funkce `Relay_to_Client`.

Obě funkce využívají ve svých implementacích funkcí `socket`, `setsockopt` a `bind`. První zmíněná vytváří koncový bod (angl. endpoint), což je síťový uzel, který slouží jako zdroj nebo cíl pro komunikaci. Druhá uvedená poté nastavuje pro daný socket konkrétní parametry, třetí přiřadí socketu adresu[7]. Pro odeslání packetu je používána funkce `sendto`. Všechny uvedené funkce jsou implementovány v knihovně `sys/socket.h`.

3.5.1 Přeposílání DHCPv6 relay → server

Hlavním smyslem funkce `Relay_to_Server` je tedy přeposlat zprávu od klienta serveru a od serveru obdržet odpověď. Nejprve je naplněna hlavička ve formátu pro DHCPv6 relay, která je implementována strukturou `relay_message`. Pro získání `link-adresy` je využita funkce `get_IP_from_interface`, jejíž jediným smyslem je získání a uložení lokální/globální adresy rozhraní do struktury `Addresses`. Následně funkce `Relay_to_Server` vytvoří buffer, do kterého nakopíruje obsah hlavičky a všechny „options“, jenž funkce obdržela na vstupu. Tento buffer reprezentuje odesílaný packet.

Přijetí odpovědi ze strany serveru je zařízeno užitím funkce `recvfrom`. Ta vrací délku přijatého packetu a přijatou zprávu uloží do bufferu. Oba parametry jsou předány funkci `Relay_to_Client`.

3.5.2 Přeposílání DHCPv6 relay → klient

Funkce `Relay_to_Client` analyzuje packet přijatý od serveru, vyjme z něj část určenou pro klienta a přepošle ji. V případě, že uživatel spustil program s nastaveným přepínačem `-d` nebo `-l` z přijaté zprávy zjistí IPv6 adresu nabízenou klientovi a podle příslušného přepínače ji vypíše na standardní výstup nebo do logu užitím `syslog` zpráv.

4 Překlad aplikace

Pro překlad aplikace slouží přiložený soubor `Makefile`.. Postup lze tedy přeložit příkazem `make`:

```
$ make
```

Odstranění spustitelných souborů vzniklé překladem lze provést pomocí:

```
$ make clean
```

Pro správné spouštění je na systémech linux potřebné oprávnění správce (případné spouštění aplikace příkazem `sudo` dodá potřebná oprávnění).

5 Spuštění aplikace

Aplikace vyžaduje pro spuštění následující parametry:

- `-s` Přepínač pro zadání IP adresy DHCP serveru. Musí být doplněn IP adresou ve verzi 6.
- `-i` Přepínač pro výběr rozhraní. Pokud je zvolen, musí být doplněn parametrem ve formátu názvu rozhraní. Není-li zvolen, implementace uvažuje všechna validní rozhraní.
- `-d` Přepínač pro debug na standardní výstup.
- `-l` Přepínač pro logování pomocí syslog zpráv.

Pouze parametr `-s` je povinný a musí být doplněn dle specifikace.

Příklad spuštění:

```
$ ./d6r -i enp0s3 -s 2001:67c:1220:80c::93e5:dd2m -d
```

6 Testování

Program byl testován ručně užitím aplikace Virtual Box (verze 6.0.14) spuštěné na operačním systému MacOS (verze 10.15 Catalina). Operačním systémem pro vývoj a testování byl OS **Ubuntu 18.04.3 LTS Bionic Beaver**.

Vzhledem k absenci nativní IPv6 adresy v místě vývoje byla využita univerzitní VPN a konfigurační soubor `FIT.ovpn`[2]. Na virtuálním stroji byl vytvořen `namespace` a virtuální rozhraní. Z něj byly vysílány zprávy typu `Solicit` užitím příkazu:

```
$ dhclient -v -6 veth2
```

Jako server byl při testování užit DHCPv6 server poskytnutý vedoucím projektu. Pro sledování síťového provozu byl využit program Wireshark.

[2] Více informací k připojení VPN na FIT VUT je k dispozici na `<https://www.fit.vut.cz/units/cvt/net/vpn.php.cs>`

Použitá literatura

[1] [online]. 2005 [cit. 2019-11-16].
URL <http://www.gnu.org/software/libc/manual/html_node/Getopt-Long-Options.html#Getopt-Long-Options>

[2] DHCPv6 Relay. [online]. [cit. 2019-11-18].
URL <https://support.huawei.com/enterprise/en/doc/EDOC1100093189/bff71e87/dhcpv6-relay>

[3] Dynamic Host Configuration Protocol for IPv6 (DHCPv6). [online]. listopad 2018 [cit. 2019-11-18].
URL <https://tools.ietf.org/html/rfc8415>

[4] The fork() System Call. [online]. [cit. 2019-11-14].
URL <http://www.csl.mtu.edu/cs4411.ck/www/NOTES/process/fork/create.html>

[5] Konfigurace adres při implementaci IP verze 6 v koncových sítích. [online]. červen 2012 [cit. 2019-11-18].
URL <http://access.feld.cvut.cz/view.php?cisloclanku=2012060001>

[6] Reference Manual Pages (3PCAP). [online]. 2018 [cit. 2019-11-16].
URL <https://www.tcpdump.org/manpages/pcap.3pcap.html>

[7] Kerrisk, M.: Man7 Linux manual. [online]. 2019 [cit. 2019-11-15].
URL <http://man7.org/linux/man-pages/man2/>