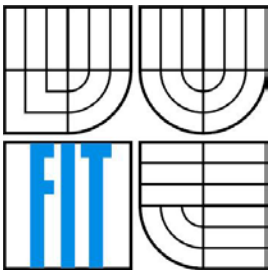
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

BEZPEČNÉ OBJEVOVÁNÍ SOUSEDŮ

SECURE NEIGHBOR DISCOVERY PROTOCOL

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. LUKÁŠ BEZDÍČEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. MATĚJ GRÉGR

BRNO 2014

Abstrakt

Tato zpráva se zabývá návrhem a vývojem ucelené implementace protokolu SEND pro operační systémy GNU/Linux. První část dokumentu obsahuje popis protokolů ND a SEND. Druhá část dokumentu definuje možné bezpečnostní hrozby spojené s nezabezpečeným protokolem ND. Třetí část zprávy popisuje vlastní návrh a implementaci protokolu SEND pod názvem *sendd*. Závěr je věnován shrnutí dosažených výsledků a informacím o dalším vývoji projektu.

Abstract

This report deals with designing and implementing of a complete SEND protocol for operating systems GNU/Linux. The first part of the document contains a description of ND and SEND protocols. The second part of the document defines security threats connected with unsecured ND. The third part of the report describes a design and implementation of SEND protocol named *sendd*. Conclusion of the document is dedicated to a summary of accomplished results and information about future development of this project.

Klíčová slova

IPv6, Objevování sousedů, ND, Bezpečné objevování sousedů, SEND, bezpečnost

Keywords

IPv6, Neighbor Discovery, ND, SEcure Neighbor Discovery, SEND, security

Citace

Bezdíček Lukáš: Bezpečné objevování sousedů, diplomová práce, Brno, FIT VUT v Brně, 2014

Bezpečné objevování sousedů

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Matěje Grégra. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Lukáš Bezdíček
28.5.2014

Poděkování

Chtěl bych poděkovat svému vedoucímu diplomové práce Ing. Matějovi Grégrovi za odborné vedení, za pomoc a rady při zpracování této práce.

Obsah

1 Úvod

Tento dokument popisuje přípravu na řešení a samotné řešení diplomové práce na téma "Bezpečné objevování sousedů". Mým úkolem bylo prostudovat bezpečnostní rizika lokálních sítí s ohledem na *Objevování sousedů* (Neighbor Discovery, ND) protokolu IPv6, seznámit se s protokolem *Bezpečné objevování sousedů* (SEcure Neighbor Discovery, SEND) a následně vytvořit jeho funkční implementaci pro operační systémy GNU/Linux.

V první části dokumentu je uvedeno seznámení s cílem práce. V následující části je uveden popis protokolu ND. Na tuto část navazují kapitoly obsahující charakteristiku bezpečnostních hrozeb spojených právě s tímto protokolem. Poté následuje popis jeho bezpečnostního rozšíření s názvem *Bezpečné objevování sousedů* (SEND). Hlavním cílem této práce je vytvořit softwarové vybavení pro podporu protokolu SEND pro operační systémy GNU/Linux, které by mohlo být provozováno v reálném prostředí. O návrhu a implementaci pojednávají kapitoly 4 a 5.

Pro správnou činnost IPv6 sítí je nutný protokol ND. Původní specifikace tohoto protokolu ale neobsahuje žádné bezpečnostní prvky k ochraně proti možným útokům. Předpokládalo se, že k zabezpečení IPv6 sítí včetně ND bude sloužit protokol IPsec. To ovšem z důvodů praktických problémů není u tohoto protokolu možné (viz kapitola 3). Proto vznikl protokol SEND jako bezpečnostní rozšíření zmíněného protokolu. V současné době ale neexistuje žádná úplná implementace protokolu SEND na operačních systémech GNU/Linux, která by mohla být provozována v reálném prostředí.

Před návrhem a vlastní implementací je však dle mého názoru nutné se detailně obeznámit s výše uvedenými protokoly. Je také vhodné se seznámit s bezpečnostními hrozbami týkající se protokolu Objevování sousedů a při návrhu implementace brát na tyto hrozby ohled. Při návrhu a implementaci je dále užitečné využít zkušenosti získané ze současných implementací protokolu SEND. Závěr dokumentu je věnován shrnutí mých výsledků, kterých jsem dosáhl.

2 Objevování sousedů

V této části dokumentu je popsán protokol *Objevování sousedů*. Text v této části je založen zejména na informacích získaných z oficiální specifikace protokolu RFC 4861 [1].

Protokol *Objevování sousedů* (Neighbor Discovery Protocol, NDP) je používán uzly k nalezení okolních uzlů v jejich lokální IPv6 síti. Protokol dále umožňuje nalezení jejich linkových (fyzických) adres a poskytuje prostředky pro sledování jejich dosažitelnosti. Tento protokol lze rozdělit na dvě části, *Objevování sousedů* (ND) a *Objevování směrovačů* (RD).

Proto, aby mohly dva uzly v Internetu spolu komunikovat, musí znát svoje IPv6 adresy. Podle IPv6 adresy a prefixu uzly zjistí, zda leží v jedné lokální síti. Pokud ale chtějí spolu komunikovat uzly nacházející se ve stejné lokální síti, musí znát svoje linkové adresy. Pro zjištění linkové adresy v IPv4 sítích se používá samostatný protokol s názvem Address Resolution Protocol, zkráceně ARP. Ten pracuje tak, že tazatel rozešle na všesměrovou IPv4 adresu 255.255.255.255 ARP dotaz na vlastníka určité IPv4 adresy. Pokud je takový vlastník právě připojen k síti, tak tazateli zašle odpověď se svojí linkovou adresou. Tvůrci IPv6 se rozhodli mechanismus pro zjištění linkové adresy zabudovat přímo jako součást protokolu. Výsledkem byl protokol ND, který kromě tohoto řeší i celou řadu dalších problémů souvisejících se vzájemnou komunikací uzlů na linkové vrstvě. Protokol definuje mechanismy pro následující účely:

- Hledání směrovačů
- Zjišťování prefixů
- Zjišťování parametrů
- Automatická konfigurace adres
- Zjišťování linkových adres
- Určování následujícího uzlu
- Ověřování dosažitelnosti sousedů
- Detekce duplicitních adres
- Přesměrování

ND definuje pět typů ICMP zpráv. Tyto zprávy slouží k následujícím účelům:

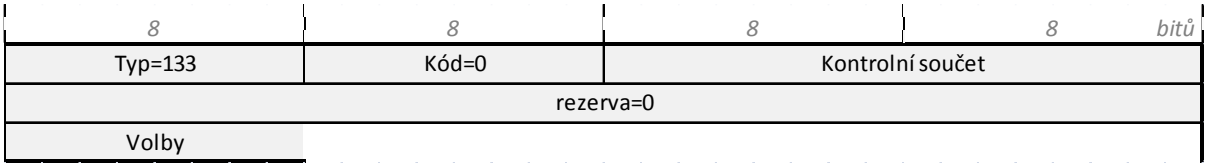
- Výzva směrovači (Router Solicitation, RS)
- Ohlášení směrovače (Router Advertisement, RA)
- Výzva sousedovi (Neighbor Solicitation, NS)
- Ohlášení souseda (Neighbor Advertisement, NA)
- Přesměrování (Redirect)

2.1 Zprávy protokolu ND

V této kapitole je uveden popis protokolu ND z pohledu zasílaných zpráv. Kapitola obsahuje informace o struktuře zpráv a k jakým účelům jsou používány. ND zprávy mohou obsahovat 0 a více voleb.

2.1.1 Výzva směrovači

Koncové uzly rozesílají ICMP zprávy typu *Výzva směrovači* proto, aby jim směrovače co nejdříve vygenerovaly zprávy *Ohlášení směrovače*. Výzvy jsou zasílány na skupinovou adresu pro všechny směrovače ve stejné lokální síti a tou je ff02::2 [1]. Ke zprávě je dále možné připojit volbu s linkovou adresou odesilatele. Jedná se o linkovou adresu síťového rozhraní, ze kterého byla zpráva odeslána. Formát zprávy a volby s vlastní linkovou adresou jsou uvedeny na obrázcích 1 a 2.

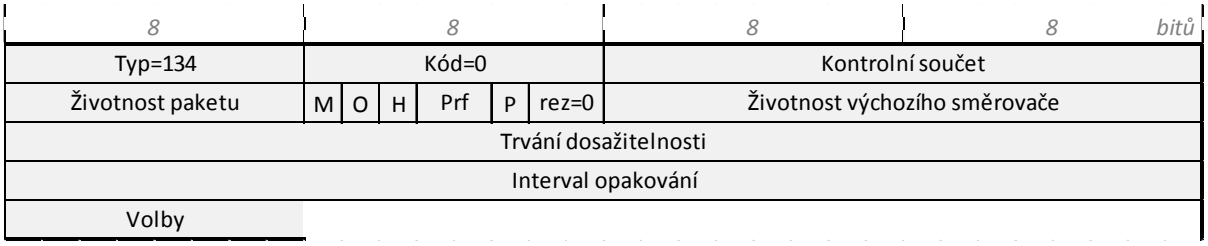


Obrázek 1: Výzva směrovači

Obrázek 2: Zdrojová linková adresa

2.1.2 Ohlášení směrovače

ICMP zprávy *Ohlášení směrovače* jsou rozesílány buď periodicky, nebo jako odpověď na *Výzvy směrovači*. Tyto zprávy jsou nosným pilířem bezstavové konfigurace [2]. Formát základní zprávy Ohlášení směrovače je znázorněn na obrázku 3.



Obrázek 3: Ohlášení směrovače

Zpráva obsahuje výchozí životnost paketu, která informuje koncové uzly, jakou hodnotu mají vkládat do odesílaných paketů. V ohlášení se pak nachází několik příznaků. Bitový příznak *M* (Managed address configuration, stavová konfigurace adres) nastaven na hodnotu 1 indikuje, že přidělování adres i komunikačních parametrů je řízeno prostřednictvím DHCPv6. Bitový příznak *O* (Other configuration, stavová konfigurace ostatních parametrů) s hodnotou 1 značí, že DHCPv6

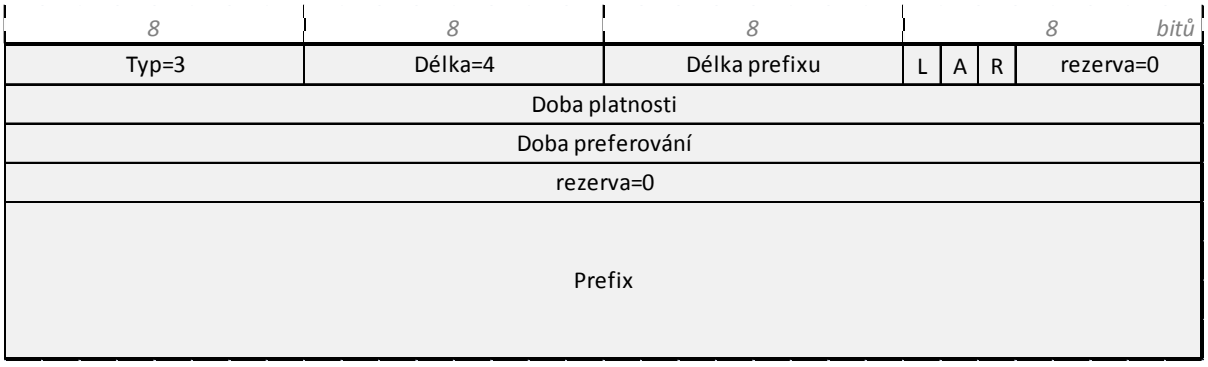
rozhoduje pouze o komunikačních parametrech. Příkladem takových parametrů mohou být informace týkající se DNS. Ve specifikaci pro podporu mobility RFC 3775 [3] byl přidán bitový příznak *H* (Home agent, domácí agent) indikující, že směrovač odesílající takováto ohlášení slouží i jako domácí agent. Domácí agent je směrovač v domácí síti, který vytváří tunel mezi domácí sítí a mobilním uzlem [3]. Příznaku *Prf* (Default router preference), který je součástí specifikace RFC 4191 [4], informuje o tom, jestli má být směrovač preferován nad jinými výchozími směrovači v síti. Možné hodnoty tohoto příznaku jsou uvedeny v tabulce 1.

Prf	Význam
01	vysoká preference
00	střední preference
11	nízká preference
10	vyhrazeno (nesmí se používat)

Tabulka 1: Preference výchozího směrovače

Příznak *P* je součástí specifikace RFC4389 [5], podle které je možné vytvořit most mezi směrovači. Zpráva dále nese informaci o životnosti výchozího směrovače. To je doba (v sekundách), po kterou bude směrovač sloužit jako výchozí. Hodnota 0 signalizuje, že se nejedná o výchozí směrovač, a proto by si ho koncové uzly neměly ukládat do seznamu výchozích směrovačů. Hodnota příznaku *Prf* je v tomto případě příjemcem ignorována. *Trvání dosažitelnosti* (Reachable time) je čas v milisekundách, po který má být uzel považován za dosažitelný. Poté následuje *Interval opakování* (Retrans time), to je doba v milisekundách mezi dvěma výzvami sousedovi. Tyto dva údaje jsou používány algoritmem pro ověřování dosažitelnosti sousedů. Hodnoty 0 znamenají, že trvání dosažitelnosti a/nebo interval opakování nejsou tímto směrovačem specifikovány.

Ke zprávě je možné přiložit různé volby. Možné volby jsou: volba, kde směrovače sdělují svoji linkovou adresu, volba oznamující MTU sítě nebo volba s *Informacemi o prefixu* (Prefix Information). Formát volby s informacemi o prefixu ilustruje obrázek 4.



Obrázek 4: Informace o prefixu

Volba s *Informacemi o prefixu* obsahuje zejména vlastní prefix a délku prefixu (Prefix length). Prefix je IP adresa nebo úvodní část IP adresy. *Délka prefixu* udává, kolik bitů ze zmíněné

adresy je platný prefix. Za délkou prefixu následuje trojice bitových příznaků. Příznak *L* (on-Link, na lince) signalizuje, jestli se prefix používá pro určení, který uzel je lokální. Lokální uzel je uzel přímo dosažitelný linkovou vrstvou [2]. Druhý příznak *A* (Autonomous address-configuration, autonomní konfigurace adres) informuje o tom, zda-li je prefix možné použít k automatické konfiguraci vlastní adresy. Poslední příznak *R* (Router address, adresa směrovače), který byl uveden také v již zmíněné specifikaci RFC 3775 [3], nastaven na hodnotu 1 říká, že položka *Prefix* obsahuje úplnou globální adresu směrovače. Tato adresa je kompatibilní s oznamovanými prefixy.

Doba platnosti (Valid lifetime) je počet sekund od doby, kdy byla zpráva odeslána, během kterých je možné podle prefixu určit, zda-li je nějaký uzel lokální. Druhý časový údaj v sekundách je *Doba preferování* (Preferred lifetime). To je čas, během kterého budou adresy vytvořené z uvedeného prefixu v průběhu automatické konfigurace preferovány. Po vypršení *Doby preferování* se adresa stává *Odmítanou* (Deprecated) [2]. V tomto stavu je adresa sice stále platná, ale brzy se stane neplatnou. Takováto adresa by se měla pro nová spojení přestat používat a měla by být nahrazena preferovanou adresou. Po vypršení doby platnosti se adresa stává neplatnou.

Každý směrovač zasílá zprávu *Ohlášení směrovače* informující o své dostupnosti periodicky do všech sítí, k nimž je připojen. Koncový uzel si po obdržení všech Ohlášení směrovačů vytváří seznam výchozích směrovačů. Směrovače generují zprávu v náhodných intervalech a dostatečně často na to, aby se koncové uzly dozvěděly o jejich dostupnosti během několika málo okamžiků, ale ne dostatečně často na to, aby koncové uzly mohly podle nepřítomnosti zpráv určovat, zda došlo k selhání směrovače. Pro určení, zda došlo k selhání slouží samostatný algoritmus s názvem *Ověřování dosažitelnosti sousedů*.

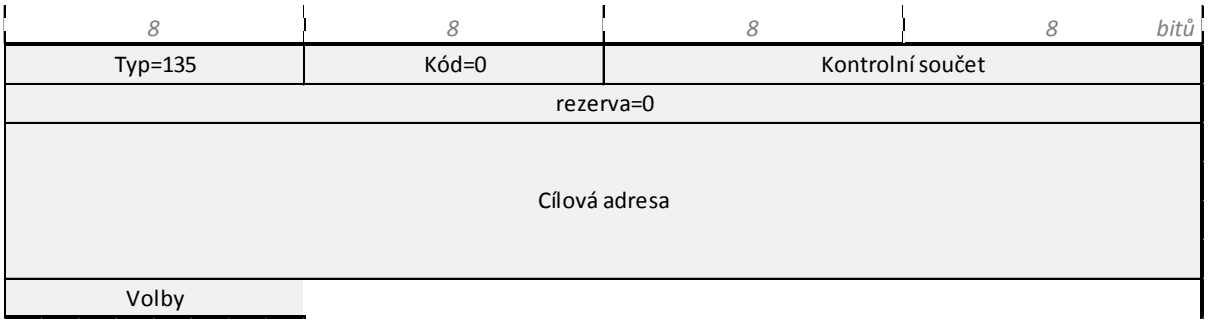
Ohlášení směrovače obsahuje seznam prefixů používaných pro určení, jestli je nějaký uzel lokální a/nebo pro autonomní konfiguraci adres. Příznaky spojené s prefixy určují použití konkrétního prefixu. Koncové uzly si vytvářejí a udržují svůj seznam oznámených prefixů. Podle nich pak určují, zda se cílový uzel nachází ve stejné lokální síti nebo až za směrovačem. Je nutné poznamenat, že se může cílový uzel nacházet ve stejné lokální síti, i když jeho adresa není pokryta žádným oznámeným linkovým prefixem. V takovémto případě může směrovač informovat odesilatele zprávou *Přesměrování*, že cílový uzel je soused.

Směrovače mohou pomocí *Ohlášení směrovače* informovat koncové uzly o tom, jakým způsobem mají provést automatickou konfiguraci adres. Směrovače tak například mohou určit, jestli by uzly měly použít DHCPv6 a/nebo autonomní (bezstavovou) konfiguraci adres.

Ohlášení směrovače mohou také obsahovat parametry Internetu, mezi které patří například životnost paketu (hop limit), které by koncový uzel měl použít pro odesílání paketů. Zprávy také mohou volitelně obsahovat parametry lokální sítě například MTU (Maximum transmission unit). Tato možnost tak usnadňuje centralizovanou správu důležitých parametrů, které mohou být prostřednictvím směrovače automaticky šířeny na všechny připojené koncové uzly.

2.1.3 Výzva sousedovi

ICMP zpráva *Výzva sousedovi* obsahuje zejména IP adresu, ke které si odesilatel přeje získat linkovou adresu. Odesilatel ke zprávě dále může připojit volbu s jeho vlastní linkovou adresou. Formát zprávy je ilustrován na obrázku 5.

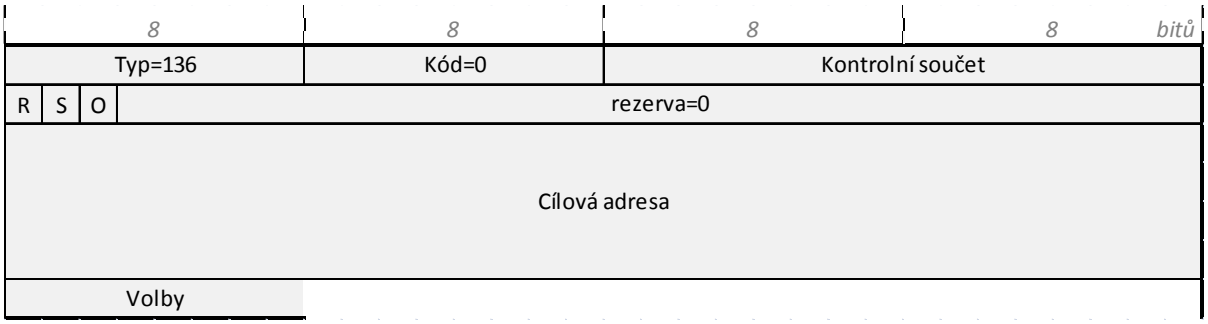


Obrázek 5: Výzva sousedovi

Zprávy typu *Výzva sousedovi* je možné použít také pro detekci duplicitních adres. Uzel pošle Výzvu sousedovi s adresou, kterou si právě vygeneroval a chystá se používat. Pokud na tuto výzvu obdrží *Ohlášení souseda*, tak to znamená, že danou adresu už někdo používá a nemůže si ji tak přiřadit.

2.1.4 Ohlášení souseda

ICMP zpráva *Ohlášení souseda* obsahuje především IP adresu uzlu, kterého se ohlášení týká (odesilatel). Dále se ke zprávě v podstatě vždy přikládá volba, kde odesilatel sděluje svoji linkovou adresou (Volba *Zdrojová linková adresa*). Výjimkou jsou případy, kdy je ohlášení zasíláno jako odpověď na individuální *Výzvu sousedovi* v rámci mechanismu pro ověřování dosažitelnosti sousedů. Zde je možné volbu vynechat a zamezit tak přebytečnému síťovému provozu, protože odesilatel takovéto výzvy adresu již zná. Zpráva kromě toho obsahuje tři příznaky. Příznak *R* (Router) informuje, že odesilatelem je směrovač. Příznak *S* (Solicited) signalizuje, zda ohlášení bylo vyžádáno Výzvou sousedovi (v rámci mechanismu pro ověřování dosažitelnosti sousedů). Poslední příznak *O* (Override) indikuje, že by si příjemce měl aktualizovat existující záznam v *cache sousedů* s danou IP adresou informacemi uvedenými v ohlášení. Formát zprávy je znázorněn na obrázku 6.



Obrázek 6: Ohlášení souseda

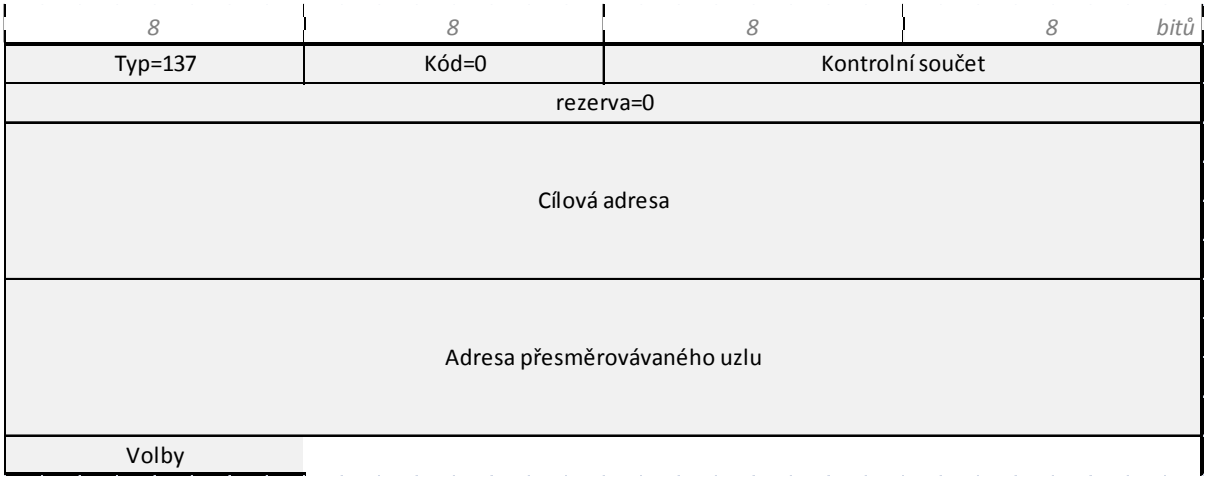
Hledání linkové adresy probíhá tak, že se z cílové IP adresy vytvoří skupinová adresa vyzývaného uzlu. Uzel vezme posledních 24 bitů z cílové IP adresy a připojí je za prefix ff02:0:0:0:0:1:ff00::/104. Uzel na výslednou skupinovou adresu zašle *Výzvu sousedovi*. Pokud je cílový uzel aktivní a výzvu příjme, tak na ni reaguje zprávou typu *Ohlášení souseda* obsahující informace o jeho linkové adrese. Je možné zajistit výměnu linkových adres obou komunikujících stran během jediné transakce, protože vyzyvatel může vložit svoji linkovou adresu již do *Výzvy sousedovi*.

Jakmile uzel zjistí, že došlo ke změně jeho linkové adresy, tak může rozeslat všem sousedním uzlům několik zpráv typu *Ohlášení souseda* informující o jeho nové adrese. Uzly, které mají původní adresu uloženou v *cache sousedů*, si po obdržení zprávy adresu aktualizují. Ostatní uzly nevyžádaná *Ohlášení souseda* ignorují.

Ověřování dosažitelnosti sousedů zjišťuje, zda nedošlo k selhání souseda nebo cesty k sousedovi. Používají se dva základní mechanismy ověřování. Je-li to možné, tak protokoly vyšší vrstvy poskytují pozitivní potvrzování, že odeslané pakety byly úspěšně doručeny. Pokud toto možné není, tak uzel posílá *Výzvy sousedovi*, na které očekává odpověď v podobě zprávy *Ohlášení souseda*. Pokud *Ohlášení souseda* nedorazí, došlo pravděpodobně k selhání. Pro omezení nadbytečného síťového provozu jsou výzvy zasílány pouze sousedům, kterým jsou právě posílány pakety.

2.1.5 Přesměrování

Směrovače posílají zprávy *Přesměrování*, aby informovaly koncový uzel o existenci lepšího následujícího uzlu na cestě k cíli. Koncový uzel tak může být přesměrován na vhodnější směrovač nebo přímo na cílový koncový uzel v případě, že se nachází v jeho sousedství. Formát zprávy je uveden na obrázku 7.



Obrázek 7: Přesměrování

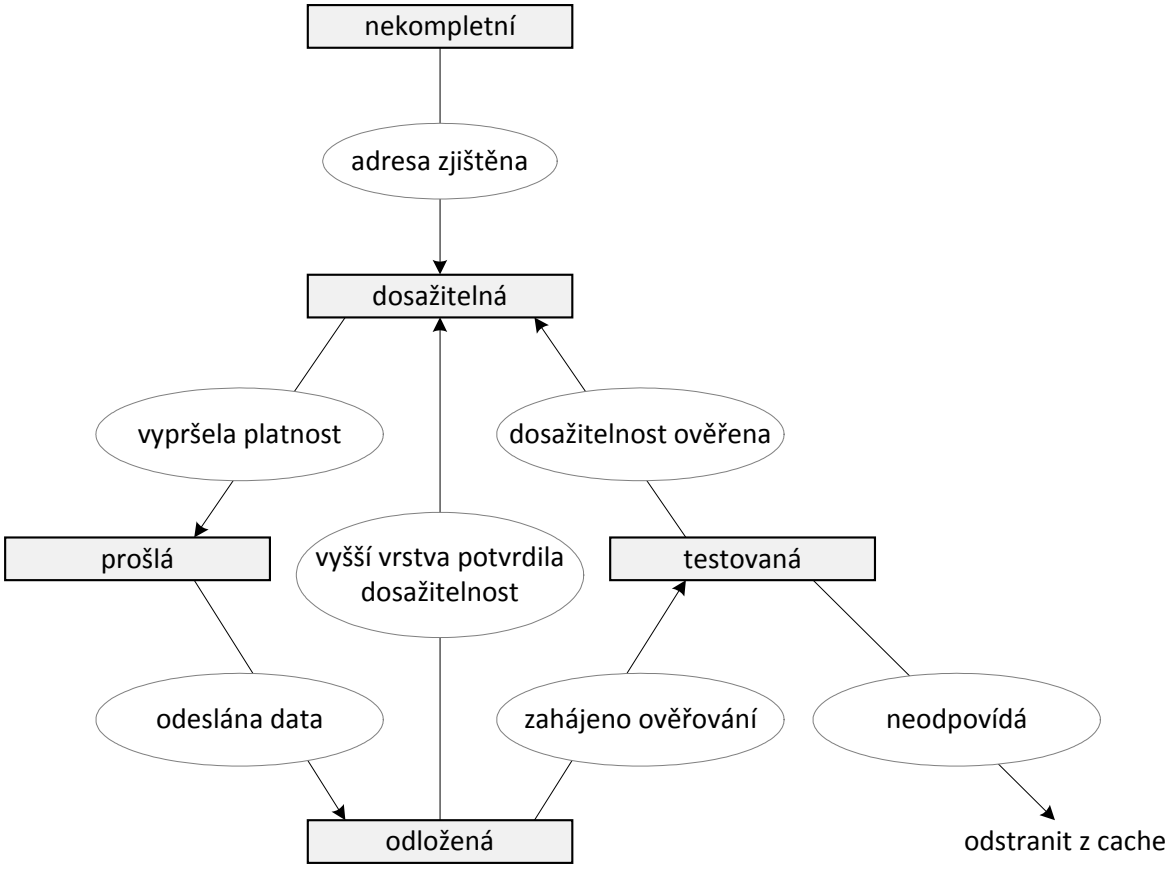
ICMP zpráva *Přesměrování* se skládá hlavně z IP adresy vhodnějšího následujícího uzlu (Cílová adresa) a IP adresy přesměrovávaného koncového uzlu. Ke zprávě je možné dále připojit volbu s linkovou adresou vhodnějšího následujícího uzlu nebo volbu s co možná největší částí paketu, který přesměrování vyvolal.

2.1.6 Cache sousedů

Cache sousedů je datová struktura obsahující množinu záznamů o jednotlivých sousedech, kterým byly v poslední době zasílány pakety. Jednotlivé záznamy jsou identifikovány podle lokální linkové IP adresy a obsahují linkovou adresu, příznak signalizující jestli je soused směrovač nebo koncový uzel, ukazatel na frontu paketu čekající na zjištění linkové adresy, atd. Záznamy dále nesou informace používané algoritmem pro ověřování dosažitelnosti sousedů: stav, počet výzev, na které doposud nebyla obdržena odpověď a plánovaný čas následujícího ověřování dosažitelnosti. Stavy položek v *cache sousedů* jsou následující:

- nekompletní (incomplete) - probíhá zjišťování adresy, linková adresa zatím není známa
- dosažitelná (reachable) - uzel byl v poslední době dosažitelný (v předchozích desítkách sekund)
- prošlá (stale) - není známo, jestli uzel je dosažitelný a v poslední době mu nebyly posílány žádné pakety, není ale třeba, aby se prověřovala jeho dosažitelnost
- odložená (delay) - není známo, zda je uzel dosažitelný a v poslední době mu byly posílány pakety, pokud v nejbližší době nebude jeho dosažitelnost potvrzena některým protokolem vyšší vrstvy, vyšle se Výzva sousedovi
- testovaná (probe) - není známo, jestli je uzel dosažitelný a byla mu zaslána Výzva sousedovi pro ověření jeho dosažitelnosti

Přechody mezi těmito stavy znázorňuje obrázek 8.



Obrázek 8: Přechody mezi stavy položek v cache sousedů [2]

2.2 Hrozby v Objevování sousedů

V této kapitole jsou popsány možné bezpečnostní hrozby a útoky proti *protokolu Objevování sousedů*. Existují obecně tři druhy bezpečnostních hrozeb [6].

- Útoky *Přesměrování*, ve kterých zlovolný uzel přesměrovává pakety od směrovače nebo do jakéhokoli jiného oprávněného příjemce k jinému uzlu na lince.
- Útoky typu *Odepření služby* (Denial of Service, DoS), ve kterých zlovolný uzel brání v komunikaci mezi uzlem pod útokem a ostatními uzly nebo uzlem s určitou adresou.
- Záplavové útoky *Odepření služby*, ve kterých zlovolný uzel přesměrovává síťový provoz jiných uzlů k oběti, a tím ho zahlcuje množstvím nesmyslných zpráv.

2.2.1 Podvržení Výzvy sousedovi a Ohlášení souseda

Uzly na lince používají zprávy *Výzva sousedovi* a *Ohlášení souseda* k vytvoření propojení mezi IP adresou a linkovou adresou. Existují dva případy, kdy si uzel vytváří záznam v *cache sousedů*. Oba dva nastávají při přijetí *Výzvy sousedovi*. V případě, kdy uzel obdrží *Výzvu sousedovi* s linkovou adresou odesilatele, si ji může vložit do *cache sousedů*. Jestliže při detekci duplicitních adres uzel

obdrží výzvu na adresu, kterou se teprve chystá používat, předpokládá se, že došlo ke kolizi adres. Navíc v takové situaci musí o tuto adresu přestat žádat.

Útočník tak může posíláním *Výzev sousedovi* s jinou zdrojovou linkovou adresou nebo *Ohlášení souseda* s linkovou adresou jiného uzlu přesměrovat pakety od legitimních příjemců na jiné uzly v síti. Útok je úspěšný, protože staré záznamy v *cache sousedů* jsou přepsány záznamy s novou linkovou adresou. Pakety jsou dále přesměrovávány, jestliže útočník posílá platné odpovědi na následné *Výzvy sousedovi* zasílané v rámci ověřování dosažitelnosti souseda.

Tento princip může být použit i při útoku typu *Odepření služby* a to, když útočník pošle *Výzvy sousedovi* a *Ohlášení souseda* s nepoužívanou linkovou adresou. Nicméně takovýto útok by trval jen několik desítek sekund, protože mechanismus pro ověřování dosažitelnosti sousedů by odstranil neplatné záznamy z *cache sousedů* a po přijetí odpovědí na *Výzvy sousedovi* by je nahradil platnými. Pokud by si útočník přál, aby útok trval déle, musel by svými vymyšlenými linkovými adresami odpovídat na následující výzvy.

2.2.2 Selhání Ověřování dosažitelnosti souseda

Mechanismus *Ověřování dosažitelnosti souseda* slouží uzlům ke sledování dosažitelnosti lokálních uzlů nebo směrovačů. Obyčejně se při tom spoléhají na informace z vyšších vrstev síťového protokolu. Pokud však nastane na vyšší vrstvě dostatečně dlouhá prodleva nebo komunikující protistrana přestane odpovídat, spustí se *Ověřování dosažitelnosti souseda*. To spočívá v tom, že uzel zašle protistraně cílenou *Výzvu sousedovi* a ten, jestliže je dosažitelný, odpoví *Ohlášením souseda*. Pokud vyzyvatel neobdrží odpověď, pokusí se zaslat výzvu ještě několikrát. Neobdrží-li odpověď ani pak, smaže si náležitý záznam z *cache sousedů*. Jakmile je záznam z cache smazán, není možné pokračovat v komunikaci na vyšší vrstvě.

Nejsou-li *Ohlášení souseda* nějakým způsobem chráněny, může útočník posílat uměle vytvořená ohlášení jako odpověď na zprávy *Výzva sousedovi* protokolu pro ověřování dosažitelnosti. Oběť se tak nedozví, že cílový uzel už není dosažitelný. Konkrétní následky závisí na tom, z jakého důvodu se stal cílový uzel nedosažitelný a jak by se chovala oběť, kdyby se o jeho nedosažitelnosti řádně dozvěděla. Jedná se o útok *Odepření služby*.

2.2.3 Odepření služby při Detekci duplicitních adres

Uzel připojující se do sítě, kde se IP adresa získává prostřednictvím bezstavové automatické konfigurace, musí ověřit, není-li adresa již používána. To uzel provede tak, že na vygenerovanou adresu zašle *Výzvu sousedovi*. Pokud obdrží odpověď ve formě *Ohlášení souseda* s danou adresou, adresu už někdo používá.

Útočník tak může *Ohlášením souseda* reagovat na všechny pokusy oběti o detekci duplicitních adres nebo může posílat *Výzvy sousedovi* a předstírat tak, že došlo ke kolizi adres. Oběť

si pak nemůže přiřadit žádnou z adres a není schopna se přihlásit do sítě. Tento problém se netýká jen sítí s automatickou konfigurací adres, ale všech sítí, kde se používá *Detekce duplicitních adres* protokolu ND.

2.2.4 Zlovolný směrovač (brána)

Útočník se může v podsíti vydávat za legitimní směrovač (bránu) a hromadně nebo cíleně rozesílat podvodná *Ohlášení směrovače*. Připojující se uzel, který si zvolí útočníka za výchozí směrovač, se stává obětí. Útočník pak má možnost stáhnout na sebe veškerý síťový provoz oběti nebo spustit Man-in-the-middle útok. Aby útočník znemožnil oběti si zvolit legitimní směrovač, může periodicky rozesílat *Ohlášení směrovače* se zdrojovou adresou legitimního směrovače s položkou *Životnost výchozího směrovače* obsahující hodnotu 0. Pro zakrytí stop může útočník před ukončením činnosti poslat zprávu *Přesměrování*, díky které bude uzel přesměrován na legitimní směrovač.

2.2.5 Odstranění výchozího směrovače

Jestliže má odesilatel seznam výchozích směrovačů prázdný, předpokládá, že se příjemce nachází v lokální síti [7]. Pokud tedy uzel útočníkovi uvěří, že na lince není žádný výchozí směrovač, pokusí se s pomocí ND posílat pakety přímo. Útočník pak může za pomoci podvržených *Výzev sousedovi* a *Ohlášení souseda* odepřít možnost komunikace nebo se i zmocnit zpráv určených pro příjemce nacházející se mimo lokální síť. Jedna možnost jak odstranit výchozí směrovač, je přetížit jej množstvím nesmyslných zpráv.

2.2.6 Podvržená zpráva Přesměrování

Zpráva *Přesměrování* může být použita k zasílání paketů na jakoukoli jinou linkovou adresu na lokální síti, než pro jakou je určena. Útočník může poslat zprávu *Přesměrování*, kde je uvedena jako linková adresa odesilatele adresa legitimního směrovače. Oběť se pak domnívá, že zpráva přišla od skutečného směrovače a provede přesměrování. Toto přesměrování bude platit, dokud útočník bude odpovídat na zprávy mechanismu pro ověřování dosažitelnosti souseda. Přesměrováním lze provést i útok *Odepření služby*.

2.2.7 Útok Přehrávání

Při tomto útoku útočník odesílá platné zprávy, které na síti zachytil dříve. Všechny ND zprávy jsou náchylné k tomuto útoku, a to i v případě, že jsou kryptograficky chráněny proti nepovolaným změnám jejich obsahu. Proti tomuto útoku se lze v komunikaci typu výzva-odpověď chránit tím, že každá dvojice zpráv obsahuje shodné unikátní číslo. Proto není možné poslat na výzvu odpověď zachycenou na síti dříve, protože neobsahuje odpovídající unikátní číslo. U samostatných zpráv, kterými jsou zprávy *Přesměrování* nebo pravidelné *Ohlašování směrovače*, je možné se chránit

počítadlem nebo časovou značkou. Časová značka omezuje jejich platnost na určitou dobu. Je potřeba, aby uzly měly alespoň přibližně synchronizované hodiny reálného času a sledovaly rozdíly mezi časy ostatních uzlů. Zmíněná řešení předpokládají použití zabezpečení proti nepovolaným změnám v obsahu zpráv.

2.3 Jak chrání SEND protokol ND

Díky volbám s parametry CGA a s RSA podpisem je chráněn původ a integrita ND zpráv. Časová značka chrání ND zprávy před opětovným posíláním a umožňuje příjemci si ověřit, zda byla vytvořena v aktuální dobu. Pomocí volby unikát příjemce ověří, zda je obdržené ohlášení opravdu odpovědí na jeho předchozí výzvu.

Naproti tomu SEND nezaručuje důvěrnost zpráv protokolu ND. Protokol SEND také neposkytuje ochranu před bezpečnostními hrozbami na nezabezpečené linkové vrstvě. Nezaručuje tak, že linková adresa, kterou se uzel prezentuje, uzlu skutečně náleží.

3 Bezpečné objevování sousedů

V této části dokumentu je popsán protokol *Bezpečné objevování sousedů* (SEND). Informace uvedené v této části jsou založené na oficiální specifikaci protokolu RFC 3971 [8] a navazují na kapitolu Objevování sousedů.

Původní specifikace protokolu ND [7] doporučuje k zabezpečení *Objevování sousedů* a mechanismu pro automatickou konfiguraci adres použití *Autentizačních hlaviček* (Authentication Header) protokolu IPsec. Nicméně tato specifikace už ale neuvádí bližší informace o tom, jak *Autentizační hlavičky* k tomuto účelu použít. Navíc v důsledku praktických problémů s automatickou správou klíčů jsou současná bezpečnostní řešení omezena pouze na manuální správu klíčů. Problémy především nastávají u ICMP zpráv související s ND. Pro zajištění zabezpečeného spojení je například potřeba zaslat ND zprávy, ty ale není možné zaslat, protože zabezpečení nebylo ještě zajištěno [9].

Díky výše uvedeným skutečnostem vznikl protokol pro *Bezpečné objevování sousedů* (SEcure Neighbor Discovery, SEND), který byl navržen právě k ochraně protokolu ND před bezpečnostními hrozbami. SEND je možné nasadit tam, kde fyzické zabezpečení sítě není možné (bezdrátové sítě) a kde existuje možnost výskytu útoků proti protokolu ND. Protokol definuje množinu nových voleb v již existující zprávách a další dvě nové zprávy pro ND. Jsou to zprávy *Žádost o certifikační cestu* (Certification Path Solicitation, CPS) a *Ohlášení certifikační cesty* (Certification Path Advertisement, CPA). Dále definuje nový mechanismus pro automatickou konfiguraci adres, který se používá ve spojení s již zmíněnými novými volbami ND zpráv. SEND se skládá ze dvou částí, mechanismu pro kryptografické generování adres (Cryptographically Generated Addresses, CGA) a mechanismu pro delegování oprávnění (Authorization Delegation Discovery), které se stará o certifikaci směrovačů [8].

3.1 Kryptograficky generované adresy

Kryptograficky generované adresy (Cryptographically Generated Addresses, CGA) jsou IP adresy generované z otisku veřejného klíče a dalších pomocných parametrů. Poskytuje metodu pro vytvoření zabezpečené vazby mezi veřejným klíčem a IP adresou. Uzel vytvářející CGA musí nejdříve získat pomocí algoritmu RSA pár klíčů (dvojici veřejného a soukromého klíče). Poté spočítá z identifikátoru rozhraní základ adresy a ten spojí s prefixem.

Vytváření CGA je jednorázové. Platnou kryptograficky generovanou adresu není možné ve zprávě podvrhnout nebo její parametry znovu použít. Zpráva totiž musí být podepsána soukromým klíčem korespondujícím s veřejným klíčem, který byl použit při jejím generování. Tento klíč má pouze vlastník adresy. Uzel ani nemůže znovu přeposlat celou SEND zprávu (obsahující CGA, parametry CGA a podpis CGA), protože její podpis má omezenou platnost [2].

3.2 Certifikace směrovačů

Při zavedení automatické konfigurace protokolu *Objevování sousedů* vzniklo bezpečnostní riziko, díky kterému bylo možné na nezabezpečené síti snadno vytvořit podvodný směrovač. Oběť navíc neměla při připojení do sítě informace na to, aby dokázala rozlišit mezi platnými a podvodnými zprávami ze stran směrovačů.

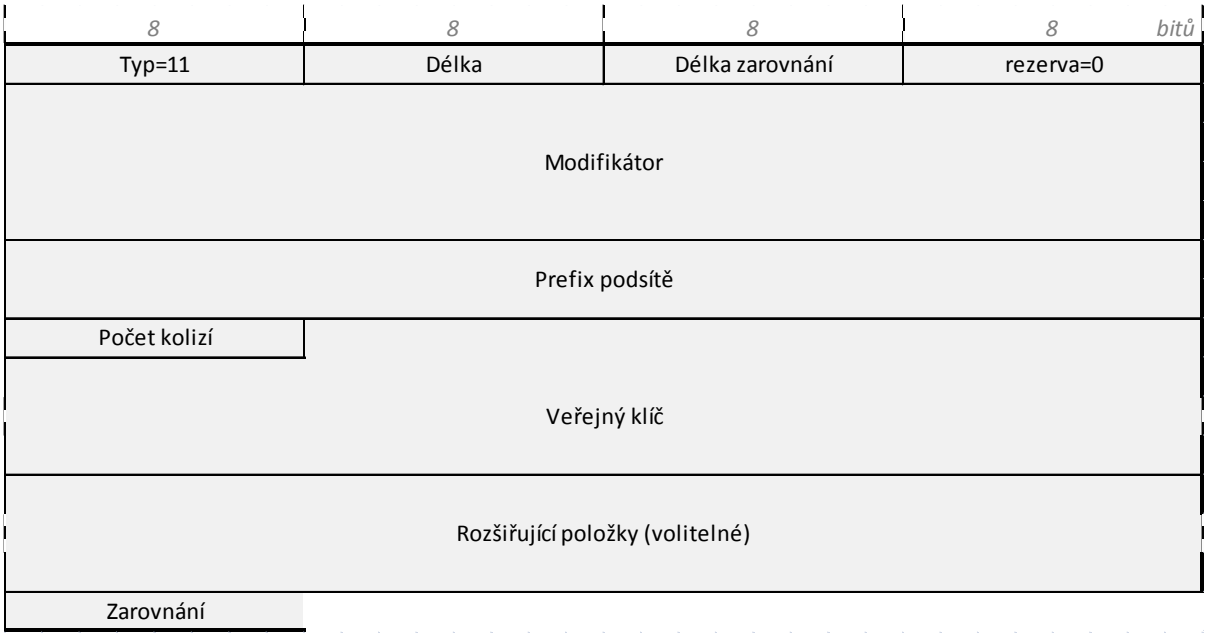
Protokol SEND k zabezpečení ND využívá certifikáty a pro ověřování důvěryhodnosti směrovačů definuje dvě nové ICMP zprávy. Jsou jimi *Žádost o certifikační* cestu a *Ohlášení certifikační cesty*. SEND k ochraně RD požaduje, aby každému směrovači byl udělen certifikát k tomu, aby mohl působit jako směrovač. Ten ho může opravňovat k ohlašování jakýchkoli prefixů nebo jen určité podmnožiny prefixů. Certifikát musí vydat certifikační autorita, které uzly důvěřují (buď přímo anebo v rámci cesty důvěry). Taková certifikační autorita se nazývá *kotva důvěry* (trust anchor). Předpokládá se, že uzly, které této certifikační autoritě důvěřují, vlastní její veřejný klíč. Ten musí získat ještě před tím, než se připojují do sítě. Mohou si ho obstarat například prostřednictvím nějakého jiného komunikačního média nebo správce systému. Když uzel příjme ohlášení od směrovače, kterému prozatím nedůvěřuje, pošle mu *Žádost o certifikační cestu*. Směrovač na tuto žádost odpoví zprávou *Ohlášení certifikační cesty*, která obsahuje certifikáty opravňující ho k ohlašování. Uzel se pokusí na této certifikační cestě nalézt kotvu důvěry a díky veřejnému klíči může ověřit pravost certifikátu. Pokud byla nalezena a pravost odpovídajícího certifikátu úspěšně ověřena, tak uzel může od této chvíle směrovači důvěřovat. Důvěřovat mu bude až do té doby, než vyprší platnost některého z jeho certifikátů.

3.3 Rozšíření ND zpráv

V následujících podkapitolách jsou uvedeny nové volby pro stávající zprávy protokolu ND, které byly přidány v rámci protokolu SEND.

3.3.1 Volba CGA

Jedna z nových voleb pro zprávy protokolu ND je volba CGA. Tato volba umožňuje příjemci ověřit kryptograficky generovanou adresu odesilatele. Formát volby CGA je uveden na obrázku 9.



Obrázek 9: Volba CGA

Volba CGA je identifikována podle hodnoty 11 v položce *Typ*. Za ní následuje *Délka* volby v bytech. Je to souhrnná délka všech položek ve volbě (typ, délka, délka zarovnání, byte rozervaného místa, parametry CGA, zarovnání). *Délka zarovnání* je počet vyplňujících bytů mezi parametry CGA a koncem volby. *Parametry CGA* zahrnují modifikátor, prefix podsítě, počet kolizí, veřejný klíč a volitelné rozšiřující položky. *Modifikátor* je náhodná celočíselná hodnota, která byla použita při výpočtu kryptograficky generované adresy. Tato hodnota dává do vytváření kryptograficky generované adresy prvek náhody. *Počet kolizí* může nabývat hodnot 0, 1 nebo 2. Tato hodnota říká, kolikrát při vytváření kryptograficky generované adresy byly zjištěny duplicitní adresy. Položka *Veřejný klíč* obsahuje veřejný klíč vlastníka adresy.

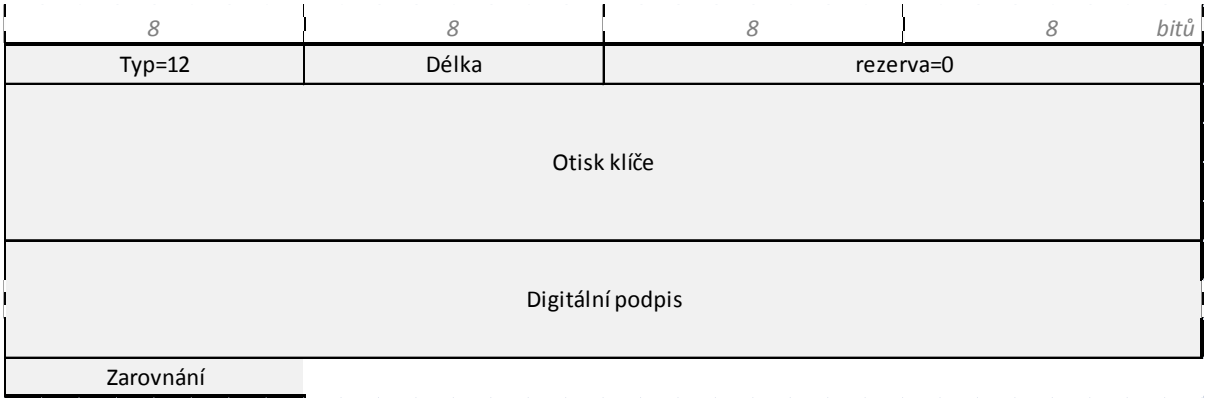
Postup pro výpočet kryptograficky generované adresy je následující [2]:

1. Uložit do modifikátoru (pseudo) náhodnou 128bitovou hodnotu.
2. Vypočíst hash algoritmem SHA-1 ze zřetězení modifikátoru, 9 nulových bajtů, veřejného klíče a případných rozšiřujících položek. Dokud nejlevějších 16xSec bitů obsahuje nenulovou hodnotu, zvětšit modifikátor o jedničku a opakovat. Pokud je Sec=0, tento krok se vynechává.
3. Nastavit počítadlo kolizí na nulu.
4. Zřetězit modifikátor, prefix podsítě, počítadlo kolizí, veřejný klíč a případné rozšiřující položky a vypočítat z této hodnoty SHA-1 hash. Nejlevějších 64 bitů výsledků je označováno jako Hash1 a tvoří základ adresy.

5. Vytvořit z Hash1 identifikátor rozhraní tak, že tři jeho nejlevější bity jsou nahrazeny hodnotou Sec a také do 6. a 7. bitu se uloží hodnoty podle pravidel pro identifikátory rozhraní v IPv6 (příznak globální/lokální a individuální/skupinový).
6. Zřetězením prefixu podsítě a identifikátoru rozhraní vznikne IPv6 adresa.
7. Pokud je požadováno, provést detekci duplicit. Při neúspěchu zvýšit počítadlo kolizí a opakovat postup od kroku 4. Po třetí kolizi zastavit a ohlásit chybu.
8. Vytvořit datovou strukturu podle obrázku 9 a uložit do ní výsledné hodnoty.

3.3.2 Volba RSA podpis

Druhá z nových voleb je *RSA podpis* (RSA Signature). Díky této volbě je možné ND zprávy digitálně podepsat a prokázat tak jejich pravost. Formát volby s RSA podpisem znázorňuje obrázek 10.



Obrázek 10: Volba RSA podpis

Volbu s RSA podpisem lze jednoznačně určit podle hodnoty 12 v položce *Typ*. *Délka* podobně jako u volby CGA udává celkovou délku volby v bytech. Po 16 bitech rezervovaného místa následuje dvojice nejvýznamnějších položek. Jsou jimi *Otisk klíče* (Key hash) a *Digitální podpis* (Digital signature). Položka *Otisk klíče* obsahuje nejlevějších 128 bitů z SHA-1 hashe veřejného klíče použitého při vytváření podpisu. Pomocí otisku klíče lze identifikovat veřejný klíč pro ověření podpisu. Samotný podpis je vytvářen algoritmem RSA s použitím soukromého klíče odesilatele. Podepisuje se zdrojová IP adresa, cílová IP adresa a celá ND zpráva včetně všech voleb předcházejících RSA podpisu (typ, kód a kontrolní součet z ICMP hlavičky, ND hlavička a všechny volby nacházející se před volbou pro podpis).

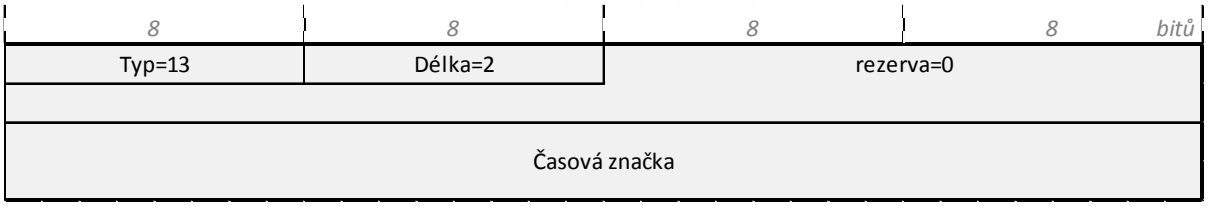
Pokud je odesílající uzel nakonfigurován, aby používal SEND, musí podepisovat (musí obsahovat volbu RSA podpis) v zásadě všechny ND zprávy, jmenovitě *Výzvu sousedovi*, *Ohlášení souseda*, *Ohlášení směrovače* a *Přesměrování*. *Výzvu směrovači* nemusí odesilatel podepisovat pouze tehdy, nemá-li určenou zdrojovou adresu. Odesilatel při tvorbě ND zprávy vkládá volbu *RSA podpis* až úplně nakonec.

ND zprávy, které nedisponují podpisem podle výše uvedených pravidel, jsou považovány za nezabezpečené. To znamená, že příjemce s nimi musí nakládat jako se zprávami zaslanými odesilatelem nepodporující SEND. Při zpracování přijaté zprávy musí příjemce ignorovat všechny další volby vložené za *RSA podpis*. Otisk klíče musí odpovídat známému veřejnému klíči. Veřejný klíč se může nacházet ve volbě CGA aktuální zprávy nebo mohl být získán i jiným způsobem dříve. Pokud podpis poruší některou z výše uvedených podmínek, tak je zpráva považována za nezabezpečenou. Jestliže je příjemce nastaven tak, aby přijímal pouze zabezpečené zprávy, bude zpráva zahozena. Příjemce, který je konfigurován tak, aby přijímal zabezpečené i nezabezpečené zprávy, ji může přijmout. Ovšem s touto zprávou bude nakládat, jako by byla nezabezpečená.

Vytváření a ověřování RSA podpisu je výpočetně náročné. Nicméně koncové uzly musejí provést pouze několik podpisových operací při přihlášení do sítě, při zjišťování linkové adresy souseda nebo při ověřování dosažitelnosti souseda. Naproti tomu směrovače musí těchto operací provádět daleko více. I přesto se jedná o několik desítek operací za sekundu. Navíc pro některá plánovaná skupinová ohlášení (*Ohlášení směrovače*, *Ohlášení souseda*) je možné RSA podpis předpočítat dopředu.

3.3.3 Volba Časová značka

Účelem volby *Časová značka* (Timestamp) je ochrana proti přeposílání *Ohlášení směrovače*, *Ohlášení souseda* a *Přesměrování*. Formát volby *Časová značka* je ilustrován na obrázku 11.



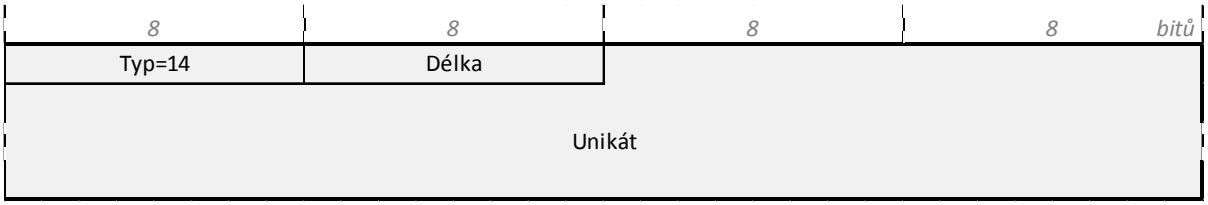
Obrázek 11: Volba Časová značka

Volbu *Časová značka* identifikuje hodnota 13 v položce *Typ*. Následuje *Délka* volby, která je udávána v bytech. Po délce se zde nachází 6 bytů vyhrazených pro budoucí použití. Samotná položka s časovou značkou udává počet sekund od 1. ledna 1970, 00:00 UTC.

Uzly používající SEND musejí vkládat volbu *Časová značka* do všech ND a SEND zpráv. Zprávy bez jediné této volby jsou považovány za nezabezpečené. Každá zpráva s volbou *RSA podpis* musí obsahovat i volbu *Časová značka*.

3.3.4 Volba Unikát

Poslední přidanou volbou v rámci protokolu SEND je *Unikát* (Nonce). Ten má zaručit, že ohlášení (*Ohlášení směrovače*, *Ohlášení souseda*) je čerstvá odpověď na výzvu zaslanou dříve. Formát volby *Unikát* je uveden na obrázku 12.



Obrázek 12: Volba Unikát

Volbu *Unikát* lze jednoznačně určit podle hodnoty 14 v položce *Typ*. Velikost celé volby v bytech se nachází v položce *Délka*. *Unikát* je (pseudo) náhodná hodnota zaslaná odesilatelem výzvy. Velikost této hodnoty musí být nejméně 6 bytů a musí být zarovnána na celé byty.

Jestliže je uzel nakonfigurován tak, aby používal SEND, všechny jeho výzvy musí obsahovat tuto volbu. Při odesílání výzvy si uzel musí zapamatovat hodnotu, kterou vložil do volby *Unikát*, aby ji mohl porovnat s odpovídající odpovědí. Výzvy s volbou RSA podpis musejí disponovat i volbou *Unikát*. Cílená oznámení s volbou RSA podpis, ale bez volby Unikát jsou zpracována jako nevyžádaná (nebyla na ně odeslána výzva).

3.4 Žádost o certifikační cestu

Koncové uzly zasílají směrovačům ICMP zprávu *Žádost o certifikační cestu* proto, aby jim odpověděli zprávou *Ohlášení certifikační cesty*. Formát *Žádosti o certifikační cestu* je znázorněn na obrázku 13.

Obrázek 13: Žádost o certifikační cestu

Hodnota v položce *Identifikátor* slouží ke spárování odpovídajících žádostí a oznámení. Tato hodnota nesmí být nulová a měla by být pseudonáhodně generována. *Komponenta* je identifikátor certifikátu v certifikační cestě, který si přeje odesilatel získat. Pokud odesilatel do tohoto políčka vloží hodnotu 65535, tak tím říká, že si přeje získat všechny certifikáty. Ke zprávě je možné připojit volbu *Kotva důvěry*, ve které odesilatel udává kotvu důvěry, pro kterou si přeje obdržet certifikační cestu.

3.5 Ohlášení certifikační cesty

Směrovače posílají ICMP zprávu *Ohlášení certifikační cesty* jako odpověď na zprávu *Žádost o certifikační cestu*. Formát *Ohlášení certifikační cesty* ilustruje obrázek 14.

Obrázek 14: Ohlášení certifikační cesty

Identifikátor se používá ke spárování odpovídajících žádostí a oznámení, viz zpráva *Žádost o certifikační cestu*. Hodnota v položce *Všechny komponenty* příjemce informuje o počtu certifikátů v certifikační cestě. Hodnota v položce *Komponenta* říká, který certifikát je právě posílán. Zprávu je možné rozšířit o jednu nebo více voleb *Kotva důvěry*. Více těchto voleb dává příjemci možnost si určit, která oznámení jsou pro něj užitečná. Asi nejvýznamnější je volba *Certifikát*, která se může vyskytovat v ohlášení také více jak jednou. Každý zasílaný certifikát je uložen v samostatné volbě. Pokud by ale jedno ohlášení mělo obsahovat více certifikátů, je vhodné jej rozdělit, aby nedocházelo k jeho nadměrné fragmentaci na IP vrstvě. Jestliže směrovač není schopen nalézt žádný certifikát pro požadovanou kotvu důvěry, musí odpovědět ohlášením bez certifikátu a měl by požadovanou kotvu důvěry uvést ve volbě *Kotva důvěry*.

Kombinace kryptograficky generovaných adres, digitálních podpisů a certifikace směrovačů by měla ochránit Objevování sousedů proti téměř všem známým útokům [2]. SEND ale nezabezpečuje důvěrnost ND zpráv a neposkytuje zabezpečenou vazbu mezi linkovou a IP adresou. Na nezabezpečené linkové vrstvě proto vzniká hrozba útoku typu *Odepření služby* [8].

3.6 Současné implementace

V následujících podkapitolách je seznámení se současnými implementacemi protokolu SEND pro operační systémy GNU/Linux.

3.6.1 DoCoMo SEND

DoCoMo SEND je implementace protokolu SEcure Neighbor Discovery (SEND) pracující zcela v uživatelském režimu (user space) na operačních systémech Linux a FreeBSD vyvíjená společností NTT DoCoMo. Navíc obsahuje knihovny s funkcemi pro vytváření a ověřování kryptograficky generovaných adres (CGA) a rozšíření certifikátů standardu X.509 o podporu certifikace IP adres.

Hlavním úkolem tohoto díla bylo vytvořit přenositelnou a snadno použitelnou aplikaci protokolu SEND. Díky tomu, že aplikace běží výhradně v uživatelském režimu, nebylo nutné modifikovat jádro operačního systému a ani jiné aplikace.

DoCoMo SEND pracuje na bázi filtrování síťového provozu, kde všechny příchozí a odchozí zprávy protokolu ND jsou přeposílány do uživatelského paměťového prostoru ke zpracování aplikací.

Ta k odchozím ND zprávám přidává a u příchozích SEND zpráv ověřuje rozšíření protokolu SEND. Zprávy, které byly ověřeny, jsou zasílány zpět do jádra operačního systému k dalšímu zpracování. Naproti tomu zprávy, které neprošly ověřováním, jsou aplikací zahazovány.

DoCoMo SEND je prototyp implementace, kde je kladen důraz na správnost implementace protokolu SEND. Tvůrci ovšem uvádějí, že implementace může obsahovat jiné chyby a nelze očekávat spolehlivost a bezpečnost komerční aplikace [10].

Aplikace k zachytávání zpráv a jejich transportu mezi uživatelským paměťovým prostorem a paměťovým prostorem jádra používá rozhraní Berkley Packet Filter (BPF) podsystému netgraph (síťový podsystém operačního systému FreeBSD). Tento přístup má dvě zásadní nevýhody. Aplikace je závislá na podsystému *netgraph*, který je dostupný pouze na operačních systémech FreeBSD a DragonFlyBSD , navíc veškerý síťový provoz musí procházet přes rozhraní BPF, což má negativní dopad na propustnost systému a znemožňuje tak nasazení aplikace na vysokorychlostních sítích [12].

Implementace je napsána v programovacím jazyce C a poslední známá verze je 0.2 z dubna roku 2006 [11].

3.6.2 Native SeND kernel API for *BSD

Vývoj Native SeND kernel API for *BSD byl založen na implementaci DoCoMo SEND s cílem odstranit některé nevýhody uvedené v předešlé kapitole. Aplikace se skládá z uživatelské části vycházející z DoCoMo SEND a modulu jádra operačního systému send.ko, který tvoří bránu mezi síťovým podsystémem a aplikací. Byla odstraněna závislost na podsystému *netgraph* a tím i již zmíněná výkonnostní omezení.

Implementace se nachází ve stádiu prototyp a je napsána v programovacím jazyce C. Poslední známá aktualizace je ze dne 12.8.2010 [13].

3.6.3 Easy-SEND

Implementace s názvem Easy-SEND vznikla v rámci školního projektu na Universidad Central de Venezuela Facultad de Ciencias Escuela de Computación Laboratorio de Comunicación y Redes. Výsledná aplikace byla napsána v jazyce Java a je určena zejména pro výukové účely a účely testování. Aplikace obsahuje samostatný nástroj pro vytváření a ověřování Kryptograficky generovaných adres (CGAGen). Poslední modifikace projektu byly uskutečněny dne 8.4.2009 [14].

3.6.4 IPV6 SEND CGA

Hlavním cílem implementace s názvem IPv6 SEND CGA bylo odhalit chyby protokolu SEND. Aplikace se skládá z modifikovaného modulu jádra operačního systému (ipv6 module) a procesu běžícího v uživatelském režimu (SEND Daemon). Prostřednictvím modulu jádra je možné přímo přistupovat do směrovací tabulky a do cache sousedů. Naproti tomu většina operací typu šifrování a

dešifrování, jako jsou vytváření a ověřování CGA, vytváření a ověřování podpisu a práce s certifikáty, probíhá v procesu SEND Daemon. Implementace je napsána v jazyce C a je vyvíjena a udržována společností Huawei Technologies Corp. a univerzitou Beijing University of Post and Telecommunications.

Implementace se nachází ve stádiu výzkumný prototyp a podle autorů může obsahovat chyby, které mohou způsobit pád jádra operačního systému. Poslední známá aktualizace je z prosince roku 2009 [15].

3.6.5 NDprotector

NDprotector je prototyp implementace protokolu SEND pro operační systémy Linux. Tato implementace nemá přímý přístup do jádra operačního systému a díky tomu nemůže případně způsobit jeho selhání. Nicméně tato skutečnost vyžaduje emulaci některých datových struktur, které jsou dostupné pouze v rámci jádra. Aplikace nemá prozatím podporu Kryptograficky generovaných adres, ta je plánována v následujících verzí aplikace.

Implementace je napsána ve skriptovacím jazyce Python a poslední známá verze je 0.5 z roku 2009 [16].

3.6.6 TrustRouter

TrustRouter je částečná implementace protokolu SEND pro operační systémy Linux, Windows a Mac OS X. Tato implementace prozatím neobsahuje podporu pro *Kryptograficky generované adresy*. Při vývoji byl kladen důraz na možnost snadné správy aplikace. Prozatím byla implementována jen klientská část pro ověřování certifikace směrovačů (získávání certifikační cesty, ověřování podpisu, ověřování certifikátu směrovače). Druhá část aplikace - aplikace pro směrovače(přenos certifikátů, podepisování zpráv) je plánována jako součást programu pro rozesílání *Ohlášení směrovače* (radvd). Aplikace je napsána částečně v programovacím jazyce C a částečně ve skriptovacím jazyce Python3. Poslední její známá verze je v1.1 ze dne 18. dubna 2012 [17].

3.6.7 Jiné implementace

V současné době existují implementace protokolu SEND i pro jiné operační systémy. WinSEND je implementace protokolu SEND pro OS Windows. Pro zachytávání ND zpráv využívá knihovnu WinPcap. Nicméně tato implementace není prozatím dostupná pro veřejnost [18]. Na zařízeních firmy Cisco s operačním systémem IOS od verze 12.4(24)T v distribuci Advanced Enterprise Services je k dispozici proprietární implementace.

4 Návrh

V této kapitole je diskutován návrh aplikace pro podporu protokolu SEND. Jsou zde uvedeny různé možné problémy spojené s tvorbou aplikace a možné přístupy, jakými lze tyto problémy řešit. V závěru kapitoly je prezentován mnou zvolený postup.

Protokol ND je implementován zcela v jádře operačního systému a za normálních okolností k nim z uživatelského prostoru nemáme přístup. Jestliže tyto zprávy potřebujeme modifikovat, musíme buď protokol SEND implementovat přímo v jádře, upravit stávající implementaci protokolu ND nebo nalézt způsob, jakým tyto zprávy přesměrovat do uživatelského prostoru, kde by mohly být dále zpracovány. V uživatelském prostoru je sice možné vytvořit soket, který bude naslouchat zprávám protokolu ND. Nicméně tímto způsobem nelze zprávy před příchodem do jádra operačního systému modifikovat, protože jsou současně doručovány do jádra v původní nezměněné podobě [19].

Implementace protokolu SEND čistě uvnitř jádra by podle mého názoru nebyla vhodná, protože tento přístup má řadu nevýhod. V případě chyby nebo selhání by tímto mohla být negativně ovlivněna stabilita celého systému. Další nevýhodou jsou omezení na dostupné funkce a knihovny. Navíc by instalace této implementace vyžadovala rekompilaci jádra, a to by v mnoha případech bránilo praktickému použití.

Druhou možností je tedy vytvořit v jádře pouze pomocnou část aplikace - hák, který by sloužil jen k odchytávání ND zpráv a k jejich ukládání do uživatelského paměťového prostoru. Druhá část aplikace implementující protokol SEND a běžící v uživatelském prostoru by si takto uložené zprávy vyzvedávala a po jejich zpracování by je předávala zpět do jádra skrze již zmíněný sdílený paměťový prostor. V jádře by pak pokračovalo jejich zpracování v rámci původní implementace protokolu ND. Takto by byly odstraněny některé nevýhody předešlé varianty, stále by ale bylo nutné zasáhnout do zdrojových kódů jádra a při instalaci aplikace jádro rekompilovat.

Pokud tedy požadujeme, aby naše implementace byla plně nezávislá na modifikaci jádra, bylo nutné najít jiný způsob, jakým tyto zprávy přesměrovat z jádra do uživatelského prostoru. Možností je namísto přímé úpravy jádra vytvořit modul jádra (loadable kernel module - LKM), který by plnil funkci háku z předchozí varianty. Instalace by pak spočívala v zavedení takovéhoto modulu za běhu systému do jádra. Hák by mohl být realizován substitucí existujících funkcí pro zpracování ND zpráv modifikovanými verzemi. Ty by obsahovaly navíc kód, který by ukládal zprávy do sdílené paměti s uživatelským prostorem pro následné zpracování. Jakmile by byly zpracovány, hák by je předal zpět původní funkci. Samotná substituce by mohla probíhat tak, že během zavedení modulu jádra by se hodnota adresy původní funkce uložila a nahradila adresou nové funkce. Při uvolnění modulu by se adresa změnila zpět na adresu původní funkce. Pro tuto substituci by se mohly vybrat konkrétně funkce *icmpv6*_rcv (net/ipv6/icmp.c) nebo *ndisc_rcv* (net/ipv6/ndisc.c). Funkce *icmpv6*_rcv slouží zejména k rozlišení jednotlivých ICMPv6 zpráv a k jejich následnému zaslání do náležitých

podsystémů. Tato funkce se volá pro každý příchozí ICMPv6 paket a bylo by nutné uvnitř vytvořit navíc filtr zpráv protokolu ND. Proto je lepší variantou funkce *ndisc*_rcv, která se volá pouze pro ND zprávy. U substituce je třeba sledovat změny v kódu uvnitř těchto funkcí a reflektovat je do modifikovaných funkcí modulu. Pseudokód v jazyce C pro substituci funkcí by mohl vypadat takto:

```
static void (*original_function_ref)(); /* Odkaz na nahrazovanou funkci */
extern void (*internal_function)();     /* Nahrazovaná funkce */

static void new_function()
{
  /* Nová implementace funkce */
}

int init_module()
{
  original_function_ref = internal_function;
  internal_function     = &new_function;
  return 0;
}

void cleanup_module()
{
  internal_function = original_function_ref;
}
```

Zmíněné funkce ale není možné nahradit přímo, protože jejich symboly nejsou exportovány v tabulce symbolů, a proto adresy těchto funkcí nejsou globálně přístupné. Exportování těchto symbolů by muselo být provedeno ve zdrojových kódech jádra a vedlo by opět k nutnosti jádro znovu přeložit. Částečným řešením by mohly být *kprobes* [20], které slouží ke vkládání návěstí do funkcí jádra. Jakmile proces jádra dosáhne návěstí, je zavolána předem definovaná funkce. Nevýhodou však je, že *krobes* nejsou v mnoha případech povoleny a jejich povolení by vyžadovalo rekompilaci jádra.

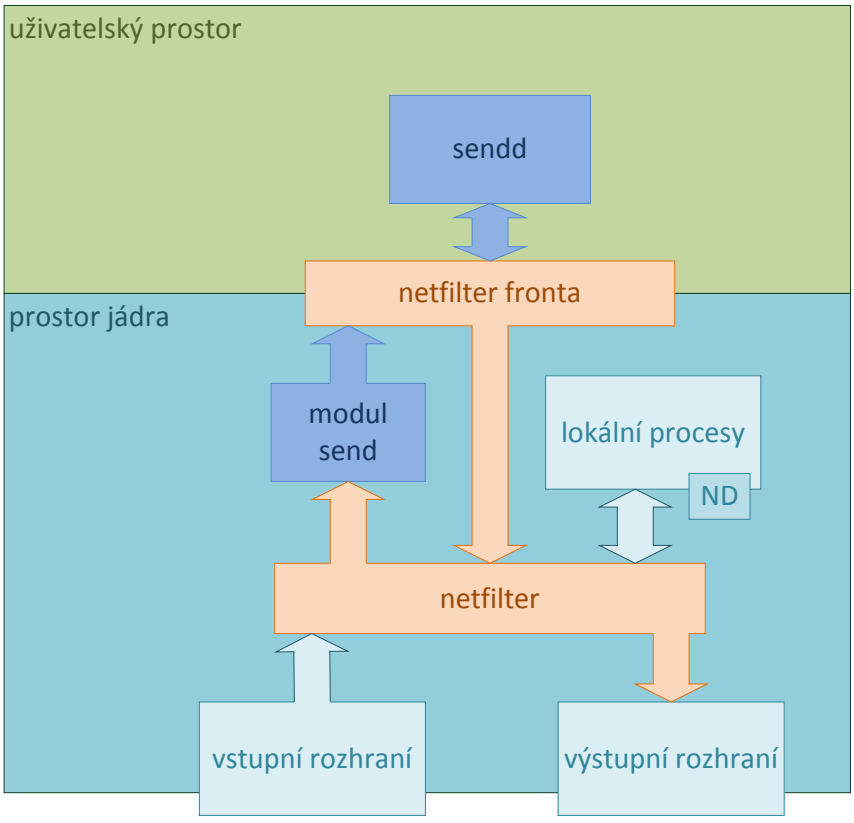
Od verze 2.4 je k dispozici uvnitř jádra architektura *netfilter* [21]. Pomocí ní lze zachytávat pakety ještě dříve, než jsou doručeny procesům protokolu ND. Konečnou alternativou je tedy aplikaci rozdělit na dvě části. První částí aplikace bude pomocný modul, ve kterém bude probíhat přesměrování ND zpráv z jádra do uživatelského prostoru a to prostřednictvím fronty systému *netfilter*. Druhou částí je implementace samotného protokolu SEND běžící čistě v uživatelském prostoru.

Při implementaci uživatelské části aplikace pro podporu SEND máme dvě možnosti, buď vytvoříme zcela novou implementaci, nebo jako základ použijeme už některou existující. Dle mého názoru není nutné znovu vytvářet naprosto novou aplikaci, protože existuje řada nedokončených projektů obsahující spoustu užitečného zdrojového kódu a bylo by možné je rozšířit a uvést do prakticky použitelného stavu. Navíc bych se mohl dostat do stavu, kde bych řešil problémy, které už jednou řešeny byly. To by z časového hlediska nebylo efektivní. Při výběru základní implementace jsem se rozhodoval zejména podle toho, v jakém jazyce je napsána a jak velkou část specifikace

protokolu SEND splňuje. Preferoval jsem jazyk C nad jazyky Java a Python, protože se domnívám, že výsledná aplikace tak bude efektivněji využívat výkon platformy. Navíc nebude závislá na interpretu jazyka a jeho modulech.

Rozhodl jsem se tedy rozšířit stávající implementaci DoCoMo SEND, protože kromě faktu, že je napsána v jazyce C, pokrývá i největší část specifikace protokolu SEND. Kromě toho se autoři implementace aktivně podíleli na tvorbě této specifikace, a proto je výsledný kód v tomto směru robustní. Nicméně původní zdrojové soubory již nejsou z oficiálních zdrojů dostupné [22]. Jedinou možností bylo využít zdrojové soubory z projektu Native SeND kernel API for *BSD. Zde byla upravená aplikace DoCoMo SEND použita v uživatelské části, jak už bylo zmíněno v kapitole věnující se existujícím implementacím. Kód ovšem obsahuje rozsáhlé modifikace specifické pro platformu BSD a bude třeba jej nejprve uvést do původního stavu. Protože zdrojové soubory navíc nebyly delší dobu aktualizovány, bude dále nutné nahradit některé použité funkce a knihovny novějšími verzemi. V neposlední řadě bude odstraněno co možná nejvíce nalezených nedostatků původní implementace.

Část implementující SEND bude tedy realizována jako démon běžící v uživatelském prostoru. Přebírání zpráv protokolu ND z *netfilter* fronty se uskuteční s pomocí knihovny *libnetfilter_queue*. Pro manipulaci s certifikáty a kryptografické operace se využije knihovna *libcrypto*, která je součástí OpenSSL [23]. Ovládání síťového podsystému a rozhraní se bude provádět prostřednictvím protokolu *netlink*. Návrh je graficky znázorněn na obrázku 15.



Obrázek 15: Architektura sendd

5 Implementace

Jak bylo uvedeno již v návrhu, tak vlastní protokol SEND je implementován v podobě démona běžícího v uživatelském prostředí. Aplikace byla napsána v jazyce C a vychází z projektu DoCoMo SEND. Dále zde byly použity zejména knihovny *libcrypto*, *libnetfilter*_queue a protokol *netlink*. Tato kapitola je rozdělena do několika částí. V první části se nachází seznámení s technologiemi, které byly použity při implementaci. Ve druhé části je uveden stručný popis původní implementace *sendd0.2* včetně popisu provedených úprav. Třetí část je věnována podrobnějšímu popisu rozsáhlejších modifikací a rozšíření.

5.1 Použité technologie

V této kapitole jsou uvedeny technologie a prostředky zmíněné během návrhu a použité při implementaci.

5.1.1 Netfilter a libnetilter_queue

Systém *netfilter* [21] slouží k filtrování paketů, umožňuje vytváření NAT a PAT a poskytuje prostředky i pro jinou manipulaci s pakety. Je součástí jádra operačního systému Linux od verze 2.4. Jeho architektura se skládá ze sady háků uvnitř síťového podsystému a označují se takto:

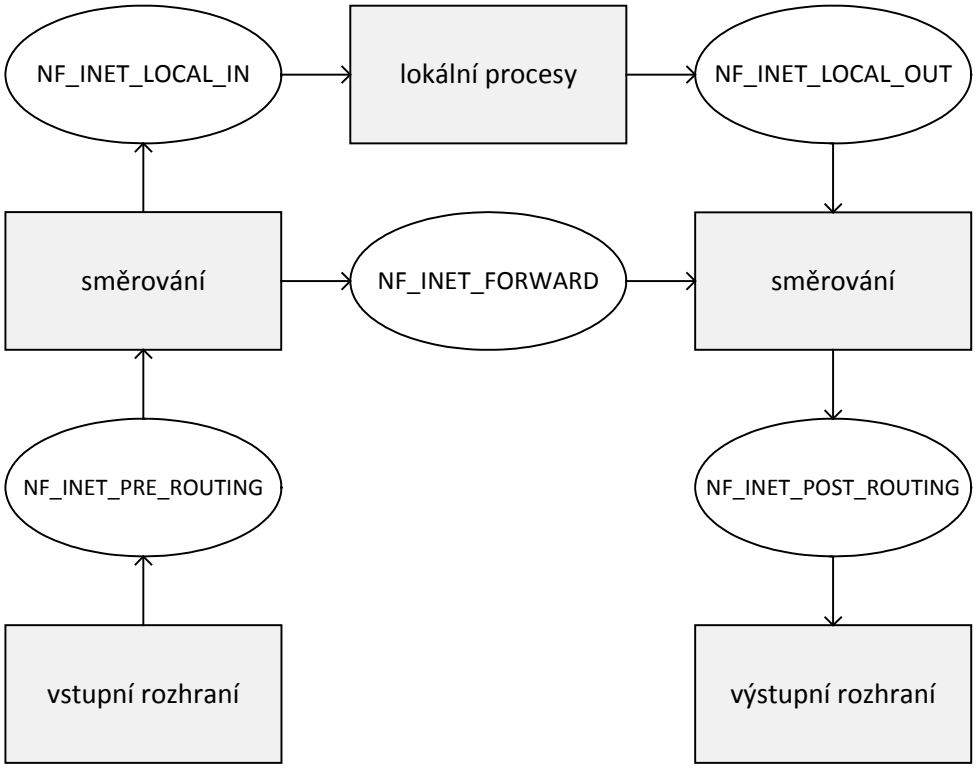
- NF_INET_PRE_ROUTING
- NF_INET_LOCAL_IN
- NF_INET_FORWARD
- NF_INET_LOCAL_OUT
- NF_INET_POST_ROUTING

Jejich rozmístění je znázorněno na obrázku 16. K jednotlivým hákům je možné přiřadit funkci, která se provede ihned, jakmile k nim dorazí nějaký paket. Tyto funkce se zpravidla implementují uvnitř modulů jádra a nazývají se *netfilter* moduly. K háku lze přiřadit i více takových funkcí, ty jsou pak vykonávány sekvenčně. Funkce musí v neposlední řadě určit, co se s daným paketem provede. Možnosti jsou následující:

- NF_DROP
- NF_ACCEPT
- NF_STOLEN
- NF_QUEUE
- NF_REPEAT

NF_DROP značí, že paket se má zahodit. Naproti tomu NF_ACCEPT říká, že má pokračovat dál síťovým řetězcem. Volbou NF_STOLEN *netfilter* informujeme, že jsme nad paketem převzali plnou

kontrolu a že se o něj nemáme dále starat. Pokud je zapotřebí, aby byl paket nejdříve zpracován procesem v uživatelském prostředí, je zvolena možnost NF_QUEUE. Paket je tak uložen do fronty, kde si jej může zmíněný proces vyzvednout. Po zpracování musí proces opět určit, jak se s paketem naloží. V systému *netfilter* je možné vytvořit až 65536 samostatných front (číslované od 0). Výchozí číslo fronty je 0, pokud si přejeme vložit paket do jiné fronty, tak je zde k dispozici makro NF_QUEUE_NR(x), kde x označuje číslo námi vybrané fronty. Fronty mají omezenou délku , pokud výchozí fronta s číslem 0 bude zaplněna, tak další příchozí pakety budou zahazovány. NF_REPEAT udává, že paket musí projít znovu všemi funkcemi, které jsou k danému háku přiřazeny.



Obrázek 16: Architektura netfilter

Paket, který právě dorazil na síťové rozhraní, projde nejprve hákem s názvem NF_INET_PRE_ROUTING. Poté prochází procesem směrování, kde se určuje, zda je paket adresován nám, nebo zda má být přeposlán dál. Jestliže je paket určen nám, tak před tím, než je předán lokálním procesům, musí projít hákem NF_INET_LOCAL_IN. Pokud má být doručen někomu jinému, směřuje do NF_INET_FORWARD. Všechny odchozí pakety včetně těch, které mají být přeposlány dál, prochází opět procesem směrování. Dříve, než jsou předány síťovému rozhraní pro odeslaní, projdou hákem s označením NF_INET_POST_ROUTING.

Netfilter je nástupce systémů *ipfwadm* (Linux 2.0) a *ipchains* (Linux 2.2). K jeho konfiguraci se používá program *iptables* (ip6tables pro protokol IPv6).

Libnetfilter_queue je knihovna pro aplikace běžící v uživatelském prostoru a slouží k práci s pakety v *netfilter* frontě. Obsahuje funkce, které umožní frontu otevřít tak, že ke vloženým paketům

je pak možné přistupovat skrze standardní funkci *recv*. Data obdržených paketů jsou uloženy ve struktuře *nfq_data*, k těm se dále přistupuje prostřednictvím specifických funkcí. V neposlední řadě je součástí knihovny funkce, díky které lze nad pakety vynést rozsudek, co se nimi provede dál, viz výše. Rozsudek je nutné vynést pro každý paket ve frontě. Pokud se tak nestane, zůstávají ve frontě. A ty, jak již bylo řečeno, mají omezenou kapacitu. *Libnetfilter_queue* nahrazuje zastaralou knihovnu *libipq*.

5.1.2 OpenSSL a libcrypto

OpenSSL [23] je sada nástrojů pro podporu protokolů pro zabezpečené sokety Secure Sockets Layer (SSL) a jeho nástupce Transport Layer Security (TLS). Součástí jsou i kryptografická knihovna *libcrypto* a knihovna *ssl* implementující protokoly SSL (v2 a v3) a TLS v1.

Knihovna *libcrypto* obsahuje širokou škálu kryptografických algoritmů a může být použita k implementaci různých kryptografických standardů. Je rozdělena do několika částí podle toho, jaké algoritmy konkrétně implementují. Jsou zde k dispozici symetrické šifry, zejména *blowfish*, *des*, *idea*, *rc4*. Knihovna dále poskytuje některé asymetrické šifrovací algoritmy jako jsou *dsa* a *rsa*. Můžeme zde nalézt i algoritmus *dh* (Diffie–Hellman) pro bezpečnou výměnu klíčů. V neposlední řadě libcrypto obsahuje i funkce pro manipulaci s certifikáty x509 a hashovací funkce, zejména *md5* a *sha*.

5.1.3 Netlink a rtnetlink

Netlink [24] je protokol pro přenos informací mezi jádrem a procesy v uživatelském prostoru. Tyto procesy mohou komunikovat protokolem *netlink* prostřednictvím standardních soketů. Moduly jádra na druhé straně mají k dispozici specifické API.

Zprávy protokolu *netlink* se skládají z jedné nebo více *nlmsghdr* hlaviček. Po každé hlavičce následuje blok souvisejících dat. Struktura hlavičky vypadá následovně:

```
struct nlmsghdr {
  __u32 nlmsg_len;    /* Délka zprávy včetně hlavičky */
  __u16 nlmsg_type;   /* Typ obsahu zprávy */
  __u16 nlmsg_flags;  /* Dodatečné příznaky */
  __u32 nlmsg_seq;    /* Pořadové číslo */
  __u32 nlmsg_pid;    /* ID portu odesilatele */
};
```

Standardní typy obsahu zprávy jsou NLMSG_NOOP, NLMSG_DONE a NLMSG_ERROR. NLMSG_NOOP říká, že obsah má být ignorován. Hlavička s typem NLMSG_NOOP zakončuje zprávu složenou z více částí. Zprávy typu NLMSG_ERROR signalizují chybu a obsahují strukturu *nlmsgerr*, která vypadá takto:

```
struct nlmsgerr {
  int error;         /* Negativní kód chyby errno, nebo 0 pro potvrzení */
  struct nlmsghdr msg; /* Hlavička, která chybu způsobila */
};
```

Netlink specifikuje i další typy zpráv, v rámci implementace naší aplikace pro podporu SEND nás ale budou nejvíce zajímat zprávy protokolu *rtnetlink*. *Rtnetlink* je součástí protokolu *netlink* a bude blíže popsán níže. Standardní příznaky, které se mohou vyskytovat v položce *nlmsg_flags*, jsou následující:

- NLM_F_REQUEST: Tento příznak musí být nastaven ve všech zprávách, které požadují nějaká data.
- NLM_F_MULTI: Pokud je zpráva složena z více částí, je tento příznak nastaven ve všech přítomných hlavičkách a poslední zpráva je typu NLMSG_DONE.
- NLM_F_ACK: Zpráva s tímto příznakem vyžaduje potvrzení o doručení.
- NLM_F_ECHO: Jestliže je tento příznak nastaven, zpráva je beze změn zaslána zpět odesilateli.

Ve zprávách, které žádají příjemce o zaslání nějakých dat, se společně s NLM_F_REQUEST mohou nastavit tyto doplňující příznaky:

- NLM_F_ROOT: Tímto příznakem říkáme, že požadujeme celou tabulku datových objektů.
- NLM_F_MATCH: Vrací všechny datové objekty splňující podmínky definované v těle zprávy. Tento příznak prozatím není implementován.
- NLM_F_ATOMIC: Pokud vyžadujeme zaslání atomického snímku tabulky datových objektů, nastavíme tento příznak.
- NLM_F_DUMP: Tento příznak odpovídá spojení NLM_F_ROOT|NLM_F_MATCH.

Ve zprávách, které žádají o vytvoření nějakého datového objektu na straně příjemce, se společně s NLM_F_REQUEST mohou nastavit i příznaky následující:

- NLM_F_REPLACE: Nahradí již existující datový objekt.
- NLM_F_EXCL: Nenahrazovat, jestliže datový objekt již existuje.
- NLM_F_CREATE: Vytvořit datový objekt, pokud neexistuje.
- NLM_F_APPEND: Přidat datový objekt na konec seznamu.

Pod pojmem datový objekt si lze představit například IP adresu síťového rozhraní. Tabulka takovýchto objektů je pak seznam adres přiřazených k danému rozhraní. Konkrétní příklad je uveden později v popisu implementace.

Hodnota v položce *nlmsg_seq* udává pořadové číslo zprávy a slouží ke sledování jejich doručování. Během přenosu může dojít k nedostatku paměti nebo k jiným chybám a zpráva může být zahozena, a proto, i když se protokol snaží o doručení každé zprávy, není úplně spolehlivý. Pro zvýšení spolehlivosti může odesilatel přidat do hlavičky zprávy příznak NLM_F_ACK, tím požaduje potvrzení o jejím doručení. Potvrzení je ve formě zprávy typu NLMSG_ERROR, která bude v položce *error* obsahovat hodnotu 0. Aplikace se musí sama postarat o vytváření takovýchto potvrzení.

Hodnota v položce *nlmsg_pid* slouží k určení soketu odesilatele zprávy. Pokud ji odesilatel nespecifikuje (vloží hodnotu 0), tak mu bude přidělena jádrem. Hodnota odpovídá obvykle

identifikátoru procesu PID. Jestliže má ale daný proces otevřených více soketů, tak mu jádro určí jinou unikátní hodnotu. Aplikace si ji může zvolit, nicméně musí zajistit, aby byla unikátní.
Protokol *rtnetlink* [25], který je součástí protokolu *netlink*, definuje množství dalších zpráv. Tyto zprávy se používají především ke čtení a k úpravám směrovacích tabulek, IP adres a parametrů síťových rozhraní. Dále lze s jejich pomocí měnit chování paketového plánovače a modifikovat třídění síťového provozu. Ke každé uvedené zprávě lze přidat atributy, které mají následující strukturu:

```
struct rtattr {
  unsigned short rta_len;    /* Délka atributu */
  unsigned short rta_type;   /* Typ atributu */
  /* Vlastní data atributu */
};
```

Typ atributu je specifický pro každý typ zprávy. Z pohledu naší konkrétní implementace nás budou nejvíce zajímat zprávy, které se týkají konfigurace síťových rozhraní a IP adres. Zprávy jsou popsány dále.

Zprávy typu RTM_NEWLINK, RTM_DELLINK a RTM_GETLINK jsou určeny k vytváření, mazání a získávání parametrů síťových rozhraní. Tyto zprávy obsahují strukturu *ifinfomsg*, která vypadá následovně:

```
struct ifinfomsg {
  unsigned char  ifi_family; /* Rodina protokolů */
  unsigned short ifi_type;   /* Typ zařízení */
  int            ifi_index;  /* Index rozhraní */
  unsigned int   ifi_flags;  /* Příznaky */
  unsigned int   ifi_change; /* Maska změn */
};
```

Položka *ifi_family* říká, jakou rodinu protokolů rozhraní podporuje. Používá se zde hodnota AF_UNSPEC. Index rozhraní je unikátní nenulové číslo identifikující síťové rozhraní. Položka s maskou změn je rezervována pro pozdější použití a měla by být vždy nastavena na hodnotu 0xFFFFFFFF. Zmíněné zprávy mohou dále obsahovat tyto typy atributů:

- IFLA_UNSPEC: nespecifikovaný atribut
- IFLA_ADDRESS: linková adresa rozhraní
- IFLA_BROADCAST: všesměrová linková adresa
- IFLA_IFNAME: jméno rozhraní
- IFLA_MT: MTU rozhraní
- IFLA_LINK: typ linky
- IFLA_QDISC: paketové plánování
- IFLA_STATS: statistiky rozhraní

Pomocí zpráv typu RTM_NEWADDR, RTM_DELADDR, RTM_GETADDR lze přidělovat, odebírat a číst IP adresy síťového rozhraní. Tyto zprávy obsahují strukturu *ifaddrmsg*:

```
struct ifaddrmsg {
  unsigned char ifa_family;    /* Protokol */
  unsigned char ifa_prefixlen; /* Délka prefixu adresy */
  unsigned char ifa_flags;     /* Příznaky */
  unsigned char ifa_scope;     /* Typ adresy */
  int           ifa_index;     /* Index rozhraní */
};
```

Hodnota v položce *ifa_family* říká, k jakému protokolu se daná adresa váže. V současné době může nabývat hodnot AF_INET (pro adresy protokolu IPv4) nebo AF_INET6 (pro adresy protokolu IPv6). Položka *ifa_prefixlen* dále obsahuje buď masku sítě (pro IPv4) nebo délku prefixu (pro IPv6). Uvedené zprávy lze rozšířit o tyto atributy:

- IFA_UNSPEC: nespecifikovaný atribut
- IFA_ADDRESS: IP adresa rozhraní
- IFA_LOCAL: lokální adresa
- IFA_LABEL: označení rozhraní
- IFA_BROADCAST: všesměrová adresa
- IFA_ANYCAST: výběrová adresa
- IFA_CACHEINFO: doplňkové informace o adrese

Atribut typu IFA_CACHEINFO dále obsahuje strukturu *ifa_cacheinfo*, která nese informace o platnosti adresy a má takovýto tvar:

```
struct ifa_cacheinfo {
  __u32   ifa_prefered;  /* Doba preferování */
  __u32   ifa_valid;     /* Doba platnosti */
  __u32   cstamp;        /* Časová značka vytvoření */
  __u32   tstamp;        /* Časová značka modifikace */
};
```

5.2 Sendd0.2

Sendd0.2 je označení původní verze implementace DoCoMo SEND. Ta pokrývá všechny povinné části specifikace protokolu SEND. Součástí je i nástroj *cgatool* sloužící k vytváření a ověřování CGA a jejich parametrů a nástroj pro vkládání rozšíření s IP adresami do certifikátu x509. Implementace byla kromě toho navržena tak, aby co největší podíl kódu byl přenositelný na jiné platformy. Zatímco kód pro konkrétní platformy je umístěn v náležitých podadresářích. Jak již bylo řečeno, tak původní zdrojové kódy již nejsou dostupné a jediná možnost je využít zdroje z projektu Native SeND kernel API for *BSD. Kód je ovšem specificky upraven pro operační systémy BSD a původní rozdělení na přenositelný a platformně specifický kód nebylo zachováno. Navíc je tato implementace závislá na modulu send.ko a podpoře protokolu SEND ze strany jádra. U projektu nebyla k dispozici

podrobnější dokumentace, tak jedinou možností, jak se seznámit s provedenými změnami, bylo studium zdrojových souborů.

5.2.1 Struktura zdrojových souborů

Byla zachována adresářová struktura zdrojových souborů původní implementace DoCoMo SEND. Toto rozložení umožňuje sdílení podprogramů mezi jednotlivými nástroji. V následujících podkapitolách je uvedena jejich stručná charakteristika.

5.2.1.1 Pomocné knihovny

Zdrojové soubory pomocných knihoven vzniklých v rámci implementace protokolu SEND se nacházejí v podadresáři *libs*. Pokud není uvedeno jinak, tak knihovny byly součástí původní implementace *sendd0.2* a nebylo nutné v nich provádět rozsáhlé úpravy.

Libappconsole

Díky knihovně *libappconsole* je možné *sendd* spustit na popředí a prostřednictvím konzole zadávat příkazy a získávat informace o běhu programu. Knihovna se skládá z inicializační funkce, funkce pro čtení znaků ze standardního vstupu a funkce, která se postará o řádné uvolnění alokovaných zdrojů po ukončení programu. Pokud je navíc *libappconsole* přeložena s knihovnou *libreadline*, lze v konzolovém prostředí využívat historii příkazů.

Libcga

Knihovna *libcga* obsahuje funkce pro vytváření a verifikaci CGA a jejích parametrů. Skládá se z datových struktur sloužící k uložení podrobných informací o CGA parametrech a funkcí pro jejich tvorbu, analýzu a verifikaci. Dále jsou zde funkce pro manipulaci s RSA klíči, které využívají kryptografické algoritmy a datové struktury obsažené v knihovně *libcrypto*.

I když byla knihovna *libcga* založena na zastaralé verzi *libcrypto*, nebylo zapotřebí provádět velké změny. Pouze byly přidány testy pro ověření správné funkce knihovny.

Libconfig

Libconfig poskytuje funkce pro načítání dat ze souboru ve formátu *klíč=hodnota*. Tato data poté uchovává v paměti uvnitř hashovací tabulky implementované knihovnou *libhashtbl*. Knihovna *libconfig* je využívána programem *sendd* při načítání konfiguračních souborů.

Knihovna *libconfig* byla součástí původní implementace *sendd0.2*, ale byla zde nalezena chyba, která způsobovala pád aplikace. Ta byla způsobena realokací dat a použitím ukazatele odkazujícího na jejich původní adresu, která už nebyla platná.

Libhashtbl

Uvnitř knihovny *libhashtbl* je implementována především hashovací tabulka. V té jsou uložena data podle klíče. Knihovna obsahuje funkce pro vytváření a odstraňování hashovací tabulky a funkce pro přidávání a odebírání prvků z tabulky. *Libhashtbl* dále umožňuje definovat vlastní funkce pro průchod tabulkou a porovnání dvou prvků.

Libincksum

Knihovna *libincksum* obsahuje funkci pro výpočet kontrolního součtu pro hlavičky paketů protokolu IP.

Liblog

Pomocí funkcí z knihovny *liblog* je možné logovat události na různé výstupy a ukládat ladící informace v rozmanitých formátech. Současné podporované výstupy jsou *stderr* a *syslog*. Záznamy událostí obsahují časovou značku, kontext, v jakém k události došlo a vlastní popis události. Tuto knihovnu jsem využil i v rámci implementace nových rozšíření.

Libpkixipext

Knihovna *libpkixipext* poskytuje zejména funkce pro vkládání rozšíření s IP adresami do certifikátů x509 podle specifikace RFC3779. Nacházejí se zde také funkce pro ověřování certifikačních cest sestavené z těchto certifikátů a funkce pro načítání konfigurace IP rozšíření ze souboru. Zpracování konfigurace probíhá lexikálním analyzátorem vytvořeným programem *lex*.

Knihovna *libpkixipext* byla založena na zastaralé verzi *libcrypto* a bylo zapotřebí provést řadu úprav. Nová verze *libcrypto* definuje přísnější typovou kontrolu některých datových struktur a část kódu nebyla již přeložitelná.

Libprioq

Uvnitř knihovny *libprioq* se nacházejí datové struktury implementující prioritní frontu. Podobně jako v *libhashtbl* jsou zde funkce pro vytváření, odstraňování prioritní fronty a funkce pro vkládání a odebírání prvků z fronty. Knihovna dále umožňuje také definovat vlastní funkce pro průchod frontou a porovnání dvou prvků ve frontě. *Libprioq* není přímo použita v žádném z nástrojů, ale je využita v knihovnách *libthrpool* a *libtimer*.

Libsendctl

Knihovna *libsendctl* implementuje kontrolní soket, který měl sloužit ke komunikaci mezi programy *cgatool* a *sendd* a operačním systémem. Nicméně jeho funkce byla nahrazena protokolem *netlink*. Knihovna *libsendctl* je současně použita pouze v nástroji *cgatool*.

Libthrpool

Libthrpool obsahuje funkce pro vytváření nových vláken a jejich následnou správu. Pokud je aplikace přeložena s podporou pro vícevláknové zpracování, tak díky této knihovně je možné paralelní zpracování určitých operací dvěma a více vlákny. Jedná se zejména o kryptografické operace a nezávislé zpracování jednotlivých zpráv protokolu ND.

Libtimer

Funkce uvnitř knihovny *libtimer* umožňují vytvářet a spravovat virtuální časovače. Ty slouží k plánování akcí, které mají proběhnout v určitém čase. Implementace správy virtuálních časovačů se skládá z jednoho reálného časovače a seznamu plánovaných akcí. Seznam plánovaných akcí je seřazen podle doby, za kterou mají akce proběhnout. Reálný časovač je nastaven vždy na nejbližší akci. Po vykonání akce je odpovídající virtuální časovač odstraněn a reálný časovač je přenastaven. Pokud je knihovna přeložena s podporou pro více vláken, tak správa časovačů běží v samostatném vlákně.

5.2.1.2 Cgatool

Nástroj *cgatool* slouží k vytváření a ověřování CGA. Jeho zdrojové kódy se nacházejí ve stejnojmenné složce *cgatool*. Tento nástroj v podstatě tvoří konzolové rozhraní pro knihovnu *libcga*.

5.2.1.3 Ipexttool

Pomocí nástroje *ipexttool* je možné vkládat do certifikátů x509 rozšíření s IP adresami. Protokol SEND používá toto rozšíření k určení, jaké prefixy nebo rozsahy prefixů je směrovač oprávněn rozesílat. Dále lze pomocí *ipexttool* ověřovat certifikační cesty složené z takovýchto certifikátů. Tento nástroj je postaven na přiložené knihovně *libpkixipext*.

5.2.1.4 Sendd

Uvnitř složky s názvem *sendd* se nachází samotná implementace démona protokolu SEND. Kód zde byl původně rozdělen na platformně nezávislý a platformně specifický. Právě zde byly prováděny největší změny v rámci projektu Native SeND kernel API for *BSD.
Nejdříve bylo nutné prostudovat zdrojové soubory a nalézt a odstranit veškerý kód specifický pro operační systémy BSD. Kód byl uschován pro pozdější reimplementaci podpory pro tento operační systém.

Byla zde nalezena částečná podpora pro operační systém Linux z původní implementace DoCoMo SEND. Ta umožňovala transport zpráv protokolu ND z jádra do uživatelského prostoru prostřednictvím *netfilter* fronty. Tuto podporu bylo ovšem nutné přepracovat, neboť byla založena na zastaralé knihovně *libipq*, která už není k dispozici. Jako její náhrada je použita knihovna

libnetfilter_queue. Pro přístup a ovládání síťových rozhraní včetně přidělování adres se zde používaly pseudo souborový systém */proc* a funkce *ioctl*. Nicméně funkce *ioctl* byla v novějších verzích jádra operačního systému Linux odebrána, protože její používání způsobovalo určitá výkonnostní omezení. Další nevýhodou bylo, že u vytváření a přidělování IP adres nebyla prováděna detekce duplicit (DAD). Navíc nebylo možné k přiřazované adrese připojit některá důležitá data. V rámci původní implementace se muselo poté neustále kontrolovat, zda nedošlo k nahrazení přidělené CGA. Uvedené nedostatky byly odstraněny použitím protokolu *netlink*.

Konfigurace *sendd0.2* je rozdělena do dvou konfiguračních souborů. To je hlavně z důvodu, že v této verzi jsou implementovány dva syntaktické analyzátory pro dva různé účely. Analyzátor pro základní konfiguraci byl napsán ručně v jazyce C a slouží pro zpracování dat v jednoduchém formátu *proměnná=hodnota*. Pro zpracování pokročilé konfigurace byl implementován robustnější analyzátor s pomocí nástrojů *lex* a *yacc*. Ten navíc dokáže rozlišit i různý kontext konfigurace. Prozatím bylo ponecháno původní řešení, nicméně v budoucí verzi bude konfigurace sjednocena a bude upřednostněn zmíněný robustnější syntaktický analyzátor.

Původní implementace obsahovala řadu drobných chyb, které se podařilo opravit a zvýšit tak stabilitu programu.

5.3 Sendd0.4

Pro novou implementaci jsem zvolil název *sendd0.4*. To z důvodu, že označení *sendd0.3* je často spojováno právě s projektem Native SeND kernel API for *BSD.

Tato kapitola obsahuje popis zásadních změn v *sendd0.4* oproti původní implementaci *sendd0.2*.

5.3.1 Zachytávání ND zpráv

Prvotní realizace zachytávání zpráv protokolu ND pro operační systém Linux byla založena na architektuře *netfilter* s pomocí již zastaralé knihovny *libipq*. Nicméně tato realizace byla částečně odstraněna v projektu Native SeND kernel API for *BSD. V tomto projektu bylo zachytávání zpráv přizpůsobeno platformě BSD. K tomu bylo využíváno modifikované jádro a modul implementující specifický soket. Tímto soketem bylo možné transportovat ND zprávy mezi prostorem jádra a programem *sendd*.

Během vývoje byla odstraněna závislost na výše uvedených změnách a obnovena podpora pro platformu Linux. Jak bylo uvedeno v návrhu, tak současná implementace zachytávání ND zpráv je rozdělena do dvou částí. První částí je pomocný *netfilter* modul a druhá část je uvnitř samotného démona *sendd*.

Netfilter modul *send* je pomocný modul jádra operačního systému Linux, který slouží k přeposílání zpráv protokolu ND z prostoru jádra do prostoru uživatelských aplikací. Základ

implementace se skládá ze tří funkcí: *send_module_init*, *send_module_cleanup* a *nd_hook*. Při zavedení modulu se zavolá funkce *send_module_init*. Uvnitř této funkce se registrují dvě filtrovací pravidla. Pravidlo reprezentuje struktura *nf_hook_ops*. Obě struktury nesou informaci o tom, že se bude filtrovat IP protokol verze 6. Jedno pravidlo se vztahuje k háku NF_INET_LOCAL_IN, na kterém se budou kontrolovat příchozí pakety a druhé pravidlo se vztahuje k háku NF_INET_LOCAL_OUT, kde bude probíhat filtrace odchozích paketů. V obou strukturách je dále uvedena funkce *nd_hook*, která implementuje vlastní filtr a bude zavolána pro každý paket splňující zmíněné podmínky, tedy pro každý paket protokolu IPv6. Funkce *nd_hook* kontroluje pakety, zda se jedná o zprávy protokolu ICMPv6 a pokud ano, tak jestli navíc jde o ND zprávy. Tyto zprávy jsou následně uloženy do fronty číslo QUEUE_NUM systému *netfilter*, kde se očekává jejich další zpracování. Ostatní pakety jsou propuštěny zpět do síťového podsystému. Funkce *send_module_cleanup* se zavolá během uvolnění modulu a postará se o odstranění výše zmíněných pravidel.

Konstanta QUEUE_NUM byla pracovně definována na hodnotu 58. V případě, že by výchozí fronta s číslem 0 byla zaplněna jinou aplikací, byly by další příchozí pakety zahazovány. Zvolením tak jiné méně používané fronty se sníží riziko ztráty paketů. Hodnotu lze změnit v příslušném zdrojovém souboru.

Uvnitř programu *sendd* je vyzvedávání ND zpráv z *netfilter* fronty založeno na funkcích z knihovny libnetfilter_queue. Během inicializace se vytvoří soket, který spojuje konkrétní *netfilter* frontu (s číslem 58) s programem *sendd*. Jakmile je zachycena nějaká zpráva protokolu ND, je vložena do speciální struktury obsahující další informace o paketu. Jsou to zejména informace o síťovém rozhraní, ze kterého přišla nebo do kterého směřuje. Tuto strukturu je možné si vyzvednout prostřednictvím zmíněného soketu. Struktura je nejprve předána funkci, která určí, zda se jedná o odchozí nebo příchozí zprávu. Poté je obsažená zpráva zpracována funkcemi implementující vlastní protokol SEND. Tyto funkce nad zprávou vynesou verdikt a vrátí ji zpět definovaným *netfilter* funkcím. Tyto funkce provedou konečné zpracování a zprávu buď přijmou, modifikují nebo zahodí. Funkce s pakety nepracují přímo, ale je nutné je kopírovat do uživatelského prostoru. Pokud tedy byla zpráva přijata a modifikována, tak je nezbytné ji nakopírovat zpět do *netfilter* fronty.

5.3.2 Netlink komunikace

Ve verzi *sendd0.4* byla nově zavedena podpora pro komunikaci prostřednictvím *netlink* zpráv. Implementace této podpory se skládá zejména ze struktury *nl_msg* reprezentující obecnou *netlink* zprávu. Struktura *nl_msg* je definována následovně:

```
struct nl_msg {
  struct nlmsghdr *hdr;
  void *buff;
  ssize_t msg_len;
  ssize_t buff_len;
};
```

Ukazatel *hdr* obsahuje adresu hlavičky uvnitř zprávy. Adresa v paměti, která je rezervována pro celou zprávu, je uložena v položce *buff*. Hodnota v *msg_len* udává velikost zprávy, naproti tomu hodnota v položce *buff_len* je celková velikost přidělené paměti pro uložení zprávy. Tato struktura byla navržena pro pohodlnější manipulaci s různými *netlink* zprávami.

Pro práci s těmito zpráva bylo dále definováno několik funkcí. Mezi ně patří zejména funkce *nl_send*, *nl_recv* a *nl_talk*. Funkce *nl_send* slouží pro odeslání *netlink* zprávy jádru. V neposlední řadě zajišťuje spolehlivý přenos a pro každou odeslanou zprávu vyžaduje potvrzení. Funkce *nl_send* je určena pro jednosměrnou komunikaci. Naproti tomu funkce *nl_recv* se stará o příjem zprávy. Tato funkce nealokuje paměť pro příchozí zprávy staticky. *Nl_recv* nejprve zjistí délku zprávy a až poté dynamicky alokuje dostatečně velkou paměť pro její uchování. S tím je spojená drobná výpočetní režie, nicméně efektivněji využívá přidělenou paměť a nemůže se stát, že by dorazila větší zpráva, než je velikost pro ni alokované paměti. Funkce *nl_talk* je v podstatě spojení obou předchozích funkcí. Byla vytvořena především pro komunikaci typu výzva - odpověď.

Netlink komunikace byla navržena pro obecné použití. Její konkrétní využití je uvnitř správy IP adres a síťových rozhraní.

5.3.3 Správa síťových rozhraní

V původní implementaci *sendd0.2* se pro získávání základních informací o síťových rozhraních využíval pseudo souborový systém */proc*. Rozhodl jsem se implementaci správy rozhraní a správy IP adres sjednotit a použil jsem k tomu robustní protokol *netlink*.

Oproti původní verzi byla tato část implementace oddělena od přenositelného kódu a byla vložena do zdrojových souborů specifických pro konkrétní operační systém, v našem případě pro Linux. V budoucnosti tak bude možné implementaci snadněji rozšířit i na jiné platformy.
Pro získání informací o všech síťových rozhraních najednou je použita zpráva protokolu *netlink*. Tato *netlink* zpráva je typu RTM_GETLINK a obsahuje příznaky NLM_F_REQUEST | NLM_F_ROOT. Sestavená zpráva je předána funkci *nl_talk*, která ji doplní o další potřebné údaje a odešle jádru. Poté je očekávána odpověď opět v podobě *netlink* zprávy, ale nyní typu RTM_NEWLINK, obsahující podrobné informace o rozhraní. Získaná data jsou následně uložena v seznamu *ifaces*. Tento seznam je složen ze struktur *s_iface*. Struktura *s_iface*, kde jsou uloženy informace o síťových rozhraních má následující definici:

```
struct s_iface {
  struct s_iface *next;      /* Ukazatel na následující prvek */
  char name[IF_NAMESIZE];    /* Název rozhraní */
  unsigned int index;        /* Index rozhraní */
  int send_enabled;          /* Příznak, zde je povolen SEND */
  void *os_data;             /* Ukazatel na doplňková data OS */
};
```

Ukazatel *next* se používá k implementaci jednosměrně vázaného seznamu. Příznak *send_enabled* určuje, na kterém rozhraní je povolen protokol SEND a které zprávy tak mají být zabezpečeny. Prostřednictvím ukazatele os_data je možné přenášet doplňující data, která jsou specifická pro konkrétní operační systém, skrz platformně nezávislý kód.

Díky použití protokolu *netlink* je navíc možné získávat informace o stavu rozhraní asynchronně bez nutnosti se neustále dotazovat. Postačí pouze doplnit seznam popisovačů pro funkci *select* v hlavní smyčce programu o identifikátor *netlink* soketu a přiřadit funkci pro zpracování *netlink* zprávy. V neposlední řadě je tu i možnost zapínat a vypínat podporu protokolu SEND pro konkrétní rozhraní za běhu aplikace.

5.3.4 Správa IP adres

Realizace funkcí pro přidělování a změny IP adres byla v původní implementaci založena na funkci *ioctl*. Funkce *ioctl* používá pro svoji práci zámek jádra (Big Kernel Lock, BKL). BKL je zastaralá metoda synchronizace uvnitř linuxového jádra založená na serializaci přístupů do kritické sekce. Jakmile do sekce vstoupí nějaký proces, musí ostatní procesy vyčkat, než tento proces zde dokončí svoji práci a sekci opustí. Tato metoda je neefektivní pro paralelní systémy, jako jsou například systémy s vícejádrovými procesory. Proto postupem času bylo používání funkce *ioctl* omezováno a vývojářům bylo doporučeno hledat jiné efektivnější řešení. Poslední pozůstatky BKL byly odstraněny v jádře verze 2.6.39 [26]. Pro naši implementaci je nejvhodnější náhradou pro přístup k síťovému podsystému použití protokolu *netlink*.

Podobně jako u správy síťových rozhraní byla tato část implementace také oddělena od přenositelného kódu a byla vložena do zdrojových souborů pro operační Linux. Struktura *s_ip6addr*, která uchovává data o IPv6 adresách, vypadá následovně:

```
struct s_ip6addr {
  struct s_ip6addr *next;       /* Ukazatel na následující prvek */
  struct s_iface *iface;        /* Ukazatel na odpovídající rozhraní */
  struct in6_addr addr;         /* Struktura obsahující IPv6 adresu*/
  struct in6_addr prefix;       /* Struktura obsahující prefix */
  int prefix_len;               /* Délka prefixu */
  uint32_t prefix_vltime;
  uint32_t prefix_pltime;
  short flags;                  /* Příznaky */
  char saddr[INET6_ADDRSTRLEN]; /* IPv6 adresa v textové řetězci */
  void *os_data;                /* Ukazatel na doplňková data OS */
};
```

Ukazatel *iface* slouží k provázání IPv6 adresy s datovou reprezentací síťového rozhraní, na kterém je přidělena. Struktura, na kterou odkazuje *addr*, obsahuje část IPv6 adresy zvané identifikátor rozhraní. V sousední struktuře prefix je uložena zbylá prefixová část adresy. Podle příznaků je možné určit stav adresy. Zejména zde ukládáme, zda byla u adresy provedena detekce duplicit. Položka *saddr* slouží ke snadnější reprezentaci adresy na textovém výstupu programu.

Platformně specifická data se nacházejí pod os_data a slouží k obdobnému účelu jako ve struktuře *s_iface*.

Pokud je zapotřebí přidělit adresu konkrétnímu rozhraní, sestaví se *netlink* zpráva typu RTM_NEWADDR. V hlavičce zprávy jsou dále nastaveny příznaky NLM_F_REQUEST | NLM_F_CREATE | NLM_F_REPLACE. Tím říkáme, že požadujeme nastavit IP adresu a pokud na rozhraní už existuje, tak ji přepsat. Do těla zprávy je vložena strukturu *ifaddrmsg*. V této struktuře je uvedeno, že nastavujeme adresu protokolu IPv6 (v položce *ifa_family*), na konkrétní rozhraní identifikované indexem (*ifa_index*) a s konkrétní délkou prefixu (v *ifa_prefixlen*). Nakonec je přidán atribut typu IFA_LOCAL obsahující konkrétní adresu a atribut IFA_CACHEINFO, ve kterém se nachází struktura *ifa_cacheinfo* s informacemi o dobách platnosti adresy. Toto nebylo v původní implementaci možné. Zpráva je následně předána funkci *nl_talk*, která se postará o její doručení. Poté se očekává potvrzení v podobě *netlink* zprávy typu NLMSG_ERROR s chybovým kódem 0.

Jestliže je potřeba odebrat určitou adresu, postupuje se obdobným způsobem. Tentokrát se ale použije *netlink* zpráva typu RTM_DELADDR s jediným příznakem NLM_F_REQUEST. Do této zprávy se nevkládá atribut IFA_CACHEINFO.

Podobně jako u správy síťových rozhraní můžeme i zde získávat informace o změnách IP adres asynchronně. Při přidělování adres protokolem *netlink* se jádro samo postará o detekci duplicit. Jediné, co je zapotřebí, je sledovat příznaky adres.

6 Konfigurace a testování

V následujících podkapitolách je uveden popis, jak lze ovládat program *sendd* a jak s jeho pomocí zabezpečit protokol ND.

V kapitole 6.2 uvádím zjednodušenou konfiguraci démona *sendd*. Tato konfigurace slouží zejména pro představu, jak je možné pomocí této implementace protokolu SEND zabezpečit ND a pokrývá jen část možných nastavení. Seznam a význam veškerých konfigurovatelných proměnných je uveden v příloze.

6.1 Parametry příkazové řádky

Po úspěšné instalaci a vytvoření nutných konfiguračních souborů je možné aplikaci spustit příkazem *sendd*. Ve výchozím nastavení aplikace běží jako démon na pozadí operačního systému. *Sendd* lze spustit i na popředí, kde je možné získávat podrobné informace o jeho aktuálním stavu.

Ihned po spuštění aplikace se provádí vyhodnocení parametrů příkazové řádky. Tyto parametry nejsou povinné a ovlivňují spouštění a běh aplikace. Možnosti jsou následující:

```
-c <soubor>
-f
-i <rozhraní>
-l <výstup>
-V
-d
```

Volitelným parametrem *c*, následovaným cestou ke konfiguračnímu souboru, lze zvolit, jaká konfigurace má být použita pro aktuální běh programu, implicitně se používá konfigurace uložená v souboru */etc/sendd.conf*. Zadáním parametru *f* je možné aplikaci spustit na popředí. Dalším parametrem *i* můžeme určit, na kterých síťových rozhraních bude protokol SEND povolen. Tento parametr lze opakovat pro libovolný počet rozraní. Implicitně je protokol povolen na všech rozhraních. Pomocí volitelného parametru *l* následovaným hodnotou *stderr*, *syslog* nebo *none* lze zvolit, kam se budou ukládat záznamy o běhu programu. Volbou *stderr* nastavíme, aby se záznamy vypisovaly na standardní chybový výstup. Toto je výchozí chování. Zvolením hodnoty *syslog* povolíme přeposílání záznamů do programu *syslog*. Ukládání záznamů můžeme zcela vypnout volbou *none*. Zadáním parametru *V* je možné vypsat stručné informace o programu. Pokud je *sendd* přeložen s definovaným makrem DEBUG, můžeme volbou *d* povolit výpisy ladících informací na standardní výstup. Tuto volbu lze opakovat až třikrát pro ještě podrobnější informace.

6.2 Zjednodušený příklad konfigurace

Před prvním spuštěním *sendd* je nutné nejdříve vytvořit parametry CGA. Pro tento účel lze použít program *cgatool*. K jejich vytvoření je zapotřebí mít k dispozici RSA pár klíčů, IPv6 prefix a hodnotu sec. Pro samotné generování CGA parametrů je z RSA páru potřeba pouze veřejný klíč, nicméně odpovídající privátní klíč bude vyžadovat démon *sendd*. Zadat veřejný klíč můžeme několika způsoby:

- certifikátem
- RSA párem klíčů uloženým v souboru ve formátu PEM
- vygenerovat klíče přímo programem *cgatool*
- zadat již dříve vytvořené CGA parametry uložené ve formátu DER

Příkaz pro vytvoření CGA parametrů a CGA může vypadat takto:

```
cgatool --gen -R 1024 -k key.pem -p 2000:: -o params.der -s 1
```

Tento příkaz vytvoří RSA pár klíčů o délce 1024 bitů a uloží jej do souboru *key.pem*. Zároveň vygeneruje i CGA parametry a vyrobí CGA pro prefix 2000:: s hodnotou sec 1 a zapíše je do souboru *param.der*. Na standardní výstup se vypíše odpovídající CGA.

Parametry CGA se vytvářejí zvlášť proto, aby mohly být znovu použity při dalším spuštění programu. Je i možné si tak uchovávat různé CGA parametry pro různé sítě s odlišnou úrovní zabezpečení. Program *cgatool* vytváří společně s CGA parametry i samotnou CGA. Nicméně démon využívá pouze CGA parametry. Po spuštění a načtení CGA parametrů *sendd* nejprve vytvoří vlastní linkovou lokální adresu. Až po obdržení nějakého prefixu z ověřeného RA vytvoří na jeho základě další adresu.

Nyní musíme provést základní konfiguraci démona *sendd*. Tu provedeme tak, že nejprve vytvoříme soubor *sendd.conf* v adresáři */etc*. Tento konfigurační soubor bude textový soubor obsahující záznamy ve formátu *proměnná=hodnota*. Zde musíme zadat alespoň proměnnou *snd_cga_params* následované cestou k druhému konfiguračnímu souboru. Ten bude obsahovat informace o CGA parametrech. Pro náš konkrétní příklad může tento záznam vypadat takto:

```
snd_cga_params=/etc/sendd/params.conf.
```

Druhý konfigurační soubor *params.conf* má trochu odlišný formát a jeho obsah může vypadat následovně:

```
named default {
  snd_cga_params /etc/sendd/params.der;
  snd_cga_priv /etc/sendd/key.pem;
  snd_cga_sec 1;
}
```

Sekce default je povinná a musí obsahovat alespoň tyto tři uvedené údaje. Dále se doporučuje vypnout automatickou konfiguraci adres protokolu ND. Ta by totiž nahrazovala CGA vytvořené

démonem *sendd* jinými adresami. Vypnout automatickou konfiguraci adres pro konkrétní síťové rozhraní můžeme tímto příkazem:

```
sysctl -w net.ipv6.conf.<rozhraní>.autoconf=0
```

Těmito kroky bude zabezpečena první část protokolu ND. Jestliže si přejeme zabezpečit i RD musíme provést pokročilou konfiguraci. Ta se liší podle toho, zda konfigurujeme *sendd* na směrovači nebo na koncové stanici.

Směrovač musí vlastnit certifikáty x509 opravňující ho k ohlašování určitých prefixů. Takové certifikáty obsahují veřejný klíč a IP rozšíření s prefixy a jsou podepsány důvěryhodnou CA. Uvedený veřejný klíč musí být použit při generování CGA parametrů. Vložit IP rozšíření s prefixy a znovu podepsat certifikát lze pomocí nástroje *ipexttool*. Předpokládá se, že tuto činnost provádí CA. Nástroj *ipexttool* načítá konfiguraci ze souboru *ipext.conf*, jehož obsah může vypadat takto:

```
addresses {
  ipv6 {
    SAFI unicast;
    prefix 2000:0:0:1::/64;
    prefix 2000:0:0:2::/64;
    prefix 2000:0:0:3::/64;
  }
}
files {
  certfile /usr/send/certs/router.pem;
  cacert /usr/send/certs/ca.pem;
  capriv /usr/send/certs/ca_priv.pem;
  outfile /usr/send/certs/router_ipext.pem;
}
```

Takto vytvořený certifikát bude vlastníka opravňovat k rozesílání prefixů 2000:0:0:1::/64, 2000:0:0:2::/64 a 2000:0:0:3::/64. Soubor *router.pem* je certifikát, který se chystáme právě modifikovat. Soubor *ca.pem* je certifikát a *ca_priv.pem* je privátní klíč podepisující certifikační autority. Podepsaný modifikovaný certifikát bude uložen do souboru *router_ipext.pem*. Aktualizovat původní certifikát na základě právě vytvořeného konfiguračního souboru lze příkazem:

```
ipexttool -w -i /etc/sendd/ipext.conf
```

Není potřeba vykonávat žádné dodatečné kroky k zajištění spolupráce mezi *radvd* a *sendd*. *Sendd* si odchozí RA zachytí v *netfilter* frontě. Administrátor musí zajistit pouze to, aby ohlašované prefixy programem *radvd* souhlasily s prefixy obsažené v IP rozšíření přiděleného certifikátu. Podrobná konfigurace *radvd* přímo nesouvisí s tématem tohoto dokumentu, a proto zde není uvedena.

Koncová stanice musí důvěřovat alespoň jedné CA a mít k dispozici její certifikát. Pod touto CA by se zpravidla měla vyskytovat CA směrovače.

Konfigurace koncové stanice se liší pouze v obsahu souboru *ipext.conf*. Záznam v souboru *ipext.conf* může vypadat následovně (soubor *ca.pem* obsahuje certifikát důvěryhodné CA):

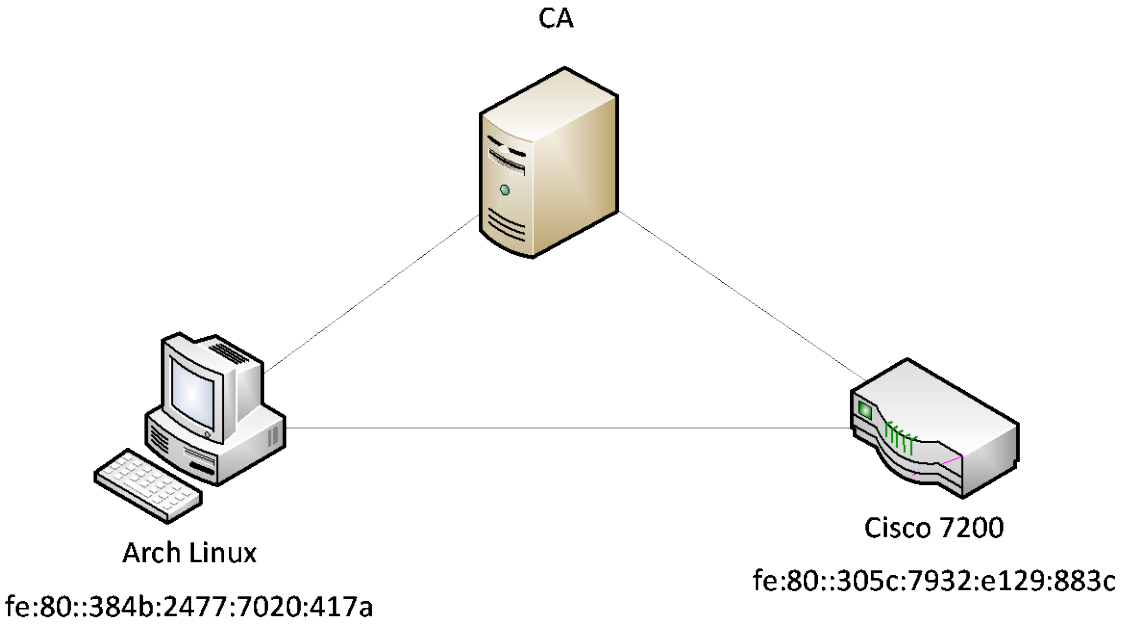
```
files {
  trustedcert /usr/certs/ca.pem;
}
```

Před samotným spuštěním démona je zapotřebí se ujistit, že pomocný modul *send* je korektně zaveden do jádra.

6.3 Testování

Během testování budeme ověřovat kompatibilitu mezi naší implementací protokolu SEND s označením *sendd0.4* a implementací společnosti Cisco. Protože obě implementace vycházely ze stejné specifikace protokolu SEND, očekává se, že budou spolupracovat.

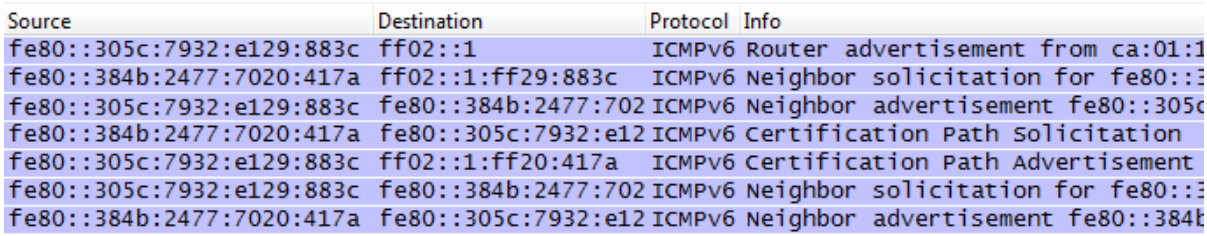
Pro účely testování postačí síť s jednoduchou topologií. Ta se skládá ze směrovače, certifikační autority (CA) a koncové stanice. Síť bude navržena a simulována v prostředí programu GNS3 a k inspekci komunikace použijeme nástroj Wireshark. Program GNS3 slouží zejména k emulaci zařízení od společnosti Cisco, ale je možné připojit i další virtuální stroje. Vybrané emulované Cisco zařízení nese označení Cisco 7206VXR NPE-400. Na tomto zařízení běží operační systém IOS verze 12.4(24)T8 z distribuce Advanced Enterprise Services. Naše implementace *sendd0.4* poběží na virtuální stroji emulovaném v prostředí programu Oracle VM VirtualBox s operačním systémem Arch Linux obsahující jádro verze 3.14.1. K získávání aktuálních informací o protokolu SEND využijeme ladících výpisů démona *sendd* a příkazu *ipv6 nd secured counters interface* systému IOS. Topologie sítě včetně přiřazených adres je ilustrována obrázkem 17. Podrobný návod ke konfiguraci Cisco zařízení je uveden v příloze.



Obrázek 17: Testovací topologie

Propojení mezi CA a zbytkem sítě nemusí být trvalé a lze jej po úspěšném vydání certifikátů přerušit.

V první části testování vystupuje *sendd* na OS Arch Linux v roli koncové stanice a IOS v roli směrovače. Konfigurace proběhla podobným způsobem, jak bylo uvedeno výše. Záznam komunikace je uveden na obrázku 18.

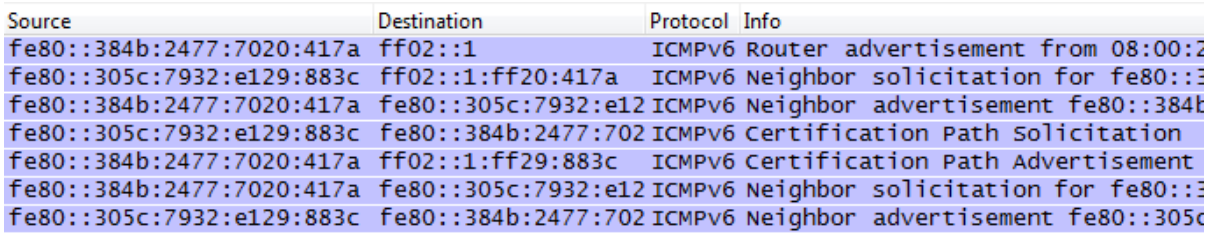


Source	Destination	Protocol	Info
fe80::305c:7932:e129:883c	ff02::1	ICMPv6	Router advertisement from ca:01:1
fe80::384b:2477:7020:417a	ff02::1:ff29:883c	ICMPv6	Neighbor solicitation for fe80::3
fe80::305c:7932:e129:883c	fe80::384b:2477:702	ICMPv6	Neighbor advertisement fe80::305c
fe80::384b:2477:7020:417a	fe80::305c:7932:e12	ICMPv6	Certification Path Solicitation
fe80::305c:7932:e129:883c	ff02::1:ff20:417a	ICMPv6	Certification Path Advertisement
fe80::305c:7932:e129:883c	fe80::384b:2477:702	ICMPv6	Neighbor solicitation for fe80::3
fe80::384b:2477:7020:417a	fe80::305c:7932:e12	ICMPv6	Neighbor advertisement fe80::384b

Obrázek 18: Sendd v režimu koncová stanice

IOS nejdříve vyšle RA. Na to reaguje ArchLinux zasláním NS, kde se dotazuje IOS na linkovou adresu. IOS odpoví zprávou NA s vlastní linkovou adresou. Protože *sendd* na OS Arch Linux směrovači s IOS ještě nedůvěřuje, pošle mu výzvu o zaslání certifikační cesty (CPS), podle které by mohl zjistit, jestli mu důvěřovat může. IOS odpoví ohlášením certifikační cesty (CPA) obsahující certifikát opravňujícího vystupovat jako směrovač. Z důvodu úspory místa zde neuvádím, zda-li zprávy obsahují všechny povinné volby protokolu SEND. Nicméně kdyby jedna nebo druhá strana neměla všechny povinné náležitosti protokolu SEND, zpráva CPS nebo CPA by v komunikaci chyběla a zařízení by už nadále spolu nekomunikovala. Toto chování vyplývá ze SEND specifikace a ND zprávy, které nesplňují bezpečnostní požadavky protokolu SEND, jsou ignorovány.

Ve druhé části testování se role zařízení prohodily. Nyní vystupuje *sendd* na OS Arch Linux v roli směrovače a IOS v roli koncové stanice. Komunikace je zaznamenána na obrázku 19.



Source	Destination	Protocol	Info
fe80::384b:2477:7020:417a	ff02::1	ICMPv6	Router advertisement from 08:00:2
fe80::305c:7932:e129:883c	ff02::1:ff20:417a	ICMPv6	Neighbor solicitation for fe80::3
fe80::384b:2477:7020:417a	fe80::305c:7932:e12	ICMPv6	Neighbor advertisement fe80::384b
fe80::305c:7932:e129:883c	fe80::384b:2477:702	ICMPv6	Certification Path Solicitation
fe80::384b:2477:7020:417a	ff02::1:ff29:883c	ICMPv6	Certification Path Advertisement
fe80::384b:2477:7020:417a	fe80::305c:7932:e12	ICMPv6	Neighbor solicitation for fe80::3
fe80::305c:7932:e129:883c	fe80::384b:2477:702	ICMPv6	Neighbor advertisement fe80::305c

Obrázek 19: Sendd v režimu směrovač

Komunikace proběhla obdobným způsobem jako v předchozí části. To znamená, že implementace protokolu SEND na operačním systému IOS a implementace *sendd* navzájem spolupracují, a to v obou režimech směrovače i koncové stanice.

7 Závěr

Původní protokol ND není zabezpečen proti řadě možných bezpečnostních hrozeb. Původní návrh předpokládal k jeho zabezpečení použít protokol IPsec. To ovšem z důvodů praktických problémů není u tohoto protokolu možné (viz kapitola 3). Proto k jeho zabezpečení vznikl protokol SEND. V současné době ale není k dispozici žádná ucelená implementace tohoto protokolu pro operační systémy GNU/Linux, která by mohla být provozována v reálném prostředí.

Výsledkem první části této práce byly zejména znalosti protokolů ND a SEND, které jsou shrnuty v odpovídajících částech dokumentu. Ze studia zdrojových souborů současných implementací protokolu SEND jsem získal řadu cenných zkušeností.

Ve druhé části diplomové práce jsem se věnoval návrhu a implementaci, kde jsem využil nabyté informace a zkušenosti. Během návrhu byly zjištěny různé problémy spojené s implementací protokolu SEND na operačních systémech GNU/Linux. Dále jsem zvažoval vytvoření zcela nové implementace nebo využití už existujícího projektu. Tyto úvahy jsem uvedl v kapitole věnující se návrhu.

Rozhodl jsem se rozšířit implementaci DoCoMo SEND. Nicméně její původní zdrojové kódy nejsou v současnosti již k dispozici a bylo nutné použít jejich modifikovanou verzi z projektu Native SeND kernel API for *BSD. Tyto zdrojové kódy bylo nejprve potřeba upravit pro použití na platformě GNU/Linux. Následně bylo nezbytné nahradit kód závislý na zastaralých knihovnách a funkcích novými technologiemi. Zvolené technologie jsou představeny v odpovídající části dokumentu. Použitím těchto technologií byly odstraněny nedostatky původní implementace. Navíc se díky nim podařilo implementaci rozšířit do prakticky použitelné podoby. V neposlední řadě byla opravena řada chyb, a tím se aplikace stala spolehlivější. Výsledná implementace nese název *sendd0.4*.

V závěru dokumentu se věnuji testování spolupráce mezi implementací *sendd0.4* a implementací, která je součástí operačního systému IOS. Testování potvrdilo jejich vzájemnou kompatibilitu uvnitř lokální sítě.

Výsledkem této práce je prakticky použitelná implementace protokolu SEND určená pro operační systémy GNU/Linux. Aktuální verze zdrojových souborů je zveřejněna online [27]. Implementaci se chci věnovat i nadále a hodlám ji rozšířit i na jiné platformy.

Literatura

[1] Neighbor Discovery for IP version 6 (IPv6). NARTEN, T., E. NORDMARK, W. SIMPSON a H. SOLIMAN. *The Internet Engineering Task Force (IETF)* [online]. September 2007 [cit. 2014-01-10]. Dostupné z: http://tools.ietf.org/html/rfc4861

[2] SATRAPA, Pavel. *IPv6: internetový protokol verze 6*. 3., aktualiz. a dopl. vyd. Praha: CZ.NIC, c2011, 407 s. CZ.NIC. ISBN 978-80-904248-4-5.

[3] Mobility Support in IPv6. JOHNSON, D., C. PERKINS a J. ARKKO. *The Internet Engineering Task Force (IETF)* [online]. June 2004 [cit. 2014-01-10]. Dostupné z: http://tools.ietf.org/html/rfc3775

[4] Default Router Preferences and More-Specific Routes. DRAVES, R., D. THALER. *The Internet Engineering Task Force (IETF)* [online]. November 2005 [cit. 2014-01-10]. Dostupné z: http://tools.ietf.org/html/rfc4191

[5] Neighbor Discovery Proxies (ND Proxy). THALER, D., M. TALWAR a C. PATEL. *The Internet Engineering Task Force (IETF)* [online]. April 2006 [cit. 2014-01-14]. Dostupné z: http://tools.ietf.org/html/rfc4389

[6] IPv6 Neighbor Discovery (ND) Trust Models and Threats. NIKANDER, P., J. KEMPF a E. NORDMARK. *The Internet Engineering Task Force (IETF)* [online]. The Internet Society, May 2004 [cit. 2014-01-10]. Dostupné z: http://www.ietf.org/rfc/rfc3756.txt

[7] Neighbor Discovery for IP version 6 (IPv6). NARTEN, T., E. NORDMARK a W. SIMPSON. *The Internet Engineering Task Force (IETF)* [online]. December 1998 [cit. 2014-01-10]. Dostupné z: http://tools.ietf.org/html/rfc2461

[8] SEcure Neighbor Discovery (SEND). ARKKO, J., J. KEMPF, B. ZILL a P. NIKANDER. *The Internet Engineering Task Force (IETF)* [online]. The Internet Society, March 2005 [cit. 2014-01-10]. Dostupné z: http://tools.ietf.org/html/rfc3971

[9] Effects of ICMPv6 on IKE and IPsec Policies. ARKKO. *The Internet Engineering Task Force (IETF)* [online]. The Internet Society, 2 June 2002 [cit. 2014-01-10]. Dostupné z: https://tools.ietf.org/html/draft-arkko-icmpv6-ike-effects-01

[10] NTT DOCOMO. DoCoMo_SEND_UserGuide.pdf. [2006]. Dostupné z: https://google-summer-of-code-2009-freebsd.googlecode.com/files/Ana_Kukec.tar.gz

[11] Google-summer-of-code-2009-freebsd: IPv6 Secure Neighbor Discovery - native kernel APIs for FreeBSD. Google Code [online]. [Sep 4, 2009] [cit. 2014-01-11]. Dostupné z: https://google-summer-of-code-2009-freebsd.googlecode.com/files/Ana_Kukec.tar.gz

[12] KUKEC, Ana. Native SeND kernel API for *BSD[online]. 2010 [cit. 2014-01-10]. Dostupné z: http://people.freebsd.org/~anchie/SeND_AsiaBSDCon_2010.pdf

[13] Secure Neighbor Discovery (SeND). KUKEC, Ana. FreeBSD.org personal home pages [online]. 2010 [cit. 2014-01-11]. Dostupné z: http://people.freebsd.org/~anchie/

[14] Easy-SEND. SourceForge - Download, Develop and Publish Free Open Source Software [online]. © 2014 [cit. 2014-01-11]. Dostupné z: http://sourceforge.net/projects/easy-send/

[15] Ipv6-send-cga: an implementation of SEND protocol in LINUX kernel. Google Code [online]. [Dec 2009] [cit. 2014-01-11]. Dostupné z: http://code.google.com/p/ipv6-send-cga/

[16] NDprotector: an implementation of CGA & SEND for GNU/Linux based on Scapy6. Amnesiak.org [online]. [(c) 2009] [cit. 2014-01-11]. Dostupné z: http://amnesiak.org/NDprotector/

[17] TrustRouter. GitHub · Build software better, together. [online]. © 2014 [cit. 2014-01-11]. Dostupné z: https://github.com/TrustRouter/TrustRouter

[18] HPI ITS: WinSEND. *Hasso-Plattner-Institut: Willkommen* [online]. (c) 2014 [cit. 2014-05-27]. Dostupné z: http://www.hpi.uni-potsdam.de/meinel/security_tech/ipv6_security/winsend.html

[19] Icmp(7) - Linux manual page. *Michael Kerrisk - man7.org* [online]. 2012-05-10 [cit. 2014-05-27]. Dostupné z: http://man7.org/linux/man-pages/man7/icmp.7.html

[20] Linux Technology Center : Welcome. *IBM Linux Portal* [online]. [2004] [cit. 2014-05-27]. Dostupné z: http://sourceware.org/systemtap/kprobes/

[21] *Netfilter/iptables project homepage - The netfilter.org project* [online]. © 1999-2014 [cit. 2014-05-27]. Dostupné z: http://www.netfilter.org/

[22] DOCOMO USA Labs: Corporate Overview. *Internet Archive: Wayback Machine* [online]. [2012-03-19] [cit. 2014-05-27]. Dostupné z: http://web.archive.org/web/20120319200932/http://www.docomolabs-usa.com/lab_opensource.html

[23] *OpenSSL: The Open Source toolkit for SSL/TLS* [online]. © 1999-2014 [cit. 2014-05-27]. Dostupné z: http://www.openssl.org/

[24] Netlink(7) - Linux manual page. *Michael Kerrisk - man7.org* [online]. 2013-03-15 [cit. 2014-05-27]. Dostupné z: http://man7.org/linux/man-pages/man7/netlink.7.html

[25] Rtnetlink(7) - Linux manual page. *Michael Kerrisk - man7.org* [online]. 2013-03-05 [cit. 2014-05-27]. Dostupné z: http://man7.org/linux/man-pages/man7/rtnetlink.7.html

[26] BigKernelLock - Linux Kernel Newbies. *Linux Kernel Newbies - Linux Kernel Newbies* [online]. [2011], 2011-05-19 [cit. 2014-05-27]. Dostupné z: http://kernelnewbies.org/BigKernelLock

[27] BEZDÍČEK, Lukáš. LucasBe/sendd · GitHub. *GitHub · Build software better, together.* [online]. © 2014 [cit. 2014-05-27]. Dostupné z: https://github.com/LucasBe/sendd

Seznam příloh