VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

MODELOVÁNÍ IPV6 V PROSTŘEDÍ OMNET++

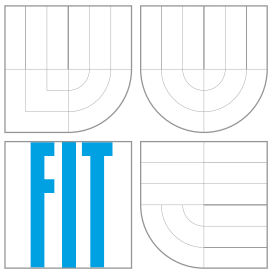
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MAREK ČERNÝ

BRNO 2011

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

MODELOVÁNÍ IPV6 V PROSTŘEDÍ OMNET++

IPV6 MODELLING IN OMNET++

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MAREK ČERNÝ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. VLADIMÍR VESELÝ

BRNO 2011

Abstrakt

OMNeT++ je diskrétní simulátor hojně používaný k vytváření nejrůznějších síťových simulací. Lze jej dále rozšířit např. frameworkem INET, který obsahuje modely protokolů a zařízení z prostředí TCP/IP síti. V této práci se soustředíme na zkoumání současných možností balíku INET modelovat internetový protokol verze 6. Obzvláště se zaměřujeme na podporu směrování. V rámci implementace pak byly vytvořeny moduly dual-stack směrovače a dual-stack klienta s podporou manuální i automatické IPv6 adresace a statického směrování. Je také představen modul OSPFv3, který poskytuje většinu podpůrných funkcí a je připraven na budoucí vývoj samotného jádra směrovacího protokolu.

Abstract

OMNeT++ is a discrete-event simulator commonly used to build various network simulations. It can be extended by INET framework that supplies models of protocols and devices from TCP/IP environment. This paper explores current capabilities of INET to model IPv6, particularly routing. Implemented extension includes modules of dual-stack router and dual-stack host that support XML-based configuration of IPv6 addressing and static routing. We also introduce an OSPFv3 module that implements most of auxiliary features and is ready for future development of the routing protocol core.

Klíčová slova

IPv6, OMNeT++, INET, ANSA, dual-stack, adresace, autokonfigurace, směrování, OSPFv3, síťová simulace, diskrétní simulace

Keywords

IPv6, OMNeT++, INET, ANSA, dual-stack, addressing, autoconfiguration, routing, OSPFv3, network simulation, discrete-event simulation

Citace

Marek Černý: Modelování IPv6 v prostředí OMNeT++, diplomová práce, Brno, FIT VUT v Brně, 2011

Modelování IPv6 v prostředí OMNeT++

Prohlášení

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

. . . . . .
Marek Černý
25. května 2011

Poděkování

Na tomto místě bych rád poděkoval svému vedoucímu Ing. Vladimíru Veselému za zasvěcení do problematiky, odbornou pomoc a možnost zapojit se do tvůrčího kolektivu projektu ANSA. Zejména chci poděkovat svému kolegovi Zdeňku Krausovi za jeho spolupráci při objevování prostředí OMNeT++. Velké díky patří mé rodině za podporu a trpělivost nejen během psaní této diplomové práce. Také bych rád poděkoval svým přátelům Vojtovi Kuličkovi a Michalu Čamborovi, především za neutuchající motivaci pro překonávání nejrůznějších překážek našeho společného studia. Můj dík si rovněž zaslouží pánové Sammet a Lucassen, jejichž tvorba mi byla vždy inspirací a v těžkých chvílích mi pomáhala udržet si zdravý rozum.

Obsah

Úvod

Pokud bychom měli vybrat několik technologických vynálezů, které na přelomu století nejvíce pronikly do obecného povědomí a staly se de facto nedílnou součástí běžného života, jedním z nich by byl bezesporu Internet. Původem armádní experiment zkoumající nový koncept síťové komunikace, dnes základní služba pro přenos dat, volně dostupný prostředek ke komunikaci, nevyčerpatelný zdroj informací i zábavy. A podobně jako řada dalších výtvorů z oblasti vědy a techniky, i Internet je s přibývajícím počtem uživatelů čím dál více vnímám jako něco zcela samozřejmého.

Navzdory masivnímu růstu a rozšíření i do odvětví, která v době vzniku Internetu ani nemusela existovat, zcela základní principy této počítačové sítě zůstaly prakticky neměnné. Mimo jiné se jedná o sadu používaných komunikačních protokolů, známou také jako *Internet Protocol Suite* nebo TCP/IP. Jedním z nejdůležitějších je pak tzv. *Internet Protocol* (IP), podle kterého dnes označujeme termínem „IP sítě“ nejen Internet, ale všechny počítačové sítě postavené na této architektuře. S růstem Internetu se však ukázalo, že ačkoliv po principiální stránce byl protokol IP navržen nadčasově a svými funkcemi je v podstatě dostačující i dnes, prostor 32bitových identifikátorů pro jednotlivé síťové uzly se velice rychle vyčerpává. Byly nasazeny speciální techniky, které tento trend úspěšně zpomalují, za skutečné řešení problému je však považován až přechod na novější *Internet Protocol* verze 6 (IPv6, původní IP pak bývá podle své verze označován jako IPv4). Existují mechanismy, jak tento přechod usnadnit, a díky nim IPv4 a IPv6 již v současnosti na Internetu koexistují vedle sebe [4, 13].

V praxi však původní verze IP stále dominuje. Jedním z důvodů je bezesporu fakt, že jsou v sítích často stále nasazena zařízení podporující pouze IPv4, a přechod na IPv6 tak pro provozovatele znamená investici do nového vybavení a aktualizaci síťové konfigurace. Tento proces lze usnadnit kvalitně provedenou analýzou, jejíž součástí může být i série simulací. Jejich význam je zřejmý – virtuální model sítě nám umožní sledovat její chování v zadaném nastavení, a zavčas tak odhalit případné chyby v návrhu nebo jej na základě našich pozorování vylepšit. A to obvykle rychleji a za nesrovnatelně nižší náklady, než kdybychom tyto úpravy prováděli až při reálném nasazení. Předpokladem je však existující simulační prostředí, které umožňuje vytvoření dostatečně přesného modelu a poskytuje vhodnou míru abstrakce.

Cíle této práce

Jedním z takových nástrojů je pro akademické a výukové účely volně dostupný simulátor OMNeT++. Ten ale nabízí pouze základní vývojové a simulační prostředí, pro aplikačně specifické modelování je proto dále k dispozici celá řada rozšiřujících projektů. Tyto frameworky obvykle obsahují sadu předpřipravených modulů a ukázkových příkladů, a usnadňují tak proces vytváření simulací z dané oblasti. Jmenujme například MiXiM zaměřený na mobilní a bezdrátové sítě, OverSim modelující peer-to-peer komunikaci, nebo ReaSE po-

skytující nástroje pro tvorbu síťových topologií a generování reálných datových toků. Od samotných autorů OMNeT++ je pak ke stažení komplexní balík s názvem INET, který disponuje prostředky k simulaci nejpoužívanějších protokolů ze sady TCP/IP.

V rámci projektu ANSA na FIT VUT v Brně se snažíme INET zkoumat, rozšiřovat a doplňovat o nové funkce, které reflektují chování reálných síťových zařízení. Jako příklad může posloužit model směrovače. Zatímco v balíku INET se jedná o relativně jednoduché zařízení schopné plnit svoji základní funkci, v ANSA rozšíření je doplněn o dynamické směrovací protokoly RIP a OSPF, síťový provoz je možné filtrovat pomocí ACL, nastavení modulu lze provést načtením reálného konfiguračního souboru z Cisco směrovače a další. Tato rozšíření jsou již v současnosti používána při výuce, svou roli zde hraje především grafické rozhraní aplikace OMNeT++, které dokáže velmi názorně vizualizovat činnost zařízení a komunikačních protokolů. Dlouhodobým záměrem je pak poskytnout nástroje, které umožní provádět formální analýzu a verifikaci chování a bezpečnosti počítačových sítí.

Tato práce se zabývá zkoumáním současných možností frameworku INET na poli IPv6, obzvláště se zaměřením na směrování a možnosti konfigurace zařízení. Na základě zjištěných informací se poté pokusíme navrhnout a implementovat vhodná rozšíření, která přispějí k vytvoření komplexního modelu směrovače s IPv6 podporou.

Obsah a struktura dokumentu

Úvodem je třeba shrnout základními principy protokolu IPv6 a postihnout významné rozdíly vzhledem k jeho předchozí verzi, této problematice se věnuje kapitola 1. V podobném duchu se kapitola 2 zaměřuje na dynamický směrovací protokol OSPFv3, jehož implementací se později budeme zabývat. Následuje kapitola 3, která má za úkol čtenáři představit aplikaci OMNeT++ a framework INET – popisujeme zde možnosti simulačního prostředí, způsob implementace nových modulů a proces vytváření vlastních simulací. Kapitola 4 pak mapuje současný stav podpory IPv6 v prostředí INET. Implementovaná rozšíření jsou prezentována v kapitole 5. Práci uzavírá kapitola 6, ve které je ukázáno praktické využití implementovaných modelů na několika simulacích.

Tato práce navazuje na dokument vytvořený v rámci semestrálního projektu, jehož obhajoba proběhla v lednu 2011. S drobnými úpravami byly převzaty popisné kapitoly 1 a 3. Kapitola 4 je doplněna o další nově zjištěné detaily o současném stavu IPv6 podpory frameworku INET v oblasti směrování, které jsou přínosné k lepšímu pochopení implementační části. Obdoba kapitoly 5 v semestrálním projektu pouze představila základní model vytvořeného *dual-stack* směrovače a věnovala se návrhu XML struktury pro budoucí konfiguraci tohoto zařízení. V rámci této diplomové práce došlo k mnohým rozšířením směrovače – XML struktura byla aktualizována a doplněna o další příkazy (především pro statické a dynamické směrování) a nově implementovaný modul `deviceConfigurator` slouží k načtení a aplikaci této konfigurace. Pro vylepšení možností statického směrování došlo k úpravám modulu směrovací tabulky `RoutingTable6`. Implementaci kostry protokolu OSPFv3 lze pak nalézt v modulu `ospf6`.

Kapitola 1

IPv6

Prostor volných IPv4 adres spravovaný organizací IANA byl vyčerpán v lednu 2011 [20] a je tedy jen otázkou času, než tyto přidělené bloky regionální registrátoři zcela využijí. I přes nasazování techniky NAT a hledání nevyužívaných IPv4 adres se tak brzké rozšíření IPv6 stává nevyhnutelným. Tento přechod se snaží usnadnit některé mechanismy, které byly navrženy za účelem umožnit provozovat obě verze protokolu zároveň.

V podstatě všechny moderní operační systémy IPv6 podporují a jsou přitom zpětně kompatibilní s IPv4 [26]. Tuto implementaci nazýváme *dual-stack* a kromě počítačů se s ní lze setkat i u dalších síťových zařízení, nejčastěji směrovačů. Na jednom rozhraní je pak možné přijímat a odesílat IPv6 i IPv4 pakety a na síťovém segmentu tak není nutné exkluzivně provozovat pouze jednu z verzí protokolu.

Existuje také několik technik síťového tunelování, kdy IPv6 pakety obvykle zapouzdřujeme to paketů jiného protokolu, za účelem jejich přenosu skrze část sítě bez IPv6 podpory. Tímto způsobem mohou uživatelé využívat IPv6 konektivitu i v případě, kdy jejich poskytovatel internetového připojení přes IPv6 nenabízí.

Tyto metody byly však navrženy pouze jako dočasná řešení, konečným cílem je plnohodnotné nasazení IPv6 a možnost nativně využívat veškeré funkce tohoto protokolu. Přechod na IPv6 totiž není jenom o získání většího adresového prostoru, ale zároveň poskytuje i jedinečnou příležitost aktualizovat zastaralé nebo zavést zcela nové mechanismy, které reflektují vývoj počítačových sítí za několik posledních desetiletí. Těm nejzajímavějším z nich se věnuje následující kapitola. Předpokládáme proto, že je čtenář se základy síťové komunikace obeznámen a má představu o architektuře a fungování protokolu IPv4.

1.1 Adresace

Protože je síťová adresace nejvýraznější funkce IP protokolu, zaměříme se nejprve na tuto oblast. Seznámení se se související terminologií je totiž nezbytné pro další výklad, především v kapitolách 4 a 5 při zkoumání současného stavu simulačního IPv6 modulu a popisování jeho rozšíření.

1.1.1 Notace IPv6 adres

Nejzásadnější změnou, kterou IPv6 přináší, je bezesporu změna síťových adres z 32bitových na 128bitové (RFC 4291 [10]). Obvyklá notace pro IPv4 adresy je čtveřice bajtů v desítkové soustavě oddělených tečkami, tedy např. `192.0.2.1`. Naproti tomu IPv6 zapisujeme v soustavě šestnáctkové jako 8 skupin (po 16 bitech) oddělených dvojtečkami. Adresa pak může

vypadat následovně: `2001:0db8:0000:0000:0000:0000:0000:0001`. Pro přehlednější zápis je však možné vynechat jak předřazené nuly v jednotlivých skupinách, tak i jednu sekvenci nulových skupin a nahradit ji dvojicí dvojteček. Kompaktní zápis výše uvedené adresy by tedy byl `2001:db8::1`.

Podobně jako v případě IPv4, i u IPv6 adres značíme, kolik počátečních bitů adresy určuje síť. Zbytek pak identifikuje konkrétní zařízení resp. jeho rozhraní. V IPv6 první část označujeme jako „síťový prefix“, případně mluvíme o „délce prefixu“. Notace je podobná jako zkrácený zápis masky sítě pro IPv4, tedy lomítko a dekadické číslo psané za adresou. Jako příklad si uveďme adresu `2001:db8::1/64` s délkou prefixu 64 bitů. Samotný síťový prefix je tudíž prvních 64 bitů z adresy, za kterými následuje dalších 64 nulových bitů: `2001:db8::/64`. Termín prefix se proto typicky používá v souvislosti se směrovači – směrovač šíří informace o síťovém prefixu a podobně.

1.1.2 Typy IPv6 adres

V prostředí IPv4 byly v průběhu vývoje podniknuty některé kroky, které měly zamezit „plýtvání“ adresami. Jmenujme například zavedení beztřídní adresace (CIDR, *Classless Inter-Domain Routing*), která nahradila tabulkové rozdělení adres výrazně flexibilnějším používáním síťové masky libovolné délky (VLSM, *Variable-Length Subnet Masking*). Další běžnou praktikou je skrývání lokálních sítí za jedinou globální IPv4 adresu pomocí techniky NAT (*Network Address Transaction*, překlad síťových adres).

IPv6 však vzhledem k enormně velkému prostoru těmito problémy netrpí a nepředpokládá se, že by se dostupné adresy mohly někdy vyčerpat. Nabízí se tedy možnost nakládat s nimi více volně. V síti používající protokol IPv4 je každé zařízení resp. jeho síťové rozhraní zastupováno jedinou unikátní adresou, kterou pak využívají veškeré služby. IPv6 umožňuje jednomu rozhraní přiřadit adres více a představuje také několik různých typů adres. Díky tomu jsou zařízení schopna mj. komunikovat se svými bezprostředními sousedy pomocí speciální lokální linkové adresy ještě před tím, než je jim přiřazena adresa platná pro celou síť. Další změnou je zrušení adres typu broadcast – předpokládá se, že funkcionalitu všesměrového vysílání převezmou speciální multicastové adresy. Novým typem komunikace je pak anycast, který umožňuje zaslání zprávy na „libovolné“ ze skupiny zařízení. Tuto techniku často používají stanice ke kontaktování toho ze skupiny směrovačů, který jim je v síti nejblíže.

V souladu s těmito změnami můžeme v současnosti IPv6 adresy zjednodušeně rozdělit do několika skupin ([10], sekce 2.4-2.7):

- **Globální adresy** jsou „klasické“ adresy pro identifikaci v Internetu. Skládají se 48bitového prefixu sítě, 16bitového identifikátoru podsítě a 64 bitů označujících konkrétní síťové rozhraní. Prefix může být v podstatě libovolný, kromě prefixů vyhrazených pro jiné použití, z nichž některé zmíníme v následujícím výčtu.

- **Lokální linkové adresy** používají prefix `fe80::/10`, za kterým následuje 54 nulových bitů a 64bitový identifikátor rozhraní. Adresy slouží ke komunikaci v rámci jedné sítě z pohledu linkové vrstvy, nejsou proto dále zpracovávány směrovači.[1]

- **Unikátní lokální adresy** identifikované prefixem `fc00::/7` stojí na pomezí předchozích dvou. Jedná se o obdobu privátních IPv4 adres (sítě `10.0.0.0/8`, `172.16.0.0/12`

[1] Termínem „linka“ budeme proto v kontextu IPv6 označovat nikoliv spoj mezi dvěma uzly, ale síť definovanou dosahem komunikace pouze v rámci linkové vrstvy. Hraničním bodem je tedy nejbližší směrovač.

a `192.168.0.0/24`) určených pro adresování v místních sítích. Lze je tedy směrovat, avšak na Internetu by se vyskytnout neměly. I přesto byl ale v RFC 4193 [12] definován mechanismus pro generování dalších 40 pseudonáhodných bitů prefixu, který by měl zajistit globální jedinečnost i těchto lokálních adres.

- **Multicastové adresy** s prefixem `ff00::/8` jsou dále hierarchicky členěny do skupin na základě 8 bitů následujících za tímto prefixem. Existují tak například multicastové adresy v rámci rozhraní, linky, různě velkých lokálních sítí a pak samozřejmě i globální multicastové adresy.

- Další speciální adresy – sem můžeme zařadit např. *loopback* adresu `::1/128`, adresu pro výchozí cestu `::/0` nebo obdobný výraz `::/128` pro „nespecifikovanou adresu“. Kromě těchto jsou některé prefixy vyhrazené pro speciální použití, například `2001::/32` resp. `2002::/16` jsou používány technikami pro tunelování [4] resp. překlad [13] IPv6 adres na hranici s IPv4 sítí.

.

1.2 Neighbor Discovery Protocol

NDP je protokol linkové vrstvy, který poskytuje podpůrné funkce nezbytné pro správnou funkci IPv6. Z pohledu IPv4 nahrazuje některé mechanismy poskytované protokoly ICMP a ARP, tyto však doplňuje o další užitečné prostředky, z nichž nejvýraznější je bezesporu bezstavová konfigurace. NDP je specifikován v RFC 4861 [15] a zajišťuje následující:

- ***Router Discovery*** – Objevování směrovačů v síti.
- ***Prefix Discovery*** – Získání seznamu prefixů, které se v okolí používají.
- ***Parameter Discovery*** – Zjištění nastavení sítě, např. MTU linkové vrstvy nebo maximální počet skoků při směrování.
- ***Address Autoconfiguration*** – Technika umožňující zařízení samostatně si vygenerovat a přiřadit IPv6 adresu.
- ***Address Resolution*** – Náhrada ARP protokolu pro získávání fyzických adres zařízení.
- ***Next-hop Determination*** – Algoritmus aplikovaný na každou odchozí zprávu pro optimální určení sousedního směrovače nebo hosta, na který bude zpráva na úrovni linkové vrstvy zaslána.
- ***Neighbor Unreachability*** – Schopnost detekovat ztrátu dostupnosti sousedů.
- ***Duplicate Address Detection*** – Technika umožňující ověřit, zda je zvolená IPv6 adresa v síti unikátní.
- ***Redirect*** – Funkce, pomocí níž směrovače sdělují hostům, která zařízení v síti jsou výchozí brány pro jednotlivé síťové prefixy.

Nedílnou součástí IPv6 je také protokol ICMPv6 (*Internet Control Message Protocol*), který nabízí dvě skupiny ICMPv6 zpráv - informační a chybové. Smyslem protokolu je mj. poskytnout funkce pro diagnostiku sítě a umožnit generování hlášení o chybách. Mezi nejznámější patří zprávy typu *Echo Request* zasílaná programem *ping*, která slouží k ověření konektivity mezi dvěma uzly v síti. Adresát po přijetí odpovídá zprávou *Echo Reply* – pokud se tato úspěšně vrátí odesílateli, můžeme spojení považovat za funkční. Podle názvu aplikace bývá celá tato procedura označována termínem „ping“.

Mezi další typy ICMPv6 zpráv patří petice definovaná protokolem NDP. Ty slouží k zajištění výše popsané funkcionality protokolu. Jedná se o následující:

- ***Router Solicitation*** – výzva směrovači
- ***Router Advertisment*** – ohlášení směrovače
- ***Neighbor Solicitation*** – výzva sousedovi
- ***Neighbor Advertisment*** – ohlášení souseda
- ***Redirect*** – viz výše

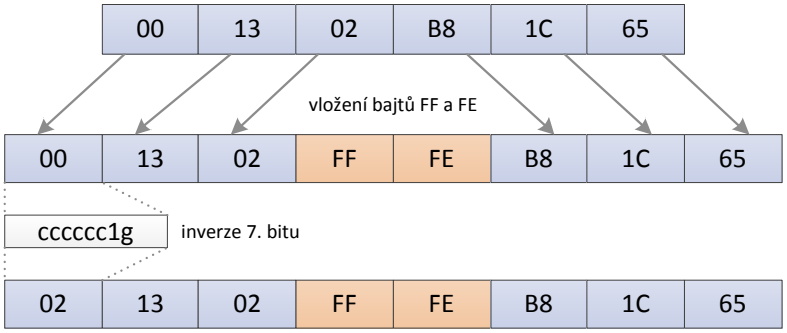
Kvůli mechanismům spojeným s těmito zprávami je IPv6 síť i v klidu na rozdíl od IPv4 poměrně aktivní. Každý klient po připojení do sítě vyšle několik *Router Solicitation* multicastových zpráv adresovaných všem směrovačům. Směrovače, které mají tuto funkci povolenu, odpovídají *Router Advertisement* zprávou s informacemi o síti. Zasílání RA zpráv probíhá nejen jako odpověď na přijatou RS výzvu, ale také periodicky (ve výchozím nastavením přibližně každých 200-600 sekund) po celou dobu běhu směrovače.

V případě, že chce zařízení zaslat IPv6 datagram, je třeba nejprve zjistit fyzickou adresu cíle nebo nejbližšího směrovače. K tomu slouží *Neighbor Solicitation*, výzva zasílaná opět pomocí multicastové komunikace, ve které se odesílatel dotazuje na zadanou IPv6 adresu. Síťové rozhraní, které v přijaté zprávě našlo svoji IPv6 adresu, odpovídá zasláním zprávy typu *Neighbor Advertisement* obsahující mj. i fyzickou adresu rozhraní. Tento údaj si původní adresát po přijetí NA zprávy uloží do tabulky sousedů, pomocí které je pak již schopen zapouzdřit IPv6 datagram do rámce linkové vrstvy a odeslat jej do sítě.

1.3 Bezstavová autokonfigurace adres

Na možnostech NDP protokolu staví zajímavá technika popsaná v RFC 4862 [22] zvaná bezstavová autokonfigurace adres. Základní funkcí je schopnost zařízení vygenerovat a přiřadit si svoji lokální linkovou adresu. Jak uvádí přehled v podkapitole 1.1.2, prvních 64 bitů adresy je dáno specifikací. Zbývá tedy získat unikátní 64bitový identifikátor rozhraní. Pro tento účel se typicky používá modifikace adresy EUI-64 definované institutem IEEE, která vychází z unikátní 48bitové fyzické adresy síťového hardwaru. Přesný postup konverze fyzické adresy na modifikované EUI-64 je popsán například v dodatku A RFC 4291 [11]. Identifikátor lze však generovat i náhodně, protože dalším krokem před přiřazením adresy je aplikace funkce *Duplicate Address Detection* protokolu NDP, která unikátnost adresy ověří.

Pokud výše uvedený postup doplníme o informace získané NDP protokolem v rámci *Router Discovery*, *Prefix Discovery* a *Parameter Discovery*, zařízení je schopné přiřadit si nejen lokální linkovou adresu, ale i globální adresu s platným prefixem. Po doplnění informací o okolních směrovačích do směrovací tabulky je pak stanice plně nakonfigurována pro síťovou komunikaci.

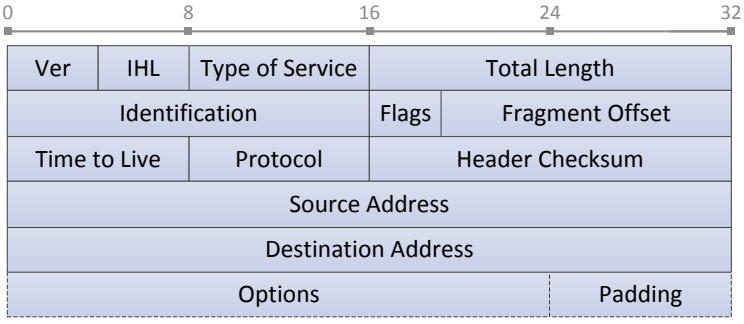


Obrázek 1.1: Ukázka transformace fyzické adresy na modifikované EUI-64

1.4 Protokolová hlavička

V této podkapitole se zaměříme na rozdíly mezi IPv4 a IPv6 na úrovni formátu protokolové hlavičky. Není naším cílem zabývat se implementačními detaily, přesný formát hlaviček a význam jednotlivých položek lze dohledat v RFC 791 [16] a RFC 2460 [9]. Uvedeme si však některé zásadní rozdíly, které mění zavedené zvyklosti nebo přináší novou funkcionalitu.

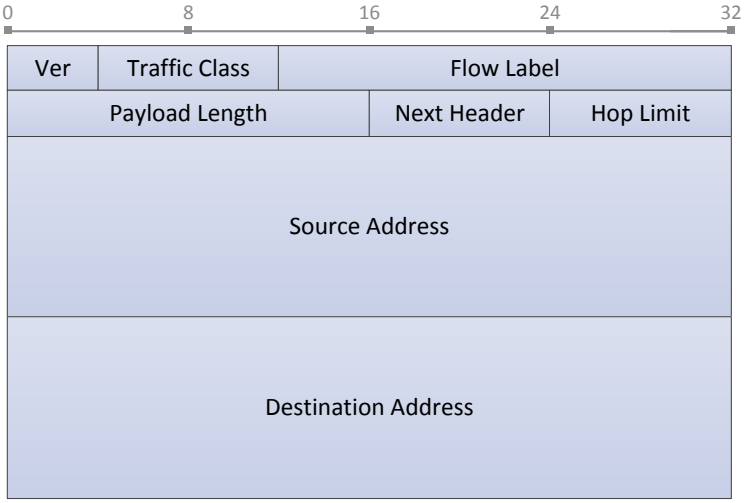
Pevná část hlavičky protokolu IPv4 je velká 20 bajtů, z nichž nejdůležitější je dvojice 32bitových adres, které určují zdroj a cíl paketu (8 bajtů). Dalších 12 bajtů je využito např. pro kontrolní součet, informaci o fragmentaci, nastavení příznaků, údaje o verzi protokolu, typu služby, délce paketu a další. Volitelně následují tzv. volby, které dále specifikují některé informace, nebo požadavky, přičemž délka této části není pevně daná. Ačkoliv je však IPv4 hlavička poměrně komplikovaná, v průběhu vývoje se ukázala býti nedostatečně flexibilní a různá rozšíření se proto musí často uchylovat ke kompromisům a např. využívat některé nevyužívané bity pro jiný účel, než pro které byly původně navrhnuty.



Obrázek 1.2: Formát IPv4 hlavičky (zpracováno dle [16], sekce 3.1)

IPv6 si klade za cíl formát hlavičky zjednodušit, ale zároveň učinit více flexibilní pro případná rozšíření. Hlavička je pochopitelně větší kvůli dvojici 128bitových IPv6 adres (32 bajtů), veškeré další položky však byly zredukovány na fixních 8 bajtů. Celá IPv6 hlavička

má tedy vždy přesně 40 bajtů a na rozdíl od IPv4 hlavičky s proměnnou velikostí se tak více hodí pro rychlé hardwarové zpracovávání ve směrovačích. Ze stejného důvodu byl také odstraněn kontrolní součet a fragmentace. Detekce chyb totiž probíhá na nižších i vyšších vrstvách a integrita dat by tak měla být dostatečně zaručena. Fragmentace se naopak v průběhu vývoje ukázala býti spíše výjimečným jevem, který z pohledu jednoduchého zpracování paketů není žádoucí. Funkce byla proto zachována pouze v jednodušší *end-to-end* ([9], sekce 5) podobě.



Obrázek 1.3: Formát IPv6 hlavičky (zpracováno dle [9], sekce 3)

Funkcionalitu nejrůznějších voleb a příznaků, které se nacházejí v IPv4 hlavičce, pak nahrazuje nová technika tzv. zřetězených hlaviček. IPv6 hlavička je tedy maximálně zjednodušená, zároveň však obsahuje novou položku *Next Header* (další hlavička), což je v podstatě identifikátor další specializované hlavičky, která bezprostředně následuje. RFC 2360 [9] uvádí řadu těchto hlaviček, jmenujme např. *Routing Header* (výčet uzlů, přes které má být paket směrován) nebo *Authentication Header* (prostředek protokolu IPsec k autentizaci spojení). Z principu je však tento návrh otevřený k budoucím rozšířením, protože nově nadefinovaná hlavička znamená z pohledu IPv6 protokolu pouze zavedení její číselné identifikace. Aby bylo možné hlavičky zřetězovat za sebe, musí každá z nich položku *Next Header* obsahovat a poslední z nich zde pak nastaví vyhrazenou hodnotu 59 s významem *No Next Header* (žádná další hlavička).

Kapitola 2

Směrovací protokol OSPFv3

Účelem síťové vrstvy, kterou protokol IP reprezentuje, je zajistit transport dat mezi dvěma koncovými stanicemi. Toho je dosaženo za pomoci tzv. směrování, techniky k určování dalšího uzlu na trase pro nalezení optimální cesty k cíli. Směrování v Internetu je hierarchické, na nejvyšší úrovni probíhá dělení na tzv. autonomní systémy, mezi kterými probíhá směrování za pomoci EGP (*Exterior Gateway Protocol*) protokolů. Uvnitř každého z nich pak už směrujeme na základě jednotlivých sítí, což nám umožňují IGP (*Interior Gateway Protocol*) protokoly nebo statické směrování. Následující výklad se zaměřuje výhradně na směrování v rámci autonomních systémů, a to mezi sítěmi, které ke komunikaci používají protokol IPv6.

Základem je tzv. směrovací tabulka, již můžeme nalézt v každém zařízení operujícím na síťové vrstvě. Tabulka obsahuje záznamy o různých sítích a u každého z nich udává, přes kterého souseda (ten bývá označován jako *next hop*) nebo síťové rozhraní má být paket adresovaný do této sítě zaslán. Jako poslední záznam se často uvádí tzv. výchozí cesta, která používá nulový prefix `::/0`. Ten tak může být použit pro libovolnou cílovou adresu a slouží jako poslední možnost, pokud neexistuje vhodnější cesta. Tato technika bývá využívána obzvláště na koncových stanicích, kde se nechceme zabývat vytvářením směrovací tabulky pro nejrůznější sítě. Proto jednoduše vytvoříme výchozí cestu na nejbližší směrovač, který obslouží veškerou odchozí komunikaci. V tomto kontextu je takový směrovač často označován jako tzv. výchozí brána. Je zvykem, že na jednoduchých zařízeních jako jsou např. počítače, do směrovací tabulky nezasahujeme ručně, naopak v uživatelském rozhraní pouze zadáme výchozí bránu a odpovídající záznam v tabulce se vytvoří automaticky.

Výběr optimální cesty probíhá na základě hledání nejdelšího shodného prefixu, proto je výchozí cesta využita pouze pokud v tabulce nebyl nalezen žádný jiný vyhovující záznam. Součástí každého záznamu je také tzv. metrika (číselné ohodnocení dané cesty). Přesný význam této hodnoty se různí – může se v ní promítat např. počet skoků na cestě k cíli, propustnost použité linky nebo administrátorem definovaná preference cesty. Obecně však platí, že menší číslo značí lepší cestu.

V závislosti na způsobu konstrukce směrovací tabulky rozlišujeme statické a dynamické směrování. V případě statického směrování jsou záznamy vytvářeny ručně administrátorem sítě, případně základními mechanismy IP a NDP (lokální adresy, přímo připojené sítě, výchozí cesta, ...). Tento způsob obvykle plně vyhovuje pro koncové stanice, jimž k zajištění konektivity stačí znát výchozí bránu. U menších sítí lze použít i pro směrovače, je-li v silách správce vložit na každém z nich cesty pro všechny dostupné sítě.

Tento přístup však bývá v praxi doplňován o dynamické směrování, které zajišťuje některý z dynamických směrovacích protokolů. Ty automaticky upravují směrovací tabulku

na základě zjištěných informací o dostupných sítích. Umožňují také flexibilně reagovat na změny v topologii, např. výpadek linky nebo přidání sítě nové.

2.1 Charakteristika OSPF

OSPF je moderní dynamický směrovací protokol, jehož specifikaci lze nalézt v RFC 2328 [14] (OSPF verze 2, pro IPv4) a RFC 5340 [6] (OSPF verze 3, pro IPv6). Díky své hierarchické struktuře poskytuje dobrou výkonnost i ve velmi rozsáhlých sítích, a patří proto mezi nejpoužívanější směrovací protokoly typu IGP. Tomu bezesporu napomáhá i volná licenční politika a také úzká vazba na protokol IP.

Ke směrování OSPF používá algoritmus typu *link-state*, což znamená, že informace zasílané mezi směrovači se týkají výhradně jejich vzájemné dostupnosti a stavu linky. Na základě nich si pak každý směrovač sestaví virtuální mapu síťové topologie, aplikováním Dijkstrova algoritmu nalezne nejkratší cesty do každého bodu a poté vygeneruje odpovídající záznamy ve směrovací tabulce. Zatímco zprávy *distance-vector* protokolů jsou typicky zasílány v pravidelně se opakujících intervalech, důležitou vlastností *link-state* konceptu je šíření informací pouze na základě detekce změn v topologii. To zajišťuje rychlou konvergenci když taková změna nastane, ale zároveň minimalizuje množství zasílaných dat pokud je síť v klidu. OSPF také zavádí tzv. cenu linky – kvalitativní ohodnocení spoje mezi dvěma směrovači na základě propustnosti, dostupnosti, spolehlivosti, ale také např. skutečné ceny, kterou je třeba platit poskytovateli za přenášení dat. Cena linky hraje významnou roli při hledání optimálních cest směrovacím algoritmem a umožňuje administrátorovi zajistit preferování určitých linek nebo vytváření záložních spojů pro případ výpadků.

2.1.1 Členění autonomního systému

OSPF protokol operuje v rámci jednoho autonomního systému, který za účelem lepší škálovatelnosti dále dělí na tzv. oblasti. Každá oblast (*area*) má svůj 32bitový číselný identifikátor, jejich počet je tak v praxi v podstatě neomezený. Podléhají však několika pravidlům, která musí být pro správnou funkčnost směrovacího protokolu dodržena.

Základem je nutnost existence tzv. páteřní oblasti (*backbone area*) označované číslem 0. Dále je třeba zajistit, aby všechny ostatní oblasti byly k páteřní oblasti připojeny – buď přímo (fyzicky), nebo za pomoci tzv. virtuální linky. Tu lze prostřednictvím virtuálních rozhraní ustanovit mezi dvěma směrovači na hranicích společné oblasti.

V souladu s touto architekturou rozlišujeme 4 typy směrovačů v rámci jednoho autonomního systému ([14], sekce 3.3):

- ***Internal router*** (IR) je nejjednodušší příklad směrovače, jehož všechny přímo připojené sítě spadají do jedné oblasti. Na takovém zařízení pak běží jediná instance směrovacího algoritmu.
- ***Area border router*** (ABR) je naopak připojen do více oblastí a udržuje si tak instanci směrovacího algoritmu pro každou z nich. Protože musí být všechny oblasti připojeny k páteřní oblasti, je i každý ABR směrovač s páteřní oblastí spojen. Jeho hlavní funkcí je zajistit vzájemnou výměnu agregovaných informací mezi páteřní a ostatními oblastmi.
- ***Backbone router*** (BR) označujeme každý směrovač, který je alespoň jedním rozhraním (fyzickým i virtuálním) připojen k páteřní oblasti. Jedná se tedy o sjednocení množiny všech ABR směrovačů s IR směrovači páteřní oblasti.

- ***AS boundary router*** (ASBR) leží na hranici autonomního systému a jeho úkolem je proto zprostředkovávat externí směrovací informace. Všechny směrovače v rámci autonomního systému musí znát cesty k ASBR směrovačům. Postavení ASBR směrovače nijak nesouvisí s dělením na oblasti, proto v této roli může vystupovat libovolný IR i ABR směrovač.

2.1.2 Speciální typy oblastí

Některé oblasti uvnitř autonomního systému plní specifické funkce nebo se liší svou konfigurací. RFC 2328 [14] a RFC 1587 [7] proto dále zavádí následující typy:

- ***Transit area*** je označení pro oblast, přes kterou vede virtuální linka. To znamená, že jednotlivé uzly musí zajistit směrování zpráv mezi dvěma ABR směrovači propojenými virtuální linkou.
- ***Stub area*** je speciální typ oblasti, která nepřijímá směrovací informace o externích autonomních systémech od ASBR směrovačů. To vede ke snížení počtu zasílaných zpráv a paměťové náročnosti směrovacího algoritmu na jednotlivých zařízeních. Konektivita je zajištěna jednodušším způsobem, a sice výchozí cestou, která vede na jediný vstupní a výstupní bod spojující tuto oblast se zbytkem autonomního systému. Z toho plyne, že jako *stub* nelze konfigurovat ani páteřní oblast, ani žádnou z *transit* oblastí.
- ***Not-so-stubby area*** (NSSA) je zvláštní varianta *stub* oblasti, kterou jako volitelné rozšíření OSPF protokolu zavádí RFC 1587 [7]. Tento typ oblasti přijímá externí směrovací informace od svých vlastních ASBR směrovačů a distribuuje je dále, ale nekomunikuje s ostatními ASBR směrovači v jiných oblastech. Toto umožňuje žádoucí šíření informací o externích autonomních systémech, a přitom stále výrazně snižuje množství zasílaných zpráv v rámci oblasti.

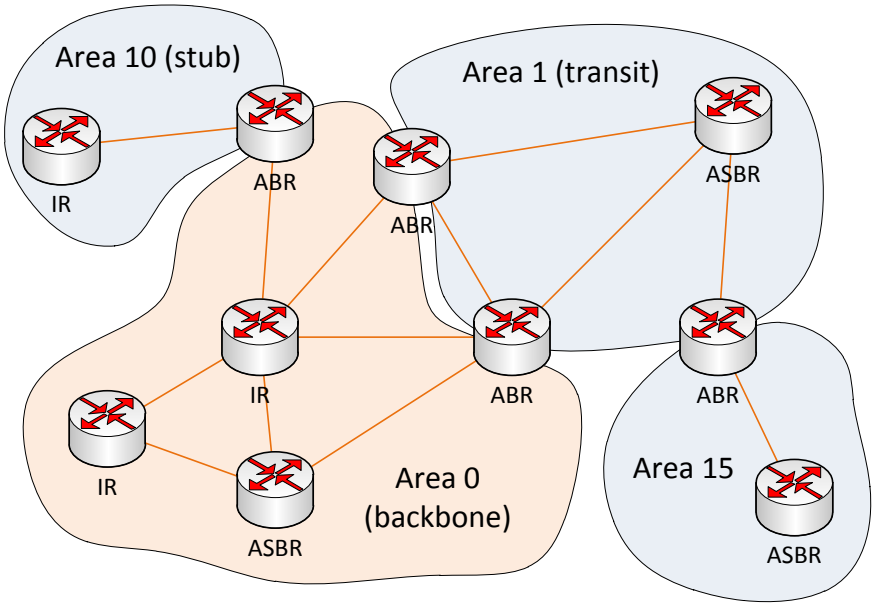
2.1.3 Komunikace mezi směrovači

Směrovače uvnitř autonomního systému jsou identifikovány pomocí unikátního 32bitového identifikátoru zvaného *router ID*. Ten se kvůli přehlednosti zapisuje ve stejné notaci jako IPv4 adresa a jedním ze způsobu přiřazení je výběr nejvyšší nebo nejnižší IPv4 adresy na zařízení ([14], sekce C.1). Lze však nastavit i manuálně na libovolnou hodnotu, která s IP adresami směrovače nemusí korespondovat.

Aby si směrovače v rámci oblasti mohly začít vyměňovat směrovací informace, musejí nejprve ustanovit tzv. sousedství. Každý směrovač si pak udržuje tabulku sousedů, jež obsahuje seznam *router ID* každého z nich a odpovídající IP adresu, přes kterou je možné souseda kontaktovat.

Pro komunikaci mezi směrovači definujeme dva druhy zpráv. Na vyšší úrovni jsou OSPF pakety s řídicí funkcí (mj. ustanovování sousedství). Uvnitř jsou pak zapouzdřeny tzv. LSA (*Link State Advertisements*), která již nesou informace o dostupných sítích a vytvářejí na směrovačích tzv. *link state* databázi. Tato databáze slouží jako zdroj informací pro vlastní směrovací algoritmus. Jak OSPF paketů tak i LSA zpráv je několik typů, kde každý z nich zajišťuje specifickou funkci. V obou případech existuje hlavička, která je společná všem, a dále tělo, jež se v závislosti na typu liší. To je důležité z toho důvodu, že některé typy OSPF paketů nenesou celá LSA, ale pouze jejich hlavičky.

Ačkoliv se implementační detaily mírně liší, obě verze OSPF protokolu používají následujících 5 typů OSPF paketů:



Obrázek 2.1: Ukázka autonomního systému s rozdělením na oblasti

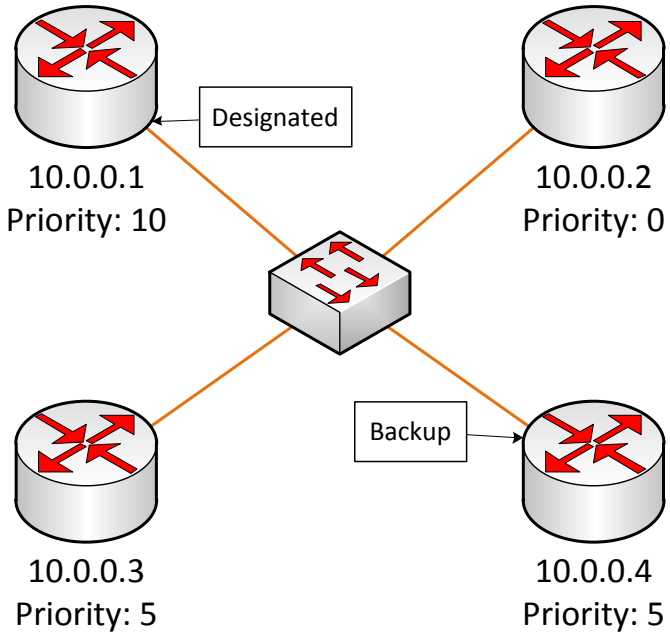
- ***Hello*** pakety jsou periodicky zasílány všem okolním směrovačům a slouží k ustanovování a udržování sousedství. Využívá se multicastové komunikace, díky čemuž je možné detekovat i nově připojené směrovače.
- ***Database Description*** pakety šíří popis *link state* databáze směrovače za pomoci zasílání hlaviček relevantních LSA. Tyto pakety jsou základem pro synchronizaci *link state* databází mezi směrovači.
- ***Link State Request*** pakety jsou typicky tvořeny v reakci na přijatý *Database Description* paket a slouží jako žádost sousedovi o zaslání konkrétních LSA.
- ***Link State Update*** paket je odpovědí na tuto žádost a nese již celá LSA. Kromě zasílání na žádost se také používají k tzv. záplavě (*flooding*), která nastává, pokud směrovač detekoval změnu v topologii sítě a chce o ní informovat ostatní.
- ***Link State Acknowledgment*** pakety slouží k potvrzování LSA přijatých pomocí *Link State Update* paketů. Díky tomu je distribuce směrovacích informací spolehlivá, protože směrovače zasílání LSA opakují, dokud k jejich potvrzení nedojde. K identifikaci odpovídajících LSA v databázi jsou opět využity jejich hlavičky.

2.1.4 Designated Router

Aby šíření informací mezi směrovači v broadcastových sítích nebylo chaotické, zavádí se tzv. *Designated Router* (DR). Navzdory názvu se nejedná o typ směrovače, ale roli, ve které směrovač vystupuje při komunikaci se sousedy dostupnými přes jedno síťové rozhraní. V každé takové skupině sousedů spojených sítí s vícenásobným přístupem pak existuje právě jeden DR směrovač, který slouží jako centrální bod pro zasílání *Database Description* paketů

a řízení záplavy *Link State Update* paketů. Tato technika zamezuje vytváření logických smyček, které by mohly vést k nekontrolovanému zahlcení sítě.

Volba DR a BDR probíhá distribuovaně za účasti všech směrovačů, hlavním faktorem určujícím vítěze je hodnota *Router Priority*, parametru nastaveného na síťovém rozhraní. V případě shody dojde k porovnání *Router ID*. Pro obě hodnoty platí, že vyšší číslo udává větší preferenci stát se DR. Pokud *Router Priority* nastavíme na 0, směrovač se DR stát nesmí. Pro účely rychlého zotavení sítě v případě výpadku DR směrovače bývá volen také záložní *Backup Designated Router* (BDR), jehož volba probíhá obdobně. Ukázku zkonvergované sítě lze vidět na obrázku 2.2.



Obrázek 2.2: Stav segmentu sítě po volbě DR a BDR

Praktická realizace volby koordinátora probíhá za pomoci OSPF zpráv typu *Hello*. Směrovač po naběhnutí začne peridicky rozesílat *Hello* pakety, zároveň je ale i přijímá od svých sousedů. V *Hello* zprávě směrovače uvádějí mj. svoji hodnotu *Router Priority* a také aktuální nastavení BR a BDR směrovače (viz obrázek 2.3).

Pokud se v síti DR již nachází, směrovač jej přijme bez ohledu na svoji prioritu. V případě že není obdržena informace o žádném DR do limitu daného časovačem *Wait Timer*, spustí se volba nového DR. Protože směrovače v tomto okamžiku již mají seznam sousedů včetně jejich *Router Priority*, všechny aplikováním algoritmu vyberou stejný DR.

Směrovače se nepokouší aktivně stát DR, dokud je stávající DR dostupný. Z toho důvodu při inspekci sítě nemůžeme vždy jednoznačně určit, který ze směrovačů vystupuje v roli DR. Nezáleží totiž pouze na jejich *Router Priority*, ale především na času jejich spuštění.

2.2 Vlastnosti OSPF verze 3

RFC 5340 [6] popisující aktuální specifikaci OSPFv3 se věnuje především změnám vzhledem k předchozí verzi. Základní principy, které zůstaly v podstatě zachovány a platí tedy pro obě verze, představila kapitola 2.1. Nyní se zaměříme na další detaily o fungování směrovacího protokolu, tentokrát již se zaměřením na OSPF verze 3 určené pro IPv6 sítě.

2.2.1 IPv6 adresace

V prostředí OSPFv2 vystupovaly unicastové IPv4 adresy v obou rolích – datové i řídicí. Protokol šířil informace o sítích a sám tyto sítě využíval při komunikaci se sousedy. IPv6 umožňuje tento model výrazně zpřehlednit – OSPFv3 totiž k unicastové komunikaci mezi sousedy využívá téměř výhradně lokální linkové adresy (výjimkou jsou pouze virtuální linky). Díky tomu protokol pracuje „na pozadí“, zcela oddělen od globálních adres a jimi daných sítí. Základní jednotkou je tedy linka (která na broadcastových IPv6 sítích propojuje i více než 2 zařízení) a nikoliv síť nebo podsíť. Při rozesílání *Hello* paketů směrovače jako zdroj uvedou právě svoji lokální linkovou adresu daného rozhraní, čímž umožní svým sousedům uložit si ji do tabulky k odpovídajícímu *router ID*.

Pro multicastovou komunikaci jsou definovány dvě IPv6 adresy – `FF02::5` s označením *AllSPFRouters* a `FF02::6` nazvanou *AllDRouters*. První z nich zahrnuje všechny OSPF směrovače na daném síťovém segmentu. Používá se například k rozesílání *Hello* paketů, nebo *Link State Update* záplavě. Pomocí *AllDRouters* pak adresujeme pouze DR nebo BDR směrovače, typicky při zasílání *Link State Acknowledgment* potvrzení.

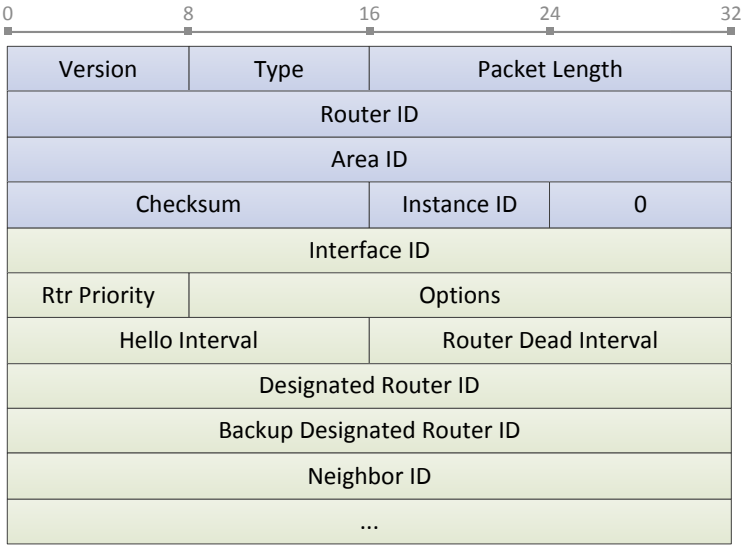
2.2.2 Zjednodušení formátu paketů

Podobně jako v případě IPv6, také OSPFv3 pakety byly s novou verzí zjednodušeny a jejich funkce zpřehledněny. Ačkoliv se zde nachází řada položek (*Router ID*, *Designated Router ID*, *Neighbor ID*, *Interface ID*, ...) z historických důvodů vypisovaných ve formátu IPv4 adresy, jedná se pouze o uměle vytvořené číselné identifikátory. Další údaje, které měly původně adresovací sémantiku (*Network Mask*, *Advertising Router*, ...), pak byly ze specifikace buď úplně odstraněny, nebo přesunuty do vhodných LSA.

Výsledkem je naprostá nezávislost OSPF paketů na architektuře síťové vrstvy, což představuje dobrý základ pro případné rozšíření protokolu mimo prostředí IPv6 sítí. Upravit by totiž stačilo pouze formát LSA a způsob zasílání zpráv mezi směrovači.

Protokol OSPFv3 také na rozdíl od svého předchůdce již nepodporuje zabezpečení prostřednictvím autentizace sousedů. Důvodem je existence nativní podpory autentizace přímo na úrovni protokolu IPv6. Naopak zcela novou funkcí, kvůli které byla do hlavičky paketů přidána položka *Instance ID*, je možnost provozovat na jedné síti více instancí OSPF směrování. Zatímco v prostředí IPv4 spolu komunikovali všichni sousedi s běžícím OSPF procesem, OSPFv3 reaguje pouze na ty pakety, jejichž číslo instance odpovídá nastavení na směrovači.

Formát *Hello* paketu je vidět na obrázku 2.3. Obdobnou filozofii lze nalézt i u dalších typů, které si ale kvůli omezenému rozsahu této práce nebudeme uvádět. Jejich formát i detailní popis funkce jednotlivých položek se nachází ve specifikaci [6].



Obrázek 2.3: Hello paket protokolu OSPFv3 (zpracováno dle [6], sekce A.3.2)

2.2.3 LSA a šíření směrovacích informací

O dostupných sítích se směrovače vzájemně informují pomocí LSA, která jsou nesena OSPF pakety typu *Link State Update*. K jejich šíření dochází typicky pomocí záplavy, kdy se zprávy lavinovitě síří mezi všechny směrovače v zadaném rozsahu, tzv. *flooding scope*. OSPFv3 definuje tyto 3 typy:

- ***Link-local scope*** – LSA jsou zasílána všem směrovačům na lince.
- ***Area scope*** – LSA jsou zasílána všem směrovačům ve stejné oblasti.
- ***AS scope*** – LSA jsou zasílána všem směrovačům v autonomním systému.

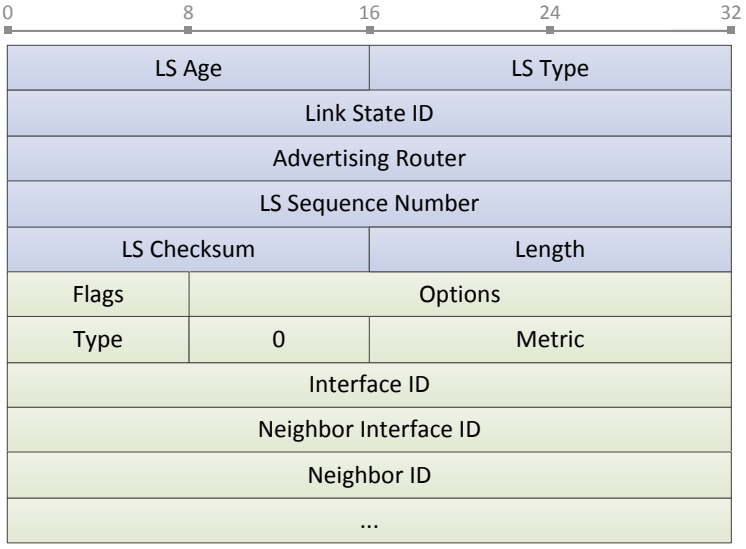
Struktura je tedy hierarchická a úzce souvisí s jednotlivými typy LSA, jejichž seznam nabízí tabulka 2.1.

Kód	Flooding scope	LSA
1	Area scope	Router-LSA
2	Area scope	Network-LSA
3	Area scope	Inter-Area-Prefix-LSA
4	Area scope	Inter-Area-Router-LSA
5	AS scope	AS-External-LSA
7	Area scope	NSSA-LSA
8	Link-local scope	Link-LSA
9	Area scope	Intra-Area-Prefix-LSA

Tabulka 2.1: Dostupné typy LSA protokolu OSPFv3

Přehled LSA a jejich použití si uvedeme pouze stručně, detailně se zabývat algoritmy pro výměnu směrovacích informací totiž široce přesahuje rámec práce. Pro účely demonstrace formátu LSA a jejich společné hlavičky slouží obrázek 2.4 zobrazující *Router-LSA*.

- ***Router-LSA*** informují o směrovačích a umožňují tak každému z nich vytvořit si virtuální mapu oblasti. Na obrázku 2.4 tak lze vidět např. položku *Metric*, která udává cenu linky zmíněnou na začátku kapitoly 2.1, nebo identifikátory *Interface ID* a *Neighbor Interface ID* popisující propojení směrovačů.
- ***Network-LSA*** jsou zasílány DR směrovači a napomáhají orientovat se v uspořádání směrovačů na broadcastových sítích.
- ***Inter-Area-Prefix-LSA*** šíří sumarizované informace o IPv6 prefixech mezi jednotlivými oblastmi v rámci autonomního systému.
- ***Inter-Area-Router-LSA*** jsou obdobou předchozího, avšak namísto IPv6 prefixů vyměňují mezi oblastmi údaje o směrovačích – například o cestě k ASBR.
- ***AS-External-LSA*** nesou informace o prefixech mimo autonomní systém.
- ***NSSA-LSA*** dle svého názvu souvisejí s *not-so-stubby* oblastmi. Stejně jako *AS-External-LSA* informují o externích prefixech od ASBR, avšak pouze v rámci oblasti.
- ***Link-LSA*** sdělují ostatním směrovačům na lince, které přímo IPv6 adresy jsou s linkou asociovány. Dále se také používají k distribuci tzv. voleb (*Options*).
- ***Intra-Area-Prefix-LSA*** doplňují *Router-LSA* o seznam IPv6 prefixů, které jsou daným směrovačům v rámci oblasti známy.



Obrázek 2.4: Router-LSA protokolu OSPFv3 (zpracováno dle [6], sekce A.4.3)

Kapitola 3

OMNeT++ a INET

Cílem projektu OMNeT++ [24] je poskytnout vývojové a simulační prostředí pro tvorbu diskrétních simulací z oblasti počítačových sítí. Základem je hierarchický systém modulů, kde každý z nich pracuje dle definovaného chování a je schopen komunikovat s ostatními zasílání zpráv prostřednictvím tzv. bran. V závislosti na požadavcích uživatele můžeme tedy navrhnout jak jednoúčelové moduly pro specifické použití, tak i moduly univerzální, které lze pak při nejrůznějších simulacích opakovaně využívat. Díky tomu není prostředí OMNeT++ z principu omezeno pouze na oblast počítačových sítí, ale lze v něm modelovat prakticky jakýkoliv problém, pro který se hodí diskrétní simulace a koncept zasílání zpráv mezi entitami.

OMNeT++ je distribuován v otevřené podobě prostřednictvím zdrojových kódů, je proto platformě nezávislý a lze jej provozovat na operačních systémech Linux, Windows i Mac OS X s plnou podporou grafického uživatelského rozhraní. Aplikace je poskytována zdarma pro akademické a nekomerční využití, v opačném případě je pak třeba zažádat o komerční licenci.

V této kapitole se také zaměříme na framework INET, který slouží jako bohatá knihovna nejrůznějších modulů z oblasti TCP/IP. Při použití v prostředí OMNeT++ tak umožňuje uživateli nezabývat se implementací elementárních protokolů a síťových zařízení, ale zaměřit svoji pozornost na vytváření samotného modelu sítě.

3.1 Projekt OMNeT++

Aplikace OMNeT++ se skládá ze dvou relativně nezávislých celků, které však mohou na první pohled splývat. Jedná se o vývojové prostředí OMNeT++ IDE a simulátor OMNeT++.

3.1.1 Vývojové prostředí OMNeT++ IDE

K modelování OMNeT++ používá dvou programovacích jazyků. Na vyšší úrovni se jedná o proprietální jazyk zvaný NED, který popisuje jednotlivé moduly, jejich parametry a vzájemné vazby. Tato část modelu není kompilována, interpretuje se přímo během simulace. Vnitřní logika těchto modulů je pak implementována v jazyce C++ a přeložena do binární podoby simulačního modelu.

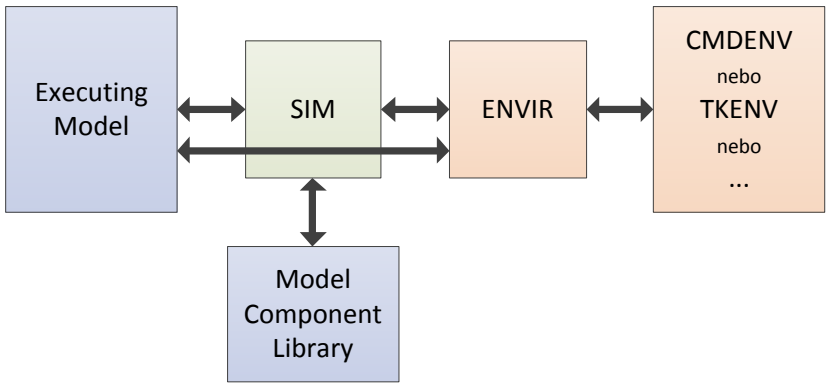
Používání vývojového prostředí při modelování nelze než doporučit, velmi totiž proces usnadňuje. V první řadě díky dostupnosti celé řady nástrojů, které se v dnešních moderních IDE obvykle vyskytují: správa projektů, zvýrazňování syntaxe, inteligentní nápověda,

integrace systémů pro správu a verzování, přímá vazba na překladač a debugger, atd. Důležitá je však také přímá podpora jazyka NED, a to včetně vizuálního módu, ve kterém je možné vytvářet simulace prostým umisťováním objektů reprezentující moduly na paletu a vytvářením spojnic mezi nimi.

I přesto však použití OMNeT++ IDE není nezbytné, implementaci v NED i C++ lze provést v jakémkoliv textovém editoru. Následný překlad pak zajistí libovolný C++ překladač, za předpokladu, že správně nakonfigurujeme sestavení programu se simulační knihovnou.

3.1.2 Simulátor OMNeT++

Architektura simulátoru byla navržena s ohledem na vysokou modularitu. To je nezbytné, protože modely vytvořené v OMNeT++ jsou při kompilaci sestavené přímo se simulační knihovnou. Tento přístup z principu umožňuje dosáhnout vyšší výkonnosti, než v případě použití interpretovaného zpracování simulace za běhu. Modulární struktura simulátoru je znázorněna na obrázku 3.1.



Obrázek 3.1: Architektura simulátoru OMNeT++ (zpracováno dle [24], kapitola 16)

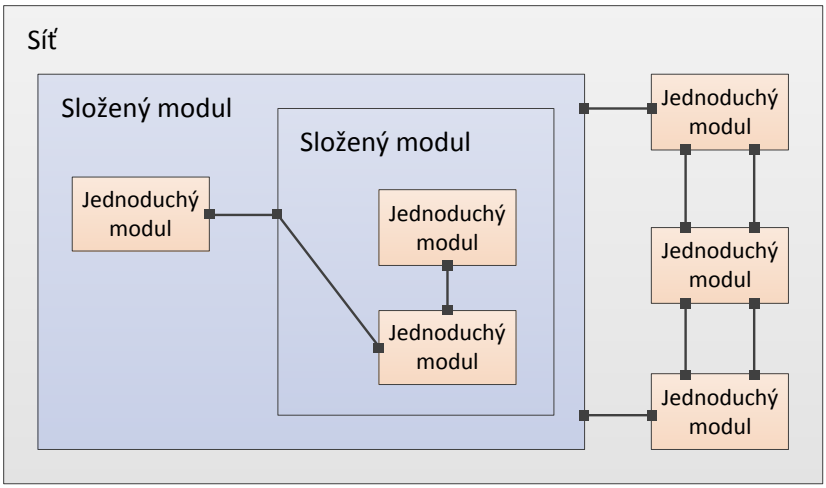
Sim je jádrem celého simulátoru, zajišťuje vytváření kalendáře diskrétních událostí, jejich vykonávání, spravuje běh jednotlivých modulů v simulaci, generuje výjimky, vytváří záznamy a další. Hlavní smyčka a startovací bod programu (funkce `main()`) se však nachází ve vedlejším bloku **Envir**. Ten prochází kalendář událostí a iniciuje jejich obsluhu, odchytává také výjimky a zapouzdřuje základní vstup a výstup programu. Vstupem se rozumí načtení konfiguračního souboru `omnetpp.ini` při startu a zpracování zadaných parametrů. K vypisování logovacích zpráv pak poskytuje rozhraní `ev`.

Model Component Library je označení pro knihovnu všech modulů, které byly v rámci projektu definovány a zkompilovány. Jejich konkrétní instance, které se v aktuální simulaci skutečně nacházejí, obsahuje **Executing Model**.

Posledním prvkem je uživatelské rozhraní simulátoru, které OMNeT++ nabízí ve dvou variantách. Pro rychlé simulace, které budeme dále analyzovat na základě vytvořených záznamů, je vhodné konzolové rozhraní **CMDENV**. Pokud naopak klademe důraz na vizualizaci simulace a možnost interaktivně řídit její průběh, sledovat přenášené zprávy a zkoumat vnitřní stavy a proměnné jednotlivých modulů, zvolíme plně grafické rozhraní **TKENV** postavené na multiplatformní knihovně Tcl/Tk.

3.1.3 Modelování v jazyce NED

Definice a spojování modulů probíhá prostřednictvím speciálního jazyka NED, který je při simulaci interpretován simulačním prostředím. Moduly můžeme rozdělit na jednoduché a složené, kde jeden složený modul zapouzdřuje několik dalších jednoduchých nebo složených modulů a propojení mezi nimi. Vzniká tak hierarchická struktura, která reflektuje reálné architektury. Na nejvyšší úrovni je pak složený modul nazývaný síť, který reprezentuje vlastní simulační model. Stupeň zanoření není ze strany vývojového prostředí nijak omezen. Tento koncept je znázorněn na obrázku 3.2.



Obrázek 3.2: Hierarchická struktura NED modulů

V jazyce NED je modul popsán ve čtyřech hlavních blocích kódu:

- `parameters:` specifikuje tzv. parametry modulu, jejichž hodnoty lze zadat ihned při deklaraci, převzít z nadřazených modulů, načíst z konfiguračního souboru, nebo vložit skrze grafické uživatelské rozhraní po spuštění simulace.
- `gates:` definuje vstupní a výstupní (případně vstupně/výstupní) brány modulu. Brány mohou být zadány také jako vektor, a to buď pevné, nebo proměnné velikosti.
- `submodules:` obsahuje výčet instancí vnořených modulů a umožňuje nastavit jejich interní parametry.
- `connections:` popisuje veškerá propojení mezi branami, a to jak vnořených modulů, tak sebe samého.

Tento výčet zjevně odpovídá pouze složeným modulům. Jednoduché totiž neobsahují submoduly ani propojení mezi nimi, jsou pro ně tedy relevantní pouze první dvě sekce.

Následuje ukázka implementace složeného modulu v jazyce NED. Jedná se o triviální směrovač, jehož parametry jsou identifikátor `deviceId` a jméno konfiguračního souboru `configFile`. Zařízení je složeno z následujících modulů: tabulka síťových rozhraní (`interfaceTable`), směrovací tabulka (`routingTable`), síťová vrstva (`networkLayer`) a skupina ethernetových rozhraní (`eth`). Modul má variabilní počet vstupně/výstupních bran,

které odpovídají jednotlivým síťovým rozhraním. Ty jsou pak dále napojeny na síťovou vrstvu. Směrovací tabulka a tabulka síťových rozhraní nejsou vybaveny branami, protože komunikují se zbytkem modulu přímým voláním metod.

```
module Router {
   parameters:
      string deviceId;
      string configFile;
   gates:
      inout ethg[];
   submodules:
      interfaceTable: InterfaceTable;
      routingTable: RoutingTable {
         parameters:
            routerId = deviceId;
            configFile = configFile;
      }
      networkLayer: NetworkLayer {
         gates:
            ifIn[sizeof(ethg)];
            ifOut[sizeof(ethg)];
      }
      e[sizeof(ethg)]: EthernetInterface;
   connections:
      for i=0..sizeof(ethg)-1 {
         ethg[i] <--> e[i].phys;
         e[i].netwOut --> networkLayer.ifIn[sizeof(pppg)+i];
         e[i].netwIn <-- networkLayer.ifOut[sizeof(pppg)+i];
      }
}
```

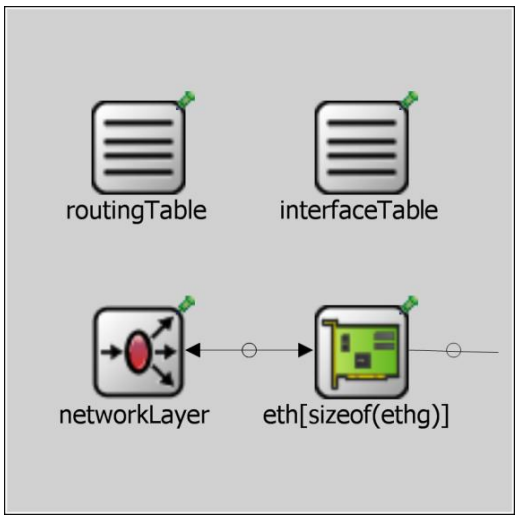
Ukázka 3.1: Implementace modulu jednoduchého směrovače

Obrázek 3.3 znázorňuje, jak se tato implementace v jazyce NED z ukázka 3.1 zobrazí ve vizuálním režimu OMNeT++ IDE.

3.1.4 Implementace jednoduchých modulů

Zatímco složené moduly jsou pouze kolekcí submodulů, jednoduché moduly mají v jazyce C++ implementovánu vnitřních logiku, která definuje jejich chování v simulaci. Ve zdrojových souborech je třeba naimportovat rozhraní simulační knihovny `omnetpp.h`, které zpřístupní prototypy nejrůznějších tříd pro moduly a zasílané zprávy a také množství pomocných maker. Zřejmě nejdůležitějším je `Define_Module()`, které provede registraci modulu a vytvoří tak vazbu s jeho NED popisem. Neméně významná jsou pak pomocná makra jako např. `par()` k načtení hodnoty NED parametru nebo `WATCH()` k exportování náhledu proměnné do uživatelského rozhraní simulátoru. K pomocným výpisům pak slouží objekt `ev`, pro který je definován vkládací operátor `<<`.

Základní třída modulu musí zdědit některý z prototypů, který modul učiní kompatibilní se simulační knihovnou. Nejjednodušším příkladem je třída `cSimpleModule`. Následně je



Obrázek 3.3: Ukázkový NED model jednoduchého směrovače

třeba implementovat virtuální metody sloužící jako vstupní bod programu. Existují dvě různé možnosti implementace, které pak ovlivňují způsob plánování činnosti modulu.

Pokud se má modul chovat jako tradiční proces, použijeme metodu `activity()`. Ta je spuštěna po inicializaci a běží až do konce simulace (nebo dokud není explicitně ukončena). Chování modulu je pak zcela v režii programátora a je třeba explicitně přijímat zprávy a ošetřit případná čekání (funkce `receive()` resp. `wait()`).

Opakem je přístup založený na událostech, který implementuje metodu `initialize()` spuštěnou při inicializaci a dále `handleMessage()` zajišťující obsluhu zpráv. Řízení v tomto případě zajišťuje simulační prostředí, které činnost modulu pozastavuje a opět spouští dle potřeby. Klíčovým mechanismem je zde zasílání zpráv – ty slouží nejen ke komunikaci s ostatními moduly, ale také k internímu plánování událostí a realizaci časovačů. Pro tento účel je definován typ zprávy *self-message*, kterou modul adresuje sám sobě a pouze zadá příkaz k jejímu odeslání v zadaném simulačním čase.

Pokud nemáme specifický důvod volit jinak, doporučuje se vždy používat přístup založený na událostech. Umožňuje totiž přehlednější návrh, který respektuje filozofii aplikovanou v celém simulačním modelu. Takto implementované moduly také většinou poskytují lepší výkon a jsou dobře škálovatelné, protože řízení jejich běhu a obsluhu zpráv je zajištěno implicitně.

3.1.5 Tvorba simulací

Konkrétní simulaci lze vytvořit implementováním složeného modulu, který nazýváme síť. Syntaxe a veškeré další náležitosti proto odpovídají popisu v kapitole 3.1.3 s jediným rozdílem, že namísto `module` použijeme klíčové slovo `network`. Pro síť také nemají význam brány, protože se nacházíme na nejvyšší úrovni hierarchie modulů.

Když vložíme všechny požadované moduly, nastavíme jejich parametry a propojíme odpovídající brány, zbývá pouze připravit rozhraní ke spušení simulátoru. K tomu slouží

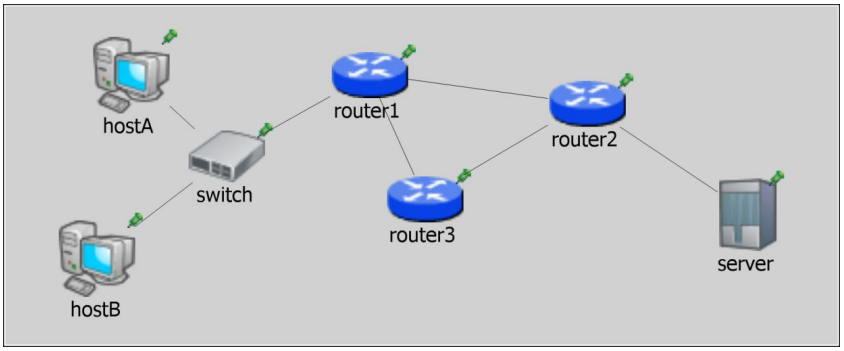
konfigurační soubor s výchozím názvem `omnetpp.ini` umístěný ve stejné složce, jako NED modul sítě. Soubor `omnetpp.ini` umožňuje specifikovat celou řadu nastavení simulátoru (délka a rychlost simulace, logování průběhu, atd.) a také definovat libovolné parametry modulů umístěných v síti. Pro základní funkcionalitu však postačí stačí přiřadit proměnné `network` jméno sítě z NED souboru (tedy např. `network = TestNetwork`). Pak stačí simulaci spustit – v grafickém uživatelské rozhraním kliknutím pravým tlačítkem myši na konfigurační soubor a zvolením *Run As OMNeT++ Simulation*.

3.2 Framework INET

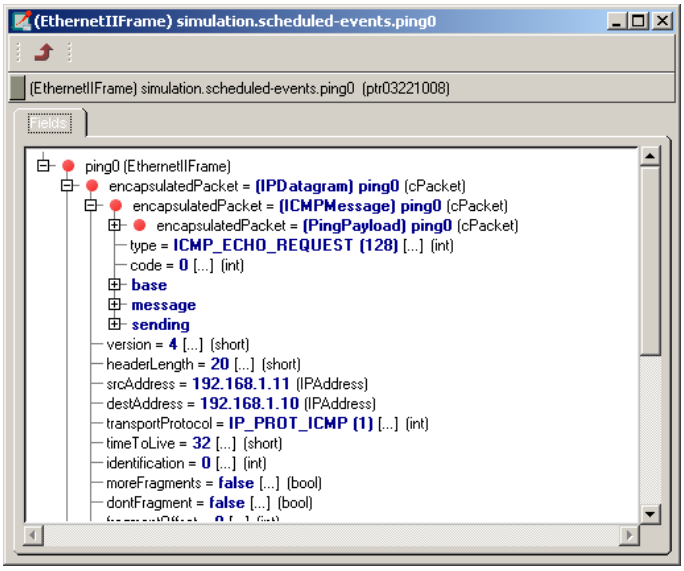
Projekt INET [3] se snaží poskytnout balík modulů, které lze v prostředí OMNeT++ použít pro snadné vytváření simulací síťové komunikace v prostředí TCP/IP. Jsou zde proto připraveny moduly reprezentující nejrůznější síťové prvky jako jsou směrovače, přepínače, bezdrátové přístupové body, pevné i mobilní stanice a další. Spousta z těchto zařízení je pak k dispozici v několika verzích pro nejrůznější potřeby simulace. Například modul počítače můžeme použít buď standardní, v rozšíření generujícím náhodný síťový provoz, nebo ve variantě umožňující zasílat uměle vytvořené TCP pakety.

Očekávané chování těchto modulů zajišťuje komplikovaná vnitřní logika. Často se jedná o univerzální moduly reprezentující např. ethernetové rozhraní, tabulku síťových rozhraní, směrovací tabulku nebo zpracování paketů na síťové vrstvě. Tyto moduly se proto ve většině výše zmíněných zařízení používají opakovaně a jsou doplněny pouze ty funkce, ve kterých se jednotlivé síťové prvky liší. Kromě poměrně abstraktních bloků popsaných výše lze nalézt i základní stavební prvky, které implementují nejrůznější síťové protokoly. Jedná se například o UDP, TCP, RTP, IP, ARP, Ethernet, PPP, ale také více „exotické“ protokoly jako MPLS, LDP, RSVP nebo OSPF.

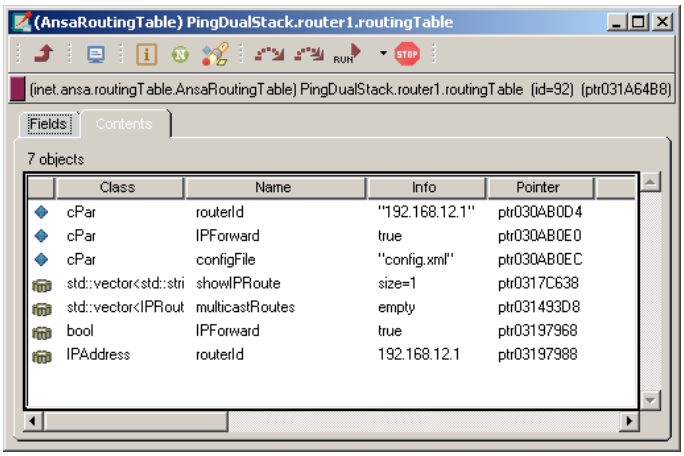
INET tak uživateli umožňuje velice snadno vytvořit novou síť, umístit do ní několik síťových zařízení (obrázek 3.5) a po krátkém nastavení spustit simulaci. Během té můžeme sledovat přenášení paketů na úrovni komunikačních protokolů (obrázek 3.5) nebo zkoumat interní stavy a proměnné jednotlivých zařízení i jejich interních modulů (obrázek 3.6). Pro modelování na této úrovni není třeba ovládat techniku vytváření nových modulů, znát jazyk C++, ani provádět opakovanou kompilaci frameworku.



Obrázek 3.4: Ukázková síť sestavená z INET modulů zařízení



Obrázek 3.5: Zobrazení obsahu ICMP paketu znázorňuje zapouzdření protokolu



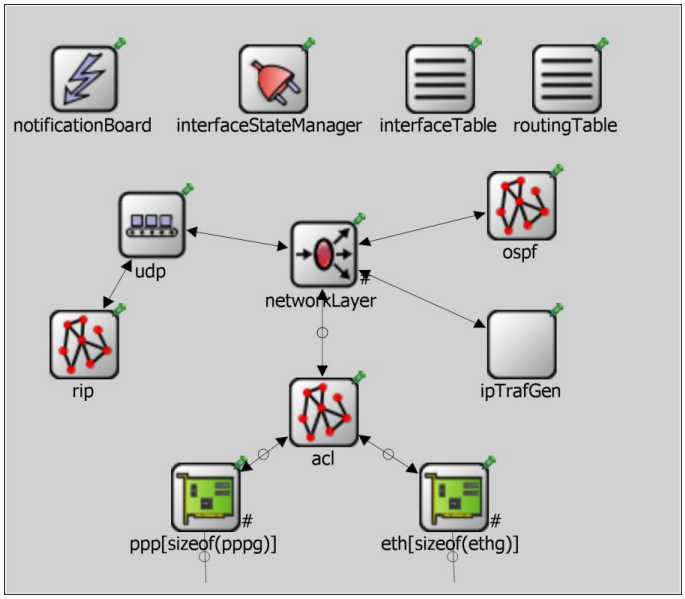
Obrázek 3.6: Inspekce modulu směrovací tabuly jednoho ze směrovačů v síti

3.3 Projekt ANSA

Projekt ANSA *Automated Network-wide Security Analysis* na FIT VUT v Brně [1] se v rámci svého dlouhodobého cíle mj. zabývá zkoumáním a rozšiřováním možností balíku INET. Ten by měl v konečném důsledku poskytnout nástroj, který umožní automatizovaně vytvářet realistické modely počítačových sítí v zadané konfiguraci. Takový model lze poté na rozdíl od fyzického zapojení algoritmicky zpracovávat, a provádět např. formální analýzu a verifikaci bezpečnosti a dostupnosti této sítě.

Nejdříve je však třeba připravit komplexní modely síťových zařízení, které budou navenek vykazovat stejné chování, jako jejich reálné vzory. Je totiž nezbytné, aby simulace dávala stejné výsledky, jako kdybychom dané zapojení provedli fyzicky v laboratoři.

Po implementační stránce tento vývoj posunula kupředu zejména skupina bakalářských prací z doby před dvěma lety. Předně byl upraven modul INET směrovače – došlo k doplnění možnosti konfigurace zařízení z XML souboru a úpravám směrovací tabulky a tabulky síťových rozhraní [19]. Do vzniklého ANSA směrovače, jehož aktuální podobu lze vidět na obrázku 3.7, byla přidána podpora ACL [21] a směrovacích protokolů RIP [17] a OSPF [8]. Existuje také práce zabývající se analýzou a návrhem implementace proprietálního protokolu EIGRP [23]. V neposlední řadě vznikl překladač, který přijímá konfigurační soubory vyexportované z fyzických zařízení Cisco a automaticky generuje ekvivalentní konfiguraci v XML formátu, který model ANSA směrovače přijímá [18]. V současnosti vznikají v rámci projektu ANSA další dvě diplomové práce. První se zaměřuje na rozšíření překladače konfigurací o podporu více konfiguračních příkazů a zařízení i jiných výrobců. Druhá se pak zabývá zcela novou implementací modulu přepínače.



Obrázek 3.7: Architektura ANSA směrovače

Kapitola 4

Současný stav

Aktuální verze frameworku INET byla uvolněna 25. února 2011 [3] a mezi hlavní aktualizace patří rozšíření podpory protokolu SCTP a příprava pro OMNeT++ 4.2 beta. Můžeme se proto domnívat, že projekt je živý a autoři se aktivně věnují jeho vývoji. To ovšem neznamená, že jsou všechny moduly zcela funkční – o tom svědčí mj. i související práce studentů ze skupiny ANSA [8, 17, 18, 19, 21, 23]. Zejména dokumentace frameworku [25], označená jako „DRAFT“, je poněkud roztříštěná a zdaleka nepopisuje všechny moduly v potřebné míře.

V této kapitole se proto zaměříme na analýzu současného stavu implementace IPv6 a dále také existujících ANSA rozšíření, která s tématem souvisejí.

4.1 INET a IPv6

Podpoře IPv6 se v manuálu ([25], kap. 10) věnuje několik odstavců stručného popisu, který představí stěžejní třídy a vazby mezi nimy. Nic ovšem neříká o implementačních detailech a dosažené funkcionalitě. Bylo proto třeba zaměřit se přímo na zdrojové soubory a jejich stav ověřit.

Ukázalo se, že celá IPv6 sekce je označena razítkem „Work In Progress“ s varováním, že některé části nejsou dokončeny a vývoj dalších momentálně probíhá. Tento popis je datován k roku 2008 a odpovídá stavu zdrojových souborů. V těch lze nalézt množství pracovních poznámek autorů, obsahují celé bloky provizorně zakomentovaného kódu, některé metody jsou definované, avšak s prázdným tělem, atd. Porovnáním souborů z různých verzí frameworku se bohužel ukázalo, že žádná z aktualizací za poslední tři roky se stavu IPv6 nedotkla.

Na základě těchto zjištění lze předpokládat, že vývoj IPv6 ve skutečnosti ustal a v brzké době tak nelze počítat s doplněním chybějících informací. Protože míra funkčnosti této rozpracované implementace je neznámá, bylo třeba provést detailní analýzu zdrojových souborů a otestovat aktuální možnosti modulu.

4.1.1 Souborová hierarchie

Třídy zajišťující IPv6 podporu lze v souladu se zavedenými konvencemi projektu INET nalézt na několika místech adresářové struktury. V následujícím přehledu si uvedeme umístění stěžejních souborů a naznačíme jejich vzájemné vazby.

`src/networklayer/ipv6/`

V této složce se nachází jádro implementace protokolu. Jednoduchý modul `IPv6` slouží jako rozhraní pro vstup a výstup datagramů, zajišťuje jejich zapouzdřování a rozbalování, výběr správného rozhraní pro odchozí pakety, zpracování ICMPv6 zpráv a další. Třídy `IPv6Datagram` a `IPv6ExtensionHeaders` popisují vlastní datagramy, obsahují všechny položky z protokolové hlavičky a poskytují také několik pomocných metod (zřetězování hlaviček, výpočet velikosti datagramu, tvorba nového datagramu, ...). Nejrůznější chybová hlášení přebírá a nadřazeným modulům prezentuje `IPv6ErrorHandling`.

Struktura `IPv6InterfaceData` sdružuje všechny informace, které se vztahují k síťovému rozhraní s IPv6 podporou. To jsou nejen přiřazené IPv6 adresy, ale také např. MTU linky, časovače pro zasílání NS, RS a RA zpráv, příznak, zda rozhraní periodicky RA zprávy zasílá (platí pro směrovač), nastavení maximálního počtu skoků při směrování a atd.

Posledním důležitým souborem je třída směrovací tabulky `RoutingTable6`. Ta umožňuje vkládat a rušit cesty s IPv6 adresami a jsou k dispozici pomocné funkce např. pro vyhledání cesty s nejdelším shodným prefixem nebo pro ověření vlastní adresy.

`src/networklayer/icmp6/`

Na tomto místě nalezneme zdrojové soubory podpůrného protokolu NDP. Jsou zde definovány struktury reprezentující jednotlivé typy ICMPv6 zpráv a mechanismy, které těchto zpráv využívají. Především tedy objevování sousedů a služba pro bezstavovou autokonfiguraci. Při objevování sousedů se postupně plní struktura mapující IPv6 adresy na odpovídající fyzické adresy, která se následně využívá při zasílání zpráv. Veškerá tato funkcionalita je integrována do dvou jednoduchých NED modulů: `IPv6NeighbourDiscovery` a `ICMPv6`.

`src/networklayer/contract/`

Tento adresář sdružuje třídy popisující IP adresy obou verzí protokolu. Právě z tohoto místa se načítá hlavičkový soubor `IPv6Address.h`, který využívá každý modul pracující s IPv6. K dispozici je také hybridní třída `IPvXAddress`, která dokáže pojmout adresu typu IPv4 i IPv6. Její použití se doporučuje zejména u modulů vyšších vrstev, které by měly být nezávislé na verzi protokolu.

Dále je k dispozici třída `IPv6ControlInfo`, která se používá jako tzv. *Control Info* při zasílání IPv6 datagramů. *Control Info* je ukazatel na libovolný objekt, který je k dispozici u každé zprávy v prostředí OMNeT++. Tento mechanismus umožňuje např. přenést dodatečné informace, které jsou nezbytné pro správné napodobení chování reálného zařízení. V případě IPv6 lze v této struktuře dohledat např. údaj o tom, na kterém síťovém rozhraní byla zpráva přijata.

`src/networklayer/common/`

Obsahem této složky jsou třídy reprezentující tabulku síťových rozhraní. Na rozdíl od směrovací tabulky v tomto případě nebylo třeba vytvářet zcela novou variantu pro protokol IPv6. Stačilo stávající strukturu upravit, aby kromě informací relevantních pro IPv4 byla schopna pojmout i data související s protokolem IPv6. Ta jsou zapouzdřena ve struktuře `IPv6InterfaceData` načítané z adresáře `src/networklayer/ipv6/`.

4.1.2 Přehled dosažené funkcionality

Ačkoliv oficiální dokumentace IPv6 modulu v podstatě neexistuje, zdrojový kód je autory poměrně hojně komentován, a to včetně nejrůznějších komentářů o existujících problémech, citací z RFC vysvětlujících některá rozhodnutí nebo *TODO* listů obsahujících seznam úkolů k doplnění. Na základě detailní analýzy zdrojových souborů si tedy lze vytvořit hrubou představu o aktuálním stavu implementace. Vše nasvědčuje tomu, že z významných funkcí IPv6 protokolu jsou plně funkční následující:

- Vytváření a zasílání IPv6 datagramů
- Příjem a dekódování IPv6 datagramů
- Podpora zřetězených hlaviček *Routing Header* a *Hop-by-Hop Header*
- NDP protokol s výjimkou funkce *Redirect*, viz kapitola 1.2
- ICMPv6 zprávy definované NDP a dále typy *Echo Request* a *Echo Reply*

Podařilo se také odvodit, které ze stěžejních mechanismů naopak podporovány nejsou buď vůbec, nebo pouze ve zjednodušené podobě:

- Fragmentace (implementace započatá, ale nedokončená)
- Multicastová komunikace (provizorně realizována broadcastovým vysíláním)
- Zřetězená hlavička *Destination Options Header*
- Další zřetězené hlavičky včetně zabezpečení IPsec

4.1.3 Podpora směrování

Zvláště se zaměříme na aktuální možnosti směrování pod IPv6. Nezbytným prvkem je směrovací tabulka, kterou implementuje třída `RoutingTable6` ze složky `src/networklayer/ipv6/`. V té je definována nejen struktura vlastní tabulky, ale také třída `IPv6Route` používána pro jednotlivé cesty s IPv6 adresací.

Při vytvoření nové cesty můžeme specifikovat všechny důležité parametry, tedy IPv6 prefix a jeho délka, *next hop* adresa, odchozí rozhraní a metrika cesty. Lze také specifikovat, co je zdrojem této cesty – statická konfigurace, bezstavová autokonfigurace nebo směrovací protokol. Cesty generované automaticky na základě příjmu *Router Advertisment* je třeba pravidelně obnovovat, uvádí se u nich proto doba platnosti.

Směrovací tabulka předně poskytuje metody pro vytvoření nové cesty ve směrovací tabulce, odebrání stávající, vytvoření výchozí cesty, vyhledání cesty s nejdelším shodným prefixem pro zadanou cílovou adresu a další. Kromě toho ale také funguje jako konfigurační centrum celého IPv6 modulu. Při inicializaci nastaví podporu IPv6 na síťových rozhraních, zajistí vygenerování lokálních linkových adres, na směrovačích spustí proces periodického zasílání *Router Advertisment* zpráv a podobně. Dále obsahuje řadu pomocných metod, které usnadňují práci při implementaci dalších modulů pro IPv6 (např. `isRouter()`, `getInterfaceByAddress()`, `isLocalAddress()`, ...).

Za pozornost stojí metoda `receiveChangeNotification()`, která má za úkol přijímat oznámení o událostech, které nastaly v nadřazeném modulu. Tato technika je v modelech zařízení obvykle využívána pro šíření informace o změně stavu síťového rozhraní – vytvoření,

smazání, přechod mezi stavy *up* a *down* nebo změna konfigurace. V `RoutingTable6` metoda sice je definovaná, avšak její tělo obsahuje pouze prázdnou `if-else` strukturu. To znamená, že směrovací tabulka pro IPv6 existuje pouze ve stavu daném po inicializací zařízení, avšak není schopná reagovat na další události vyskytnuvší se v průběhu simulace.

Ačkoliv je k dispozici směrovací tabulka schopná plnit všechny své základní funkce, v současné verzi frameworku INET není implementován žádný z dynamických směrovacích protokolů pro IPv6, ani rozhraní, které by umožnilo přidávat statické cesty.

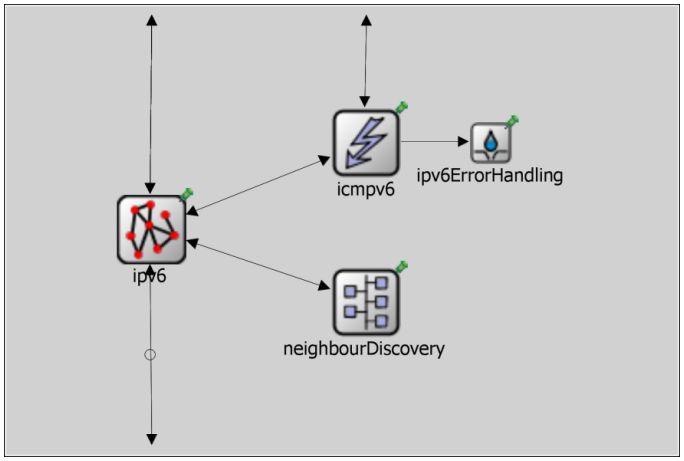
4.1.4 Složené moduly a podpůrné aplikace

Výše popsané jednoduché moduly a s nimi spojené služby tvoří stavební kameny složených modulů, které zapouzdřují implementační detaily a poskytují dostatečnou míru abstrakce, abychom je mohli prakticky použít při vytváření simulací. V souladu s konvencemi adresářové struktury projektu INET můžeme tyto bloky nalézt ve složce `src/nodes/ipv6/`. Jedná se o trojici `NetworkLayer6`, `Router6` a `StandardHost6`.

Poněkud nepřímo pak s IPv6 souvisí další 3 moduly, které se více než síťovým zařízením podobají aplikacím. Jedná se především o modul `FlatNetworkConfigurator6` umístěný v `src/networklayer/autorouting/` a dále dvojici `PingApp` a `IPTrafGen` ze složky `src/applications/`.

`NetworkLayer6`

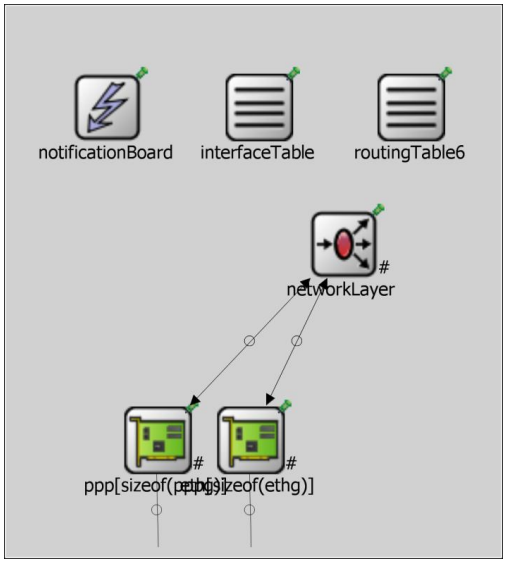
Tento modul zapouzdřuje veškerou funkcionalitu IPv6 na úrovni síťové vrstvy a v zařízeních tak figuruje jako prostředník mezi síťovými rozhraními a protokoly vyšších vrstev. Obsahuje proto řadu bran pro napojení na nejrůznější moduly uvnitř zařízení. Interní strukturu, která mj. znázorňuje provázanost IPv6 a NDP, lze vidět na obrázku 4.1.



Obrázek 4.1: Modul síťové vrstvy pro IPv6 – `NetworkLayer6`

`Router6` a `StandardHost6`

Tato dvojice stojí na nejvyšší úrovni hierarchické struktury používané projektem INET, protože se jedná o vlastní síťová zařízení. `Router6` reprezentuje směrovač, jehož schéma je znázorněno na obrázku 4.2. Obdobným způsobem je zapojen i `StandardHost6`, tedy modul zastupující osobní počítač či server. V obou případech se jedná o poměrně triviální moduly, které obsahují pouze nutné minimum pro základní IPv6 konektivitu. Nejen že zde nenalezneme protokoly vyšších vrstev, ale není zachována ani zpětná kompatibilita s IPv4.



Obrázek 4.2: Modul směrovače pro IPv6 – `Router6`

Podpůrné moduly a aplikace

V poněkud neobvyklé roli, která nekoresponduje s ničím, co známe z reálných sítí, vystupuje modul `FlatNetworkConfigurator6`. Jedná se o blok, který lze umístit kamkoliv do NED sítě (nemá totiž žádné brány a tedy ani vazby na jiné moduly), a jeho účelem je nakonfigurovat IPv6 adresaci všech ostatních zařízení. Výraz *Flat Netwok* má napovědět, že výsledkem je „plochá“ topologie, kde adresy všech zařízení spadají pod stejný prefix. Modul nemá žádné parametry a unikátnost adres je zaručena prostým inkrementováním čítače s každým dalším síťovým rozhraním.

`PingApp` a `IPTrafGen` stojí již zcela mimo základní moduly implementující IPv6 podporu. Prvně jmenovaná je aplikace pro vytváření ICMP paketů typu *Echo Request*, které se používají k otestování konektivity. `IPTrafGen` pak slouží ke generování náhodných datagramů pro účely simulace síťového toku. Oba moduly podporují jak IPv4, tak i IPv6.

Shrnutí dostupné funkcionality

Na předchozích stránkách jsme si ukázali, že i přes dojem rozpracovaného projektu poskytují IPv6 moduly frameworku INET v podstatě veškerou základní funkcionalitu a jsou připravené

k propojení s ostatními komponentami. Implementace poskytuje všechny prostředky pro ustanovení IPv6 konektivity a provádění směrování mezi sítěmi. Tyto domněnky lze snadno ověřit vytvořením několika základních simulací a sledováním chování zařízení a probíhající síťové komunikace.

Problémem je však absence vysokoúrovňových modulů, které by tyto základní stavební bloky využívaly. Obě dostupná zařízení (`Router6` a `StandardHost6`) jsou nejjednodušší možnou implementací, která nenabízí žádné implicitní prostředky ke konfiguraci. Máme tak sice zařízení s IPv6 podporou, avšak nejsme schopni přiřadit mu IPv6 adresu a veškerá komunikace se tak omezuje na dosah automaticky vygenerované lokální linkové adresy. Směrovače navíc nemohou plnit svoji základní funkci (směrování), protože nelze nijak naplnit jejich směrovací tabulky. Také exkluzivní podpora IPv6 bez zpětné kompatibility pro IPv4 zjevně neodpovídá reálným zařízením.

Částečným je použití modulu `FlatNetworkConfigurator6`, který základní nastavení zařízení provede externě. Vzhledem však k limitacím uvedeným výše nelze tento způsob použít pro modelování topologií, ve kterých je více než jedna IPv6 síť.

Tento stav budí dojem, že `Router6`, `StandardHost6` a `FlatNetworkConfigurator6` vznikly pouze jako pomocné prvky k průběžnému testování implementace IPv6 podpory. V reálných simulacích jsou však prakticky nepoužitelné. Další vývoj prezentovaný v této práci se proto bude orientovat právě na doplnění funkcí, které tato základní zařízení nejsou schopna poskytnout.

4.2 ANSA moduly

Rozšiřování frameworku INET v rámci projektu ANSA probíhá neinvazivním způsobem – veškerá přidaná funkcionalita je umístěna v modulech ze složky `src/ansa/`. Některé byly implementovány zcela od základu, většina však určitým způsobem navazuje na existující části frameworku. Typicky se jedná o základní stavební bloky zajišťující nízkoúrovňové funkce (např. protokol IP nebo síťová rozhraní typu PPP a Ethernet). Pracovní adresář ANSA také obsahuje kopie některých INET modulů (obvykle jsou označeny přidáním prefixu „ansa“ k názvu souboru a třídy), jejichž vnitřní logika byla upravena nebo doplněna.

Z pohledu této práce je stěžejním modul směrovače zvaný `ANSARouter`, jehož architekturu bylo možné vidět na obrázku 3.7. Zcela nové jsou moduly `acl` (*Access Control List*, technika umožňující filtrovat síťový provoz na základě pravidel), `rip` (implementace dynamického směrovacího protokolu RIP) a `interfaceStateManager` (pomocný modul pro zapínání a vypínání síťových rozhraní v průběhu simulace).

Původně INET moduly upravené pro účely ANSA jsou `eth` (síťová rozhraní typu Ethernet), `ipTrafGen` (generátor síťových toků), `ospf` (implementace dynamického směrovacího protokolu OSPFv2), `udp` (transportní vrstva protokolu UDP), `interfaceTable` (tabulka síťových rozhraní) a `routingTable` (směrovací tabulka pro IPv4). Zbylé prvky, tedy `ppp` (síťová rozhraní typu PPP), `networkLayer` (síťová vrstva pro IPv4) a `notificationBoard` („nástěnka“ pro hlášení o událostech), jsou importovány přímo z balíku INET.

Kapitola 5

Implementovaná rozšíření

Předchozí kapitola shrnula, do jaké míry INET IPv6 v současnosti podporuje a představila aktuální podobu ANSA směrovače. Nyní se zaměříme na návrh a implementaci dalších rozšíření, jejichž cílem je umožnit snadno vytvářet simulace počítačových sítí, které ke komunikaci protokol IPv6 využívají.

5.1 Dual-stack zařízení

Moduly síťových zařízení pro IPv6 přítomné v INET frameworku požadavkům projektu ANSA zjevně nevyhovují. Na vině je především chybějící možnost konfigurace a také neexistující zpětná kompatibilita s IPv4. Prvním krokem k nasazení IPv6 je tedy vytvoření nových modulů směrovače a klienta s *dual-stack* implementací síťové vrstvy.

Při integrování podpory IPv6 do zařízení je třeba zamyslet se nad způsobem napojení na ostatní komponenty. Jednou z možností by bylo vzít základní jednoduché moduly IPv6, doplnit je do modulu síťové vrstvy a vytvořit tak skutečně hybridní implementaci, která dokáže obsloužit pakety IPv4 i IPv6. Tento přístup by však byl v současné INET architektuře jen obtížně realizovatelný a znamenal by poměrně rozsáhlé úpravy v jednotlivých modulech.

Ukázalo se, že mnohem elegantnějším řešením je ponechat IPv4 a IPv6 uvnitř zařízení odděleně. Zaměřili jsme se tedy na integraci modulu síťové vrstvy `NetworkLayer6` do existujícího zařízení pro IPv4. INET moduly respektují princip zapouzdření, který je charakteristický pro vrstvový model TCP/IP, každá vrstva tedy funguje jako prostředník mezi vrstvou nižší a vyšší, avšak její architektura je na ostatních vrstvách nezávislá. Z toho důvodu by mělo být snadné ponechat v zařízení všechny stávající moduly a ke všem existujícím vazbám na síťovou vrstvu protokolu IPv4 vytvořit alternativy pro IPv6.

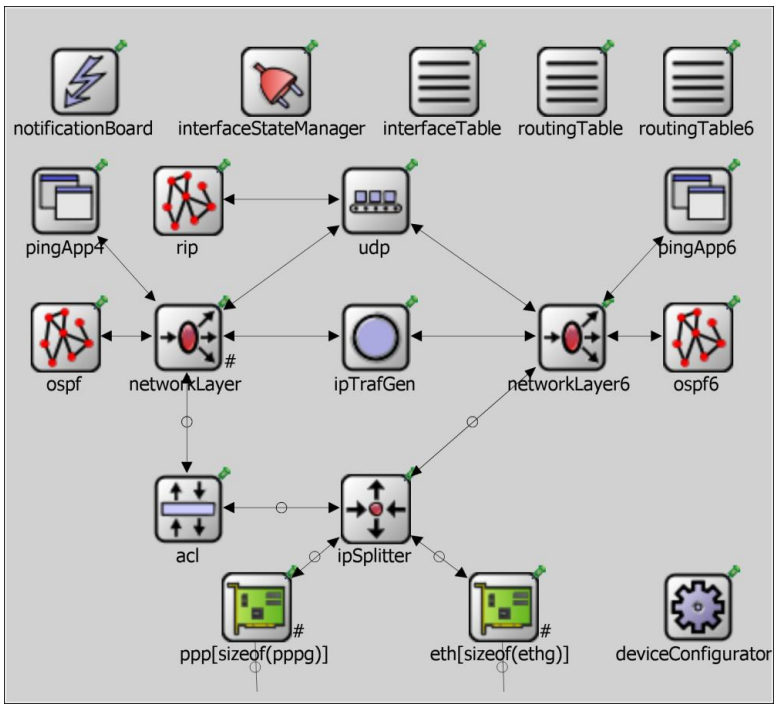
Při analýze stavu INET frameworku bylo zjištěno, že všechny moduly vyšších vrstev jsou vybaveny nejen běžně používanými branami `ipIn` a `ipOut`, ale také `ipv6In` a `ipv6Out`. Stačí tedy vytvořit nové spoje mezi těmito bránami a síťovou vrstvou pro IPv6, správné směrování komunikace již zajistí vnitřní logika daných modulů.

Komunikace s nižší vrstvou je složitější, jelikož z pohledu linkové vrstvy jsou obsahem rámců data a není proto možné rozlišit mezi datagramy typu IPv4 a IPv6. K tomuto účelu byl proto navržen a implementován modul `IpSplitter`, který lze umístit mezi síťová rozhraní a síťovou vrstvu. Tento modul zkoumá příchozí data a na základě identifikace typu datagramu provede jeho přeposlání bránou spojující `IpSplitter` s odpovídající síťovou vrstvou. Odchozí komunikace je směrována na síťová rozhraní přímo, bez ohledu na verzi IP protokolu.

Kromě modulu síťové vrstvy je do zařízení třeba umístit také směrovací tabulku pro IPv6 – `RoutingTable6`. Naopak tabulka síťových rozhraních `InterfaceTable` umožňuje pojmout konfiguraci jak protokolu IPv4, tak i IPv6 a není proto třeba vkládat další instanci modulu.

5.1.1 Modul směrovače

Protože je při nasazení IPv6 nezbytné zachovat podporu všech dosavadních ANSA rozšíření, jako referenční implementace posloužil modul ANSA směrovače (viz obrázek 3.7). Po aplikování výše uvedených úprav tak vznikl ANSA *dual-stack* směrovač (modul s názvem `AnsaDualStackRouter`), který je schopen komunikovat pomocí IPv4 i IPv6 zároveň. Jeho architekturu znázorňuje obrázek 5.1, klíčovým prvkem je zjevně `ipSplitter` realizující napojení obou modulů síťové vrstvy.



Obrázek 5.1: Architektura ANSA dual-stack směrovače

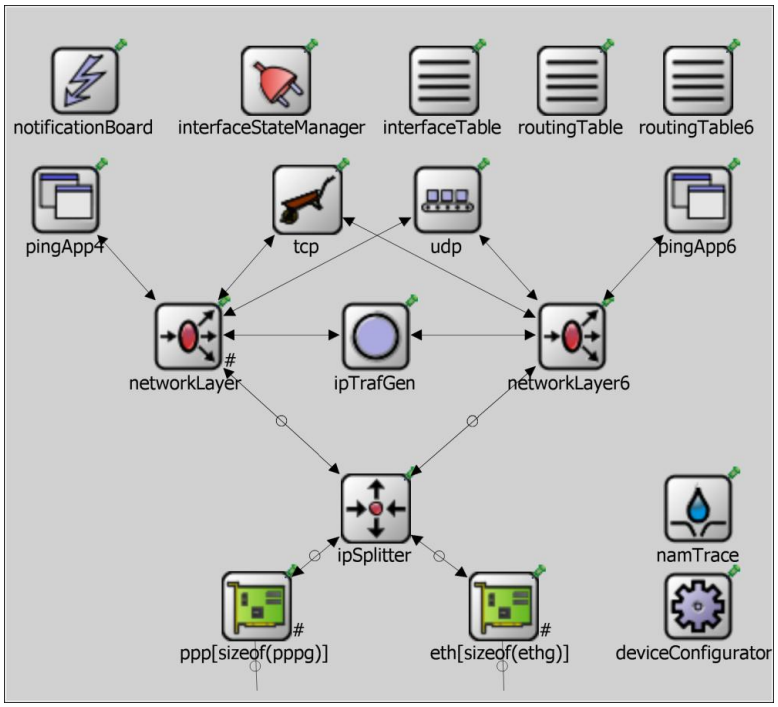
5.1.2 Modul klienta

Podobným způsobem byl sestaven i modul `AnsaDualStackHost`, který postrádá některé pokročilé funkce dostupné ve směrovačích (obrázek 5.2). Na první pohled lze vidět, že chybí podpora směrovacích protokolů a modul `ACL`. Naopak nově je dostupný transportní protokol TCP, díky kterému lze simulovat spolehlivou komunikaci mezi dvěma klienty. Pomocný modul s názvem `namTrace` umožňuje vytváření záznamů o událostech a přenesených datech.

Nejzásadnějším rozdílem je ovšem výchozí mód, ve kterém pracují směrovací tabulky. Pochopitelně i koncové stanice směrovacích tabulek využívají, avšak pouze pro odesílání svých vlastních dat. Příchozí pakety adresované jinému zařízení tak nejsou směrovány, nýbrž

jednoduše zahozeny. Operační systémy osobních počítačů se v tomto chovají stejně. Pokud bychom však chtěli toto výchozí nastavení změnit, lze tak učinit prostřednictvím parametrů `IPForward` (pro IPv4) a `isRouter` (pro IPv6).

V případě IPv6 typ zařízení dále ovlivňuje činnost protokolu NDP. Zatímco klienti opakovaně zasílají zprávy *Router Solicitation* pro získání informací o síti, účelem směrovačů je naopak na tyto dotazy odpovídat prostřednictvím *Router Advertisement*.



Obrázek 5.2: Architektura ANSA dual-stack klienta

5.2 Konfigurační rozhraní

Aby bylo možné vytvářet užitečné simulace, je třeba mít možnost dle potřeby konfigurovat chování jednotlivých zařízení. Framework INET je v této oblasti poněkud nekonzistentní, existující moduly klientů načítají soubor s příponou `irt`, který je svou strukturou podobný konfiguračním souborům známým z prostředí současných operačních systémů (`cfg`, `ini`, `rc`, ...). Naopak směrovače a s nimi související moduly ke konfiguraci používají struktury značkovacího jazyka XML.

Jedním z aktuálních cílů projektu ANSA je unifikovat konfigurační rozhraní pro všechna zařízení, a to právě s využitím souborů ve formátu XML. V rámci přidružené diplomové práce také probíhá vývoj překladače, který bude načítat konfiguraci reálných zařízení a automaticky generovat odpovídající XML strukturu použitelnou pro ANSA zařízení.

V současném ANSA modulu směrovače pro IPv4 lze prostřednictvím XML nastavit adresaci, parametry síťových rozhraní, statické směrování, protokoly RIP a OSPF, ACL a další. Jednoduchou konfiguraci demonstruje ukázka 5.1. Každý směrovač je identifikován

pomocí `id`, které musí korespondovat se stejnojmenným parametrem NED modulu zařízení. Tato vazba je důležitá, protože v jednom XML souboru se může nacházet konfigurace pro celou síť, tedy více než jeden směrovač. Parser proto musí být schopen určit, která část XML stromu se vztahuje ke kterému zařízení. Podobná pravidla je třeba dodržet i v případě jednotlivých síťových rozhraní.

5.2.1 XML knihovna

Vlastní načítání souboru probíhá za pomoci knihovních funkcí prostředí OMNeT++, které usnadňují parsování XML struktury. Kód pro načítání není v rámci směrovače nijak centralizovaný, v podstatě každý z dílčích modulů obsahuje vlastní implementaci parseru ke zpracování relevantní části konfigurace. Jedná se o dobrý přístup, který podporuje nezávislost modulů a činí architekturu systému poměrně přehlednou. Jeho realizace je však v současnosti poněkud těžkopádná, protože každý modul obsahuje vlastní kód např. pro vyhledání značky `<Router id="abcd">`. Pokud bychom tedy provedly změny v návrhu XML struktury nebo chtěli dané moduly použít i v jiných zařízeních než je směrovač, nezbývá než ručně opravit kód každého jednotlivého parseru.

```
<Router id="192.0.2.1">
   <Hostname>R1</Hostname>
   <Interfaces>
      <Interface name="eth0">
         <IPAddress>192.0.2.1</IPAddress>
         <Mask>255.255.255.0</Mask>
         <Bandwidth>100</Bandwidth>
         <Duplex>auto</Duplex>
      </Interface>
   </Interfaces>
</Router>
```

Ukázka 5.1: Základní konfigurace pro IPv4

Z tohoto důvodu byla vytvořena knihovna `xmlParser`, která obsahuje sadu statických metod pro usnadnění načítání různých částí konfigurace. V současnosti se jedná o tyto metody, avšak počítá se s doplňováním dalších dle potřeby:

- `GetDevice(char *deviceType, char *deviceId, char *configFile)`
- `GetInterface(cXMLElement *iface, cXMLElement *device)`
- `GetStaticRoute6(cXMLElement *route, cXMLElement *device)`
- `GetOspfProcess6(cXMLElement *process, cXMLElement *device)`
- `GetIPv6Address(cXMLElement *addr, cXMLElement *iface)`
- `GetAdvPrefix(cXMLElement *prefix, cXMLElement *iface)`

Použití tohoto rozhraní má několik příznivých důsledků. V první řadě odstraňuje nepříjemnou redundanci v různých implementacích parseru – všechny zúčastněné moduly použijí jednotný kód pro vyhledání určité značky. Odpovídající metodu lze kdykoliv upravit a centralizovaně tak změnit možnosti konfiguračního souboru. Tento postup znázorňuje hned první uvedená metoda `GetDevice()`, která slouží k otevření konfiguračního souboru a nalezení kořenové značky zařízení. Byla však rozšířena tak, aby umožňovala hledat nejen `<Router>`, ale v podstatě libovolné zařízení – v praxi tedy např. `<Host>` nebo `<Switch>`. Dalším přínosem použití knihovny je odstínění rutinního kódu k procházení XML stromu, implementace parseru v daném modulu je pak přehlednější a zabývá se již v podstatě jen zpracováním získaných parametrů.

5.2.2 Konfigurační modul

Ačkoliv současná architektura předpokládá, že každý modul provede načtení své relevantní části konfigurace, u každého zařízení existuje sada obecných příkazů, které nelze jednoznačně přiřadit konkrétnímu modulu. Některé z nich lze vidět v ukázce 5.1, jedná se především o parametry platné pro celé zařízení (`id`, `hostname`), ale také základní nastavení rozhraní a např. statického směrování. Na realizaci těchto mechanismů totiž spolupracuje hned několik dílčích modulů (síťová vrstva, tabulka síťových rozhraní, směrovací tabulka).

V aktuálním ANSA směrovači pro IPv4 lze tento parser nalézt v upraveném modulu směrovací tabulky. Pro účely IPv6 jsme se rozhodli vytvořit modul nový, který bude sloužit jako centrální bod pro obecnou konfiguraci zařízení. Na obrázcích *dual-stack* směrovače (5.1) a klienta (5.2) jej můžeme nalézt pod názvem `DeviceConfigurator`. Modul v současnosti zajišťuje načtení IPv6 adresace, parametrů síťových rozhraní, nastavení výchozí brány pro klienty a vkládání statických cest do směrovací tabulky.

Druhou významnou funkcí je možnost externě měnit parametry ostatních modulů. Například současná implementace NDP protokolu automaticky zapne rozesílání zpráv *Router Advertisement*, pokud je hodnota parametru `isRouter` kladná (tzn. ve všech směrovačích). Tento stav je ale v rozporu se specifikací, která udává, že generování RA je aktivováno pouze na základě explicitního zásahu administrátora systému (RFC [15]) 4861, sekce 6.2.1). Obdobným způsobem lze zasahovat do řady dalších proměnných.

Modul `deviceConfigurator` nám tak umožňuje změnit výchozí chování všech modulů v zařízení, které poskytují potřebné rozhraní, aniž bychom do těchto modulů zasahovali a museli měnit jejich kód. To je užitečné zejména v případě modulů importovaných z frameworku INET, jejichž úpravy nejsou žádoucí.

Ve spojení s knihovnou `xmlParser` tak `deviceConfigurator` představuje jednoduché řešení, jak integrovat podporu konfigurace z XML v podstatě do libovolného zařízení. Toho bylo využito při sestavování modulu klienta – `AnsaDualStackHost` se tedy na rozdíl od svých předchůdců již nekonfiguruje pomocí irt souborů, ale stejným způsobem jako moduly směrovačů. Jediným rozdílem v XML struktuře je použití značky `<Host>` namísto `<Router>`.

5.3 IPv6 adresace a statické směrování

Současná implementace INET modulu tabulky síťových rozhraních plně podporuje použití parametrů relevantních pro IPv6. Je také k dispozici sada metod, které umožňují hodnoty těchto proměnných získat a nastavit. Přiřazení IPv6 adresy na rozhraní je tak otázka zavolání vhodné metody – například na základě načtení odpovídající XML značky v konfiguračním souboru modulem `deviceConfigurator`.

Při návrhu XML struktury popisující parametry pro IPv6 nám jako referenční literatura posloužila příručka pro konfiguraci IPv6 v prostředí Cisco IOS [5]. Kompletní seznam dostupných XML značek lze nalézt v příloze B. V této kapitole si uvedeme pouze několik ukázkových příkladů pro demonstraci použité metodiky.

Pokud chceme prvnímu ethernetovému rozhraní přiřadit IPv6 adresu `2001:db8::1/64`, v prostředí Cisco IOS bychom použili sekvenci příkazů z ukázky 5.2.

```
(config)# interface FastEthernet0/0
(config-if)# ipv6 address 2001:db8::1/64
```

Ukázka 5.2: Přiřazení IPv6 adresy rozhraní v prostředí Cisco IOS

Zavedená XML struktura již obsahuje sekci popisující síťové rozhraní na základě uvedení jeho názvu (viz ukázka 5.1), stačí tedy zavést novou značku pro vlastní IPv6 adresu. Výslednou strukturu znázorňuje ukázka 5.3.

```
<Interface name="eth0">
   <IPv6Address>2001:db8::1/64</IPv6Address>
</Interface>
```

Ukázka 5.3: Přiřazení IPv6 adresy rozhraní pomocí konfiguračního XML

Modul tabulky síťových rozhraní v současnosti nepodporuje manuální přiřazení žádných dalších typů IPv6 adres. Lokální linkové adresy jsou vygenerovány automaticky při inicializaci rozhraní, stejně tak je povolena možnost autokonfigurace na základě *Router Advertisement* zprávy. *Anycast* adresace implementována není. Žádné další příkazy nebo jejich varianty pro manuální nastavení adresy proto nejsou k dispozici.

5.3.1 Automatická adresace

Síťovému rozhraní však může být IPv6 adresa přidělena i automaticky prostřednictvím techniky bezstavové autokonfigurace adres, kterou poskytuje protokol NDP. Chování klientů je v tomto případě pevné a nelze měnit (v souladu se specifikací [15], sekce 6.3.1), veškeré nastavení se tudíž provádí na směrovačích ([15], sekce 6.2.1).

Pro použití autokonfigurace je nejprve třeba zapnout zasílání *Router Advertisement* zpráv na daném rozhraní. Dále se specifikuje seznam prefixů, o který má směrovač informovat, včetně informace o době jejich platnosti nebo nastavení preference prefixů. V ukázce 5.4 je zobrazena sekvence příkazů z prostředí Cisco IOS, která povolí generování RA paketů, které budou informovat o prefixu `2001:db8::/64` s platností 30 dní (2592000 sekund) a dobou preference 7 dní (604800 sekund). Odpovídající XML strukturu, kterou je modul `DeviceConfigurator` schopen zpracovat, znázorňuje ukázka 5.5.

```
(config)# interface FastEthernet0/0
(config-if)# no ipv6 nd ra suppress
(config-if)# ipv6 nd prefix 2001:db8::/64 2592000 604800
```

Ukázka 5.4: Autokonfigurace v prostředí Cisco IOS

```
<Interface name="eth0">
   <NdpAdvSendAdvertisements>yes</NdpAdvSendAdvertisements>
   <NdpAdvPrefix valid="2592000" preferred="604800">
      2001:db8:a::/64</NdpAdvPrefix>
</Interface>
```

Ukázka 5.5: Autokonfigurace pomocí konfiguračního XML

Analogicky jsou zpracovány i další příkazy pro konfiguraci NDP protokolu na rozhraní, jejich přehled se nachází v příloze B.

5.3.2 Podpora směrování

Modul směrovací tabulky pro IPv6 poskytované frameworkem INET poskytuje drtivou většinu funkcí nutných pro zajištění směrování v síti a pro účely konfigurace existují metody pro vkládání a rušení cest s danými parametry. Několik doplňkových mechanismů však chybí a bylo proto nutné modul rozšířit.

Předchozí práce projektu ANSA zavedly zvyk, že se v těchto případech zkopírují potřebné soubory, k identifikátorům tříd se přidá prefix „ansa“ a takto zklonovaný modul se upraví podle potřeby. Tento přístup je sice funkční a dodržuje zásadu o neinvazivních rozšířeních, avšak vytváří závislost na jedné konkrétní verzi zkopírovaného modulu. Implementované rozšíření se tak uzavře vůči potenciálním aktualizacím frameworku INET.

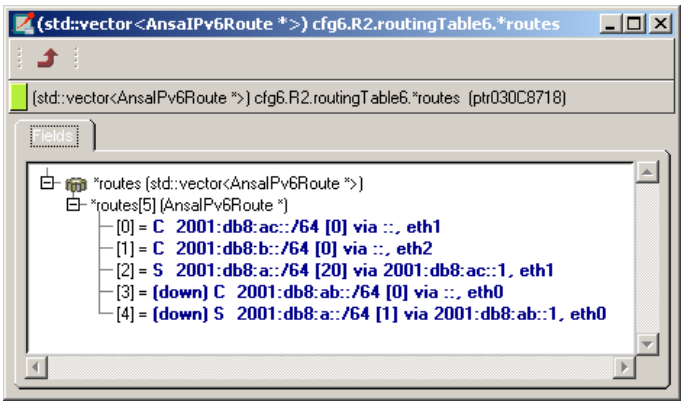
Z tohoto důvodu byl v našem případě zvolen odlišný postup využívající možností jazyka C++. Nově vytvořená třída `AnsaRoutingTable6` totiž původní `RoutingTable6` pouze dědí a redefinuje jenom ty metody, jejichž funkcionalitu je třeba upravit, případně definuje metody nové. Oblast uzavřenosti vůči INET aktualizacím se tak zmenší z celého souboru na několik vybraných metod, což je bezesporu přínosné.

Rozšíření směrovací tabulky

V současné implementaci směrovací tabulky se všem cestám přiřadila metrika s hodnotou 0, nebylo tak možné vytvářet komplexní topologie se záložními linkami a přesně definovaným směrováním. Tento nedostatek byl opraven a pro záznamy o přímo připojených sítí, které jako jediné metriku 0 skutečně používají, byla zavedena nová metoda `AddDirectRoute()`. Tu využívá `DeviceConfigurator` během přiřazování IPv6 adres síťovým rozhraním.

Druhým důležitým rozšířením je implementace metody `receiveChangeNotification()` a registrace modulu pro odebírání informací o událostech z „nástěnky“ `NotificationBoard`. Díky tomu směrovací tabulka dokáže zaregistrovat změnu stavu síťových rozhraních a na základě toho upravit své záznamy. Do struktury popisující cestu byla proto přidána proměnná popisující její „živost“. Pokud pak přijde zpráva o vypnutí síťového rozhraní, všechny cesty toto rozhraní používající se změní na neaktivní. V případě, že se rozhraní opět zapne, proběhne opačný proces.

V souvislosti s výše popsanými rozšířeními došlo také k úpravě výpisu směrovací tabulky. Hlavní motivací je pochopitelně zobrazení hodnoty metriky a indikátoru živosti záznamu. Při této příležitosti však došlo i ke změně formátu tabulky za účelem zlepšení přehlednosti a přiblížení se stylu, se kterým se můžeme setkat v prostředí Cisco IOS. Ukázku výsledné podoby výpisu směrovací tabulky lze vidět na obrázku 5.3.



Obrázek 5.3: Nový formát výpisu směrovací tabulky pro IPv6

Směrovací tabulky na zařízeních Cisco zobrazují pouze nejlepší cesty do každé sítě. Pokud tedy existují alternativní záznamy s vyšší metrikou, nebo cesty momentálně neaktivní, ve výpisu směrovací tabulky nejsou uvedeny. Implementace `AnsaRoutingTable6` se v tomto liší, protože uvádí vždy všechny cesty. Důvodem je fakt, že modelování a simulace sítí poměrně úzce souvisí s jejich laděním a hledáním potenciálních problémů. Z pohledu administrátora je tak zjevně žádoucí mít k dispozici kompletní přehled důležitých údajů, mezi které bezesporu patří i informace o tom, že některé existující cesty nejsou momentálně aktivní.

Konfigurace statického směrování

Rozhraní pro vkládání statických cest do směrovací tabulky pro IPv6 poskytuje opět modul `DeviceConfigurator` prostřednictvím konfigurační XML struktury. Jako příklad si definujme cestu do sítě `2001:db8:b::/64` přes přímo dostupného souseda s IPv6 adresou `2001:db8:a::1`. Jako metriku uveďme hodnotu 15. V prostředí Cisco IOS lze tuto cestu do směrovací tabulky přidat pomocí jediného příkazu, který zobrazuje ukázka 5.6.

```
(config)# ipv6 route 2001:db8:b::/64 2001:db8:a::1 15
```

Ukázka 5.6: Vytvoření statické cesty v prostředí Cisco IOS

Pro směrování nad IPv6 jsme do XML struktury zavedly novou značku `<Routing6>`, která zapouzdřuje jak definici statických cest, tak i nastavení dynamických směrovacích protokolů pro IPv6. Ukázka 5.7 odpovídá vytvoření statické cesty z příkladu uvedeného výše.

Kvůli omezeným možnostem směrování koncových stanic se nepředpokládá, že se při jejich konfiguraci budeme věnovat vytváření statických cest. Aby však klient mohl odesílat pakety s libovolnou cílovou adresou, je nutná existence výchozí cesty s nulovým prefixem. Ačkoliv je takovou cestu možné zadat i manuálně, pro přehlednější zápis XML struktury klientů byla zavedena značka pro specifikaci IPv6 adresy výchozí brány. Značka není zanořena v sekci pro směrování, ale vkládá se přímo na nejvyšší úroveň XML struktury pro dané zařízení. Použitou syntaxi lze vidět v ukázce 5.8.

```
<Routing6>
   <Static>
      <Route>
         <NetworkAddress>2001:db8:b:/64</NetworkAddress>
         <NextHopAddress>2001:db8:a::1</NextHopAddress>
         <AdministrativeDistance>15</AdministrativeDistance>
      </Route>
   </Static>
<Routing6>
```

Ukázka 5.7: Vytvoření statické cesty pomocí konfiguračního XML

```
<DefaultRouter>2001:db8:a::1</DefaultRouter>
```

Ukázka 5.8: Volba výchozí brány pomocí konfiguračního XML

5.4 Dynamický směrovací protokol OSPFv3

Nejen v reálných sítích, ale i při experimentování s případovými studiemi v laboratoři je některý z dynamických protokolů obvykle součástí síťové konfigurace. INET však žádný ve verzi pro IPv6 v současnosti nepodporuje. V rámci této práce byla proto zahájena implementace OSPFv3, moderního směrovacího protokolu, se kterým se můžeme setkat ve většině dnešních počítačových sítí s IP architekturou.

Framework INET obsahuje implementaci protokolu OSPFv2 určeného pro IPv4. Tento modul byl dále analyzován, upraven a rozšířen jednou z bakalářských prací projektu ANSA [8]. Vzhledem ke komplexnosti protokolu se jedná o poměrně rozsáhlý projekt (necelých 12 tisíc řádků zdrojového kódu) a to i přesto, že nejsou implementovány zdaleka všechny funkce dle specifikace (chybí např. podpora více směrovacích procesů, NSSA oblastí, sumarizace nebo redistribuce).

OSPFv3 popisuje včetně řady implementačních detailů RFC 5340 [6], jedná se však především o výčet novinek a změn, které vzhledem k předchozí verzi nastaly. To z toho důvodu, že drtivá většina mechanismů protokolu se nezměnila a bylo by to proto zbytečné vytvářet zcela novou specifikaci. Na základě analýzy dostupných zdrojových souborů jsme se proto rozhodli postupovat obdobným způsobem a tedy přesně podle pokynů v RFC upravit existující implementaci OSPFv2 na OSPFv3.

5.4.1 Kostra modulu a XML konfigurace

Směrovací proces je reprezentován instancí třídy `Router`. Kolem ní je však vystavěna struktura tvořící NED modul, která zajišťuje rutinní služby jako je načtení konfiguračního souboru, vytvoření směrovacího procesu, přeposílání OSPF zpráv a další. V případě OSPFv3 se jedná o nově implementovaný modul `AnsaOspfRouting6`.

Konfigurace OSPF

V případě návrhu konfiguračního rozhraní protokolu OSPF jsme opět vycházeli ze sady příkazů používaných v prostředí Cisco IOS [2]. Zásadní změnou v konfiguraci OSPFv3 na zařízeních Cisco je přesunutí většiny příkazů na jednotlivá síťová rozhraní. Na ukázce 5.9

je nejprve vytvořen směrovací proces číslo 1 se zadaným *Router ID*. Dále je specifikována síť `192.51.100.0/24` i s jejím zařazením odpovídající oblasti, o které bude proces šířit směrovací informace.

```
(config)# router ospf 1
(config-router)# router-id 192.0.2.1
(config-router)# network 198.51.100.0 0.0.0.255 area 0
```

Ukázka 5.9: Konfigurace OSPF pro IPv4 v prostředí Cisco IOS

V souvislosti s použitím lokálních linkových adres ke komunikaci a novému *link-local scope* je protokol OSPFv3 daleko víc vázán na konkrétní síťová rozhraní namísto celého směrovače. Na základě toho došlo i k úpravě způsobu konfigurace tohoto protokolu v prostředí Cisco IOS. Jednoduchý příklad znázorňuje ukázka 5.10.

```
(config)# ipv6 router ospf 1
(config-router)# router-id 192.0.2.1

(config)# interface FastEthernet0/0
(config-if)# ipv6 ospf 1 area 0 instance 0
```

Ukázka 5.10: Konfigurace OSPF pro IPv6 v prostředí Cisco IOS

Tedy namísto toho, abychom definovali sítě zúčastňující se směrování, zvolíme pouze síťové rozhraní a přiřadíme jej konkrétnímu OSPF procesu, oblasti a instanci. Protokol pak automaticky načte prefixy všech globálních IPv6 adres, které se na danému rozhraní vyskytují.

Na základě vybraných příkazů byla opět navrhnuta odpovídající XML struktura, kterou dokáže ANSA modul protokolu zpracovat. V ukázce 5.11 můžeme vidět jednoduchou konfiguraci odpovídající příkladům výše. Na konkrétním rozhraní lze dále specifikovat řadu dalších parametrů, např. typ a prioritu rozhraní, cenu linky, časovače OSPF paketů typu *Hello* a další. Jejich kompletní přehled lze nalézt v příloze C.

Implementace parseru XML souboru je realizována s využitím knihovny `xmlParser` a oproti svému předchůdci přináší jednu zásadní změnu. V dosavadních ANSA modulech je parametr `id` ve značce `<Router>` používán nejen jako pomocný identifikátor zařízení v XML souboru, ale také jako *Router ID* směrovacích protokolů. Tento přístup ovšem neodpovídá reálným zařízením, na kterých mohou mít různé *Router ID* přiřazeny nejen různé směrovací protokly, ale dokonce i různé procesy jediného protokolu. Z toho důvodu byla zavedena nová značka `<RouterId>`, kterou lze použít opakovaně s různou hodnotou pro každý směrovací proces.

V důsledku proto původní parametr `id` značky `<Router>` v prostředí IPv6 konfigurace ztrácí veškerý svůj sémantický význam a je využíván pouze pro identifikaci zařízení v XML souboru. Díky tomu se hodnota parametru neomezuje na IPv4 adresu, ale může jí být v podstatě libovolný řetězec. Ten obvykle volíme tak, aby zpřehledňoval popis simulace.

Správa směrovacích procesů

Nová implementace XML parseru a změna pojetí parametru *Router ID* umožnila realizovat další novinku, a to podporu více směrovacích procesů. V původní implementaci je OSPF

```
<Interfaces>
   <Interface name="eth0">
      <OspfProcess6>1</OspfProcess6>
         <OspfArea6>0</OspfArea6>
         <OspfInstance6>0</OspfInstance6>
   </Interface>
</Interfaces>
<Routing6>
   <Ospf>
      <Process id="1">
         <RouterId>192.0.2.1</RouterId>
      </Process>
   </Ospf>
</Routing6>
```

Ukázka 5.11: Konfigurace OSPF pro IPv6 pomocí konfiguračního XML

proces reprezentován instancí třídy `Router`, stačí tedy v nadřažené třídě `AnsaOspfRouting6` místo jediného ukazatele na objekt směrovacího procesu udržovat jejich vektor.

Toto rozšíření ovšem komplikuje proces předávání zpráv, protože je třeba nově řešit adresaci příchozích zpráv odpovídajícímu OSPF procesu. Při řešení byla využita vlastnost protokolu OSPFv3, že každé síťové rozhraní účastnící se směrování je přiřazeno právě jednomu procesu. Při načítání konfiguraci si proto modul vytváří tabulku, která mapuje identifikátory rozhraní na jednotlivá *Router ID*.

Modul pak po přijetí IPv6 datagramu načte strukturu *Control Info*, která obsahuje mj. záznam o síťovém rozhraní, které paket přijalo. Na základě této informace je v tabulce k danému rozhraní dohledán nadřazený OSPF proces a tomu je přijatá zpráva následně předána.

Doplňkové funkce

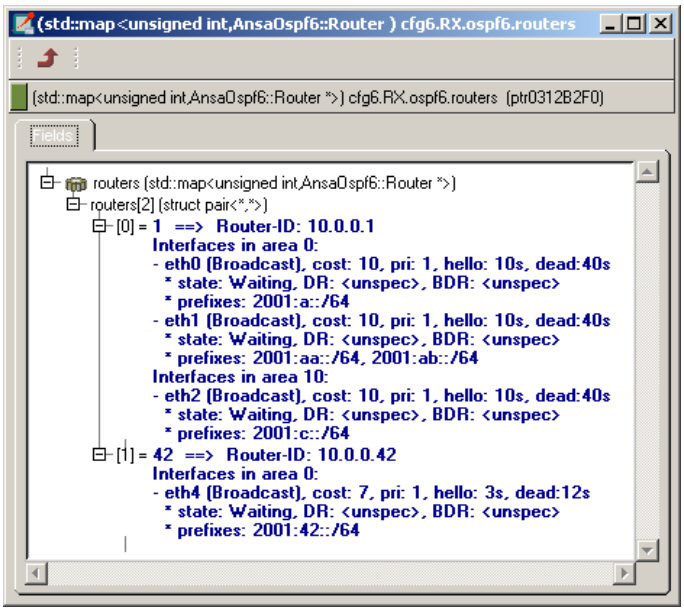
Nová implementace modulu dále umožnila upravit formát výpisu, který nyní obsahuje řadu užitečných informací, mj. o existujících směrovacích procesech a jim přiřazených oblastech, rozhraních a prefixech. Příklad lze vidět na obrázku 5.4.

Modul také implementuje metodu `receiveChangeNotification()` pro příjem hlášení o změně stavu síťových rozhraních. To protokolu OSPF umožňuje reagovat zasláním příslušných LSA a vygenerováním nové sady záznamů pro směrovací tabulku.

5.4.2 Architektura systému

Zdrojové kódy umístěné v `src/ansa/ospfv3/` jsou vzhledem k rozsáhlosti projektu dále členěny do podsložek na základě logického struktury implementace. Jedná se o následující adresáře:

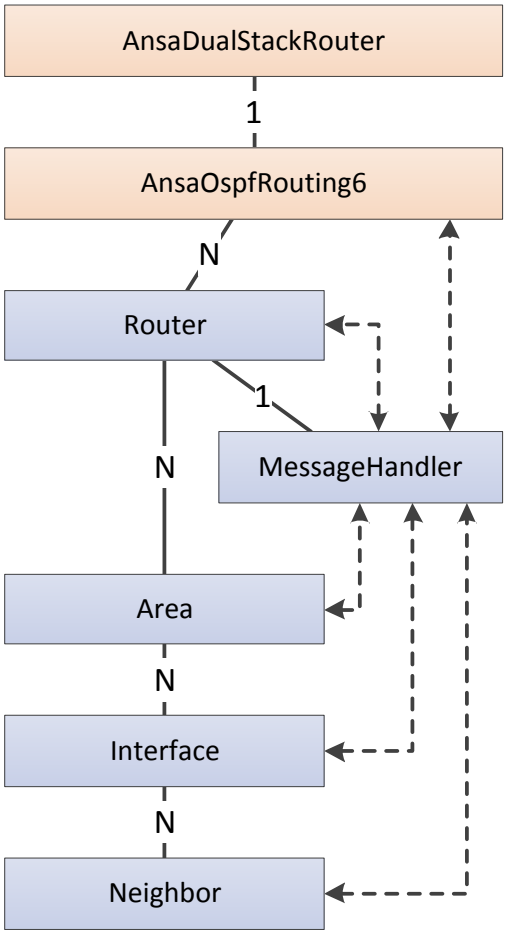
- `interface` obsahuje základní třídu `Interface`, která reprezentuje síťové rozhraní. Zde je uložena většina parametrů z XML souboru jako např. seznam IPv6 prefixů, časovače OSPF paketů, priorita rozhraní, cena linky a další. Jednotlivé soubory ve složce realizují konečný stavový automat, který řídí přechody mezi rolemi rozhraní v souvislosti s volbou DR a BDR směrovače.



Obrázek 5.4: Formát výpisu směrovacích procesů OSPFv3

- `messageHandler` poskytuje především implementaci třídy `MessageHandler` a jejích 5 variant pro jednotlivé typy OSPF paketů. Jedný se o centrální bod celého systému, který obsluhuje veškeré přijaté a odeslané zprávy (včetně interních časovačů) a zajišťuje jejich předávání odpovídajícím modulům.
- `messages` seskupuje řadu pomocných souborů souvisejících se zasíláním zpráv. Předně se zde nachází soubor `ansaOspfPacket6.msg`, ze kterého prostředí OMNeT++ automaticky generuje C++ třídu reprezentující různé typy přenášených zpráv. V souboru je implementováno všech 5 typů OSPF paketů a 7 typů LSA zpráv přesně podle specifikací v dodatku A z RFC 5340 [6]. Složka dále obsahuje třídy pro reprezentaci časovačů a také pomocné metody související s jednotlivými LSA.
- `neighbor` slouží k reprezentaci jednotlivých sousedů pomocí třídy `Neighbor`, která je doplněna konečným stavovým automatem popisujícím přechody mezi stavy sousedství.
- `router` tvoří jádro směrovacího protokolu reprezentované třídami `Area` a `Router`. Zde se nachází metody pro zpracování a generování LSA, tvorbu topologie sítě a výpočet nejkratších cest, aktualizování záznamů ve směrovací tabulce a další.

Vzájemné vztahy jednotlivých tříd a jejich hierarchické uspořádání znázorňuje obrázek 5.5. Pokud postupujeme shora, tak ve směrovači se nachází právě jeden modul směrovacího protokolu OSPFv3. Ten obsahuje obecně libovolný počet směrovacích procesů, kde každý z nich pro účely komunikace zavádí jednu instanci třídy `MessageHandler`. Směrovacímu procesu dále náleží libovolný počet oblastí, do každé oblasti lze přiřadit libovolný počet síťových rozhraní a každé rozhraní ustanovuje vazby s libovolným počtem sousedů.



Obrázek 5.5: Hierarchie tříd v OSPFv3 modulu

5.4.3 Ustanovování sousedství a volba DR/BDR

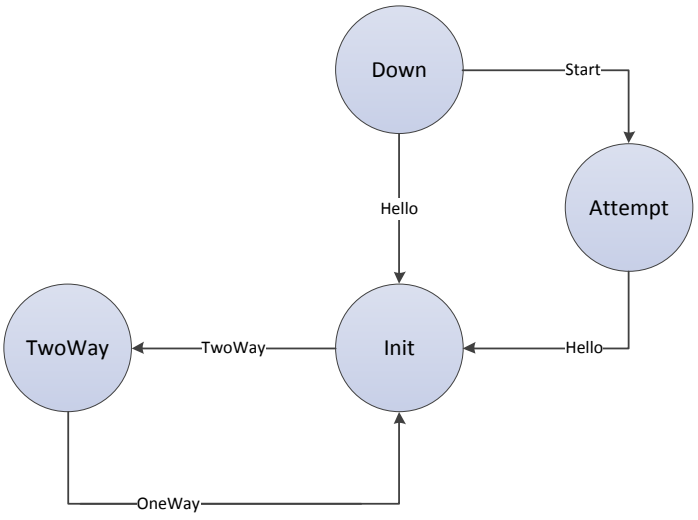
Základním předpokladem pro funkci protokolu OSPF je ustanovení sousedství mezi směrovači, které si poté mohou organizovaně vyměňovat směrovací informace. Toho je dosaženo zasíláním *Hello* zpráv v pravidelných intervalech daných parametrem *HelloInterval* na síťovém rozhraní (konfigurovatelné pomocí XML). Protože jsou zprávy zasílány na multicastovou adresu, technika umožňuje snadno objevit směrovače nově připojené do sítě. Obdobným způsobem lze detekovat i výpadek směrovače – ten nastává, pokud o sobě směrovač nedal vědět *Hello* zprávou během časového limitu daného parametrem *RouterDeadInterval* (také konfigurovatelné pomocí XML).

Nastavení těchto časovačů má tedy významný vliv na rychlost konvergence algoritmu pro ustanovování sousedství. Čím kratší časovače, tím je reakce na události rychlejší, avšak zvyšuje se zátěž sítě periodickým zasílám zpráv. To může být problematické zejména na velkých broadcastových doménách.

Tabulka sousedů

Jak je vidět na obrázku 2.3, *Hello* paket nese mj. *Router ID* odesílatele a také seznam sousedů, kterých si je směrovač vědom. Po přijetí prvního takového paketu se *Router ID* souseda uloží do tabulky sousedů, kterou směrovač vkládá do svých vlastních *Hello* zpráv. Avšak úplně ustanovění sousedství nastává až tehdy, když směrovač nalezne své vlastní *Router ID* v seznamu sousedů odesílatele. Oba směrovače tedy „ví, že o nich jejich soused ví“, tento stav sousedství nazýváme *TwoWay*. V tomto okamžiku je možné přejít do další fáze, kde začíná vlastní výměna LSA pomocí dalších typů OSPF paketů.

V implementaci modulu je soused reprezentován třídou `Neighbor`, jejichž instance jsou ukládány do tabulky uvnitř třídy síťového rozhraní. Pro jednotlivé stavy existují samostatné třídy s metodou `ProcessEvent()`, která řídí přechody dle stavového automatu. Jeho zjednodušenou variantu (bez části zabývající se výměnou LSA) lze vidět na obrázku 5.6.

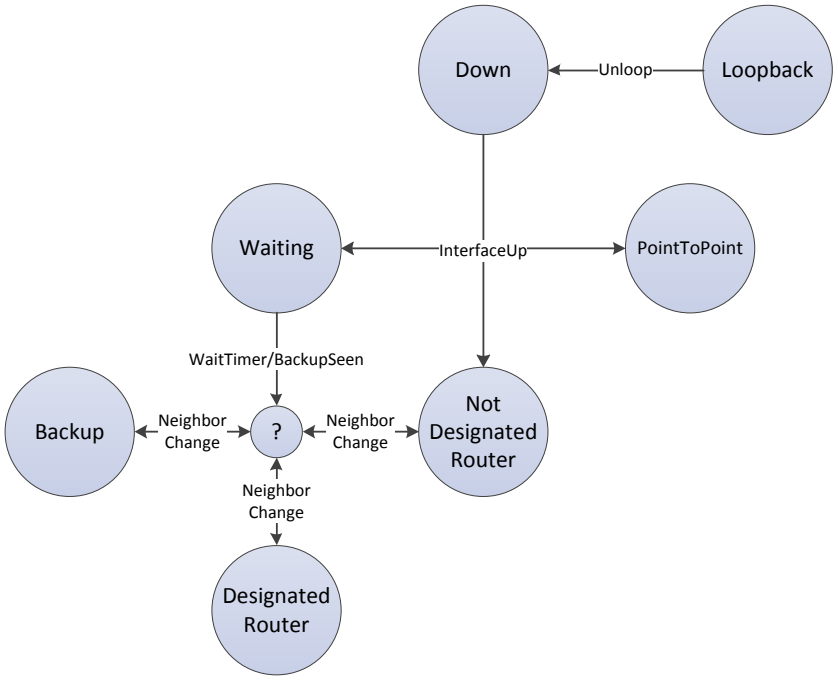


Obrázek 5.6: Diagram stavů objektu Neighbor (převzato z [8])

Síťová rozhraní směrovače

Síťová rozhraní jsou v implementaci reprezentována třídou `Interface`. Pro každé z nich jsou z XML konfigurace načítány parametry určující typ rozhraní (*Point-to-Point*, *Broadcast*, *Non-Broadcast Multi-Access*, *Point-to-Multipoint*), prioritu pro výběr DR, cenu připojené linky a časovače *HelloInterval* a *RouterDeadInterval*. Objekt síťového rozhraní dále obsahuje kontejnery pro uložení seznamu IPv6 prefixů, o nichž bude směrovací protokol informovat, nebo také databázi LSA s *link-local scope*.

Základní funkcí třídy je však zajistit komunikaci se sousedy, můžeme zde proto nalézt definici pěti metod pro vytváření a odesílání jednotlivých typů OSPF paketů. Pomocí zasílání *Hello* paketů probíhá mj. výběr koordinátora zvaného *Designated Router* dle algoritmu představeného v kapitole 2.1.4. Samotné síťové rozhraní během této volby přechází mezi různými stavy, jejichž diagram je znázorněn na obrázku 5.7.



Obrázek 5.7: Diagram stavů objektu Interface (převzato z [8])

Vlastní stavy jsou opět reprezentovány samostatnými třídami ve složce `interface`. Každá z nich obsahuje implementaci metody `ProcessEvent()`, která definuje chování na základě přijetí nejrůznějších událostí, jejichž důsledkem může být např. přechod do jiného stavu. V centrální třídě `InterfaceState` řídící stavový automat se také nachází esenciální metoda `CalculateDesignatedRouter()`, která implementuje algoritmus pro volbu DR.

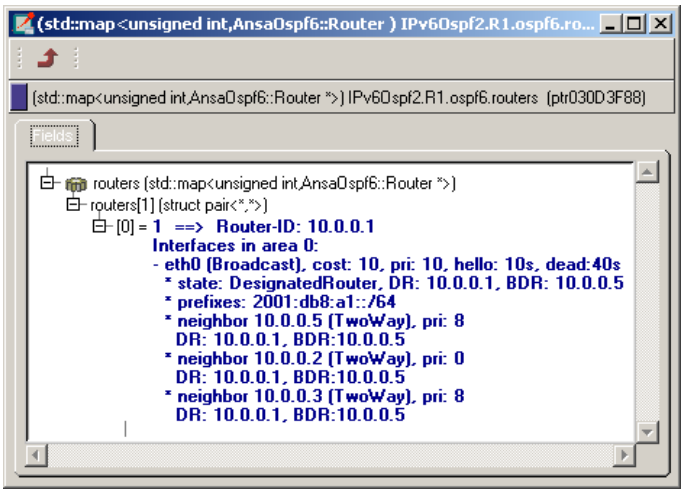
5.4.4 Dosažená funkcionalita a další vývoj

V podstatě veškerá další funkcionalita, která se zabývá správou, zasíláním a přijímáním LSA, generováním síťové topologie, hledáním nejkratších cest, úpravami směrovací tabulky a podobně, spadá pod třídy `Router` a `Area`. Protokol OSPF představuje velmi komplexní systém s rozsáhlou specifikací, jehož úplná implementace však dalece přesahuje rámec této práce. Základní principy směrovacích protokolů typu *link-state* bohužel neumožňují vytvořit zjednodušenou verzi, kterou by bylo možné použít alespoň pro základní směrování.

V aktuální implementaci je zpracováno více než 70% všech zdrojových kódu původního modulu pro OSPFv2. Jedná se v podstatě o všechny podpůrné funkce z nichž některé byly popsané výše, ale také např. definice typů OSPF paketů a LSA zpráv, vytvoření databází pro tyto údaje, implementace interní směrovací tabulky, atd. Kvůli reorganizaci použití LSA došlo k významným změnám v jádře protokolu, jeho implementace proto v první řadě vyžaduje vytvoření nového návrhu, který bude nově zavedené koncepce respektovat. Předpokládáme, že této problematice se bude věnovat některá z budoucích prací projektu ANSA.

Modul `AnsaOspfRouting6` v současnosti umožňuje následující:

1. načíst konfiguraci z XML, na základě které je vytvořena hierarchická struktura procesů, oblasti, rozhraní, a ve vytvořených objektech nastavit odpovídající parametry;
2. vytvořit mapu rozhraní a směrovacích procesů, pomocí které jsou správně předávány OSPF zprávy mezi modulem směrovače a jednotlivými procesy;
3. iniciovat zasílání OSPF paketů pro ustanovení sousedství mezi směrovači a jeho aktivní udržování, včetně objevování nových sousedů a detekce výpadků;
4. zajistit přítomnost DR a BDR pomocí algoritmu pro volbu koordinátora;
5. informovat o aktuálním stavu výše uvedených mechanismů pomocí grafického výstupu, viz demonstrační obrázek 5.8.



Obrázek 5.8: Formát výpisu směrovacího procesu OSPFv3 a ustanovených sousedství

Kapitola 6

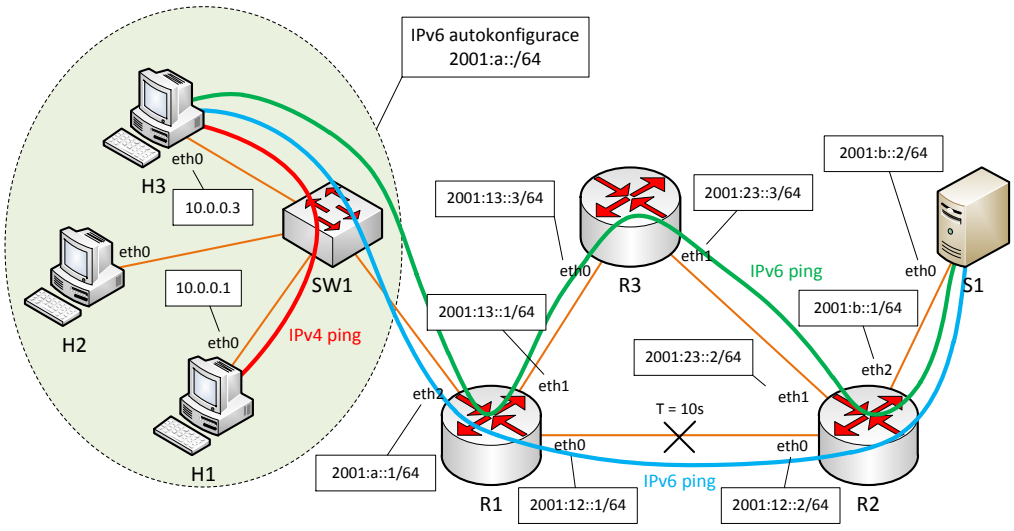
Ukázkové simulace

V průběhu vývoje modulů jsme provedli celou řadu zkušebních simulací pro otestování nejrůznějších mechanismů. Kvůli omezenému rozsahu práce si uvedeme pouze 2 z nich, ty však svým rozsahem pokrývají většinu dostupné funkcionality.

6.1 IPv6 konektivita

V této simulaci jsme se pokusili ověřit, zda správně funguje manuální a automatické přiřazování IPv6 adres, *dual-stack* režim zařízení, rozhraní pro vkládání statických cest, statické směrování a schopnost směrovacích tabulek reagovat na výpadek linky.

Byla vytvořena topologie dle obrázku 6.1, všem zařízením jsme přiřadili znázorněné IPv6 a IPv4 adresy. Klienti H1-H3 nemají IPv6 adresy přiřazeny manuálně, ale využijí techniky autokonfigurace za pomoci směrovače R1, který má na rozhraní `eth0` zapnuté zasílání *Router Advertisement* zpráv. Směrovače R1-R2 mají do směrovací tabulky vloženy cesty do sítí `2001:a::/64` a `2001:b::64`, přičemž metrika je nastavena tak, aby byla preferována linka mezi R1 a R2. Celý konfigurační XML soubor lze nalézt v příloze D.



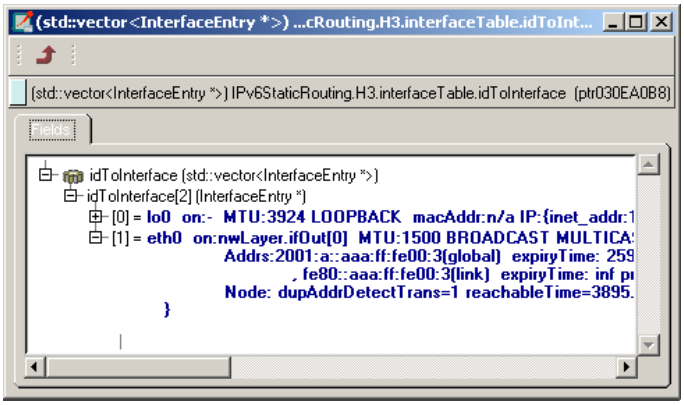
Obrázek 6.1: Ukázková topologie pro otestování IPv6 adresace a statického směrování

V čase t=1s klient H1 vyšle IPv4 ping na adresu klienta H3 a toto opakuje každou sekundu. Ve stejný čas vyšle klient H3 IPv6 ping na adresu serveru S1 a opět opakuje zasílání každou sekundu. V čase t=10s dojde k přerušení linky mezi směrovači R1 a R2.

6.1.1 Dosažené výsledky

Chování systému bylo validováno vůči stejnému zapojení v simulátoru Packet Tracer od firmy Cisco, který pro jednoduché konfigurace poskytuje stejné možnosti jako reálná Cisco zařízení.

Ověřili jsme, že funguje konfigurační rozhraní pro přiřazování IPv6 adres a vytváření statických cest ve směrovacích tabulkách. Po spuštění simulace si všechna zařízení vygenerovala lokální linkové adresy a iniciovala NDP protokol. Všichni klienti začali vysílat *Router Solicitation* pakety, na které rozhraní `eth2` směrovače R1 odpovědělo pomocí *Router Advertisement*. V reakci na tuto zprávu si klienti H1-H3 přiřadili globální IPv6 adresu, jak je vidět na obrázku 6.2.

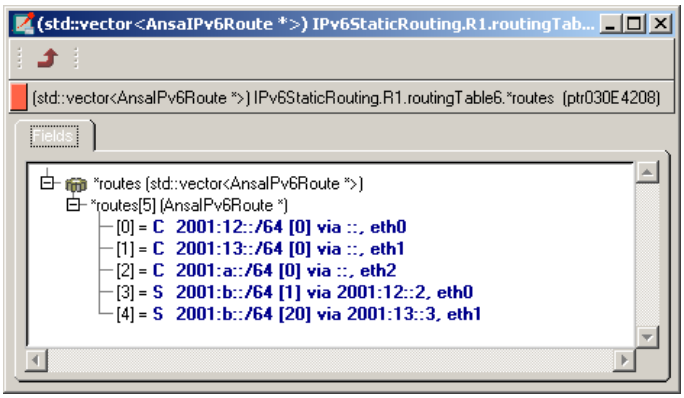


Obrázek 6.2: Informace o síťovém rozhraní včetně automaticky přiřazené IPv6 adresy

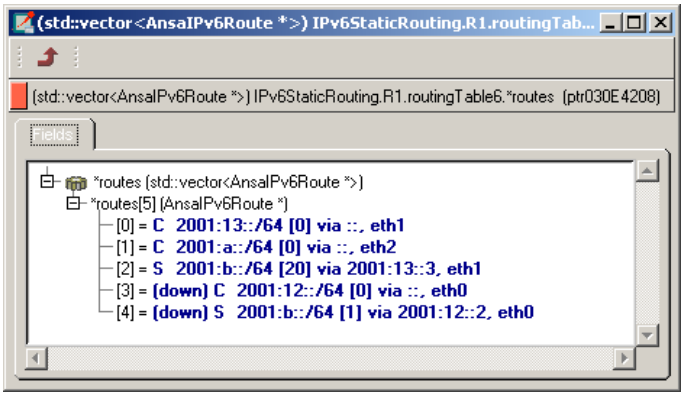
V čase t=1s se klienti H1 a H3 pokusili vyslat ping pakety (IPv4 resp. IPv6), což na síťovém segmentu přepínače SW1 spustilo procedury ARP a *Neighbor Discovery* pro získání fyzických adres nejbližších sousedů. Od tohoto okamžiku proběhl každou sekundu úspěšný IPv4 ping mezi klienty H1 a H3. Klient H3 však zároveň obsluhoval i IPv6 ping adresovaný serveru S1. Tyto datagramy byly úspěšně směrovány k cíly přes směrovače R1 a R2 a odpovědi se vracely stejnou cestou. Výpis směrovací tabulky z této fáze lze vidět na obrázku 6.3.

V čase t=10s došlo přerušení linky mezi směrovači R1 a R2 a jejich směrovací tabulky na toto zareagovaly zařazením této přímé cesty mezi neaktivní (viz obrázek 6.4). Tímto se do popředí dostala alternativní cesta přes směrovač R3, kterou také pakety zasílané mezi sítěmi `2001:a::/64` a `2001:b::/` začaly ihned využívat.

Všechna zařízení tedy v průběhu simulace prokázala očekávané chování a můžeme proto tvrdit, že zkoumané mechanismy jsou funkční.



Obrázek 6.3: Výpis směrovací tabulky směrovače R1 v čase t<10s



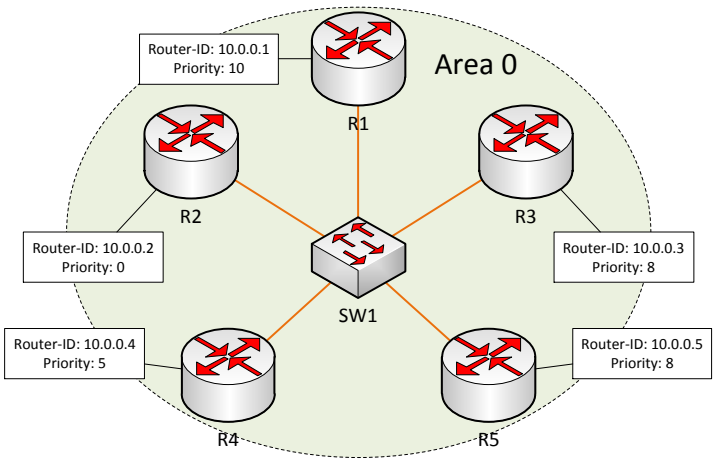
Obrázek 6.4: Výpis směrovací tabulky směrovače R1 v čase t>10s

6.2 Protokol OSPFv3 a ustanovování sousedství

Další simulace je zaměřena na modul reprezentující dynamický směrovací protokol OSPFv3. Je ověřeno správné vytvoření datových struktur na jednotlivých směrovačích a poté sledujeme chování *Hello* paketů sloužících k ustanovení sousedství a volbě směrovače do role *Designated Router*.

Použitá topologie je znázorněna na obrázku 6.5. Každý ze směrovačů je k síti připojen ethernetovým rozhraním, které je zařazeno do páteřní oblasti 0 jediného OSPF procesu. Nastavení *Router Priority* a *Router ID* lze vidět na obrázku, hodnoty ostatních parametrů jsou výchozí. Relevantní jsou v tomto případě časovače *HelloInterval*=10s a *RouterDeadInterval*=40s.

Události v simulaci není třeba nijak explicitně plánovat, OSPF procesy se po startu spustí automaticky a jsou dále řízeny svými interními časovači.



Obrázek 6.5: Ukázková topologie pro otestování OSPFv3 a ustanovování sousedství

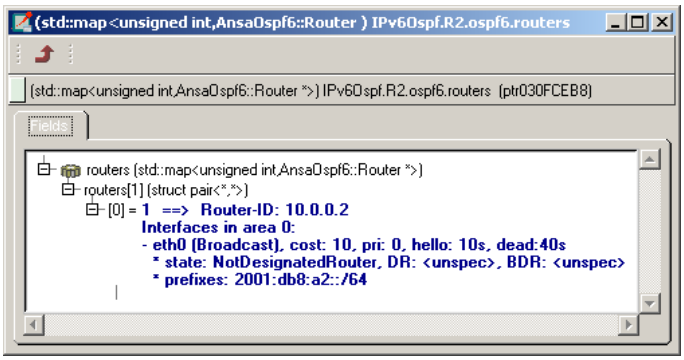
6.2.1 Dosažené výsledky

Výsledky simulace jsme validovali vůči reálnému zapojení v laboratoři za použití směrovačů Cisco 2811 se stejným nastavením všech relevantních parametrů. Pro účely sledování činnosti protokolu a měření času mezi událostmi byly zapnuty kontrolní výpisy OSPF *Hello* zpráv. Při reálném zapojení vznikla otázka, jak zajistit současné spuštění všech směrovačů, aby se proces jejich inicializace co nejvíce blížil chování simulace. Nakonec jsme využili přepínače uprostřed síťové topologie. Během nastavování všech náležitostí na směrovačích byla všechna ethernetová rozhraní na přepínači vypnuta. Pro spuštění celého procesu pak stačilo na přepínači použít příkaz `no shutdown` v kontextu `interface range fa0/1-5`.

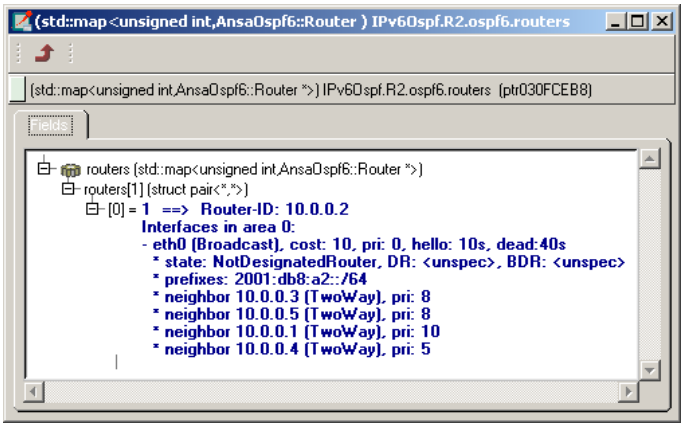
Zasílání zpráv mezi směrovači a tedy i chování směrovacích procesů je v této jednoduché simulaci řízeno výhradně časovači *Hello Timer* a *Wait Timer* (=hodnota *RouterDeadInterval*). Změny proto nastávají deterministicky po 10s intervalech. Na základě toho můžeme průběh celé simulace rozdělit na 4 časové úseky:

- **inicializace** – Směrovací procesy se právě spustily, avšak zatím nepřišel časovač pro zaslání *Hello* paketu (simulace) nebo síťové rozhraní není aktivní (reálný systém). Výpis aktuálního stavu směrovacího procesu v této fázi můžeme vidět na obrázku 6.6. Jedná se o směrovač R2 – protože konfigurace ostatních směrovačů je obdobná a uvádět výpisy všech by neúměrně navýšilo rozsah práce, budeme i nadále sledovat právě R2.

- **čas t=0-40s** – V této fázi každý směrovač objeví všechny své sousedy a ustanoví s nimi *TwoWay* relaci. Zatím však není znám *Designated Router*. Obrázek 6.7 zobrazuje výpis modelu směrovače, ukázka 6.1 pak tabulku sousedů jeho reálného vzoru.



Obrázek 6.6: Výpis směrovacího procesu na R2 v době inicializace



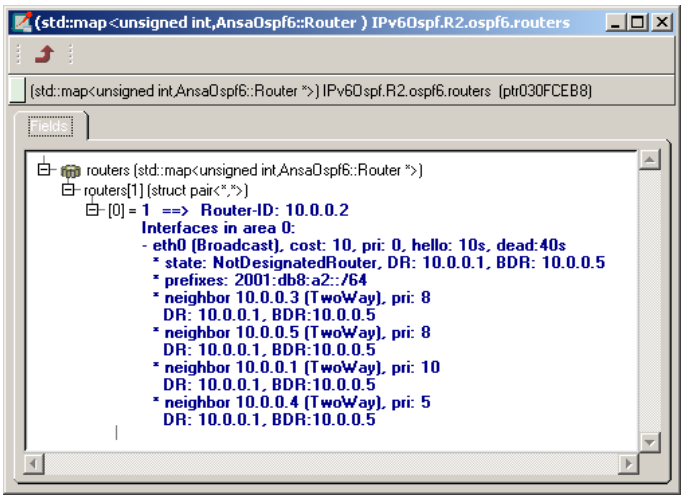
Obrázek 6.7: Výpis směrovacího procesu na R2 v čase t=0-40s

```
10.0.0.1 10 2WAY/DROTHER 00:00:36 4 FastEthernet0/0
10.0.0.5 8 2WAY/DROTHER 00:00:36 4 FastEthernet0/0
10.0.0.4 5 2WAY/DROTHER 00:00:35 4 FastEthernet0/0
10.0.0.3 8 2WAY/DROTHER 00:00:35 4 FastEthernet0/0
```

Ukázka 6.1: Výpis tabulky sousedů na R2 v čase t=0-40s

- **čas t=40-50s** – Po 40s od inicializace směrovače vyprší časovač *Wait Timer* a spustí se algoritmus pro volbu DR. Tím je jednoznačně zvolen směrovač R1, protože má nejvyšší prioritu. Reálné zařízení v tomto okamžiku začne vykazovat odchylky v chování, protože iniciuje výměnu LSA s právě ustanoveným DR. Z toho důvodu se stav rozhraní změnil na `EXSTART`, zatímco náš model, který tuto funkcionalitu nepodporuje, zůstává v `TwoWay`.

- **čas t>50s** – Po přenesení další sady *Hello* paketů v čase 50s od startu simulace je kromě DR zvolen i BDR a všechny směrovače mají kompletní informaci o svých sousedech. Volbu BDR vyhrává R5, protože ačkoliv má stejnou prioritu jako R3, jeho *Rouer ID* je vyšší. Opět platí, že v případě reálných zařízení dojde i k synchronizaci *Link State* databází, čemuž odpovídají názvy stavů síťových rozhraní. Konfiguraci R2 v simulaci zobrazuje obrázek 6.8, tabulka sousedů reálného zařízení je uvedena v ukázce 6.2. Detailní výpis o sousedech z prostředí IOS lze pak nalézt v příloze F.



Obrázek 6.8: Výpis směrovacího procesu na R2 v čase t>50s

```
10.0.0.1 10 FULL/DR 00:00:35 4 FastEthernet0/0
10.0.0.5 8 FULL/BDR 00:00:37 4 FastEthernet0/0
10.0.0.4 5 2WAY/DROTHER 00:00:36 4 FastEthernet0/0
10.0.0.3 8 2WAY/DROTHER 00:00:36 4 FastEthernet0/0
```

Ukázka 6.2: Výpis tabulky sousedů na R2 v čase t>50s

V této simulaci jsme se přesvědčili, že ustanovování sousedství mezi směrovači s běžícím OSPFv3 procesem probíhá ve stejných krocích jak v rámci simulace, tak i na skutečných zařízeních. Síť v obou případech zkonvergovala po 6 výměnách *Hello* paketů v čase 50s a všechny modely zařízení v tomto okamžiku vykazovaly stejnou konfiguraci, jako jejich reálné protějšky.

Závěr

Jedním z cílů projektu ANSA je poskytnout komplexní modely síťových zařízení, která nám v prostředí simulátoru OMNeT++ umožní vytvářet realistické simulace chování počítačových sítí. V rámci této práce jsem se zabýval současnými možnostmi frameworku INET modelovat protokol IPv6 a pokusil jsem se navrhnout a implementovat několik rozšíření pro nasazení tohoto protokolu v simulacích.

Před vlastním návrhem bylo třeba se nejprve seznámit s aktuálním stavem implementace podpory IPv6 ze strany projektu INET. Protože oficiální dokumentace je v této oblasti nedostačující, provedl jsem zevrubnou analýzu zdrojových kódů a její výsledky jsem prezentoval v první části práce. Na základě zjištěných informací byly navrženy a sestaveny modely *dual-stack* směrovače a *dual-stack* klienta, které staví na elementárních IPv6 modulech frameworku INET a integrují v sobě všechna dosavadní rozšíření projektu ANSA. Zařízení umožňují manuální i automatickou konfiguraci IPv6 adres a nastavení statického směrování. Dále byla započata implementace komplexního modulu dynamického směrovacího protokolu OSPFv3, která v současnosti zajišťuje všechny podpůrné funkce včetně ustanovení sousedství mezi směrovači a poskytuje tak dobrý základ pro budoucí vývoj jádra protokolu.

Informace o správném fungování nejrůznějších mechanismů jsem čerpal jednak z RFC specifikací, ale také praktickým ověřováním chování zařízení firmy Cisco ve školní laboratoři a simulátoru Packet Tracer. Prostředí Cisco IOS také posloužilo jako referenční soubor příkazů pro nastavení IPv6, na základě kterých byla navržena odpovídající XML struktura použitelná ke konfiguraci.

Výsledné modely lze použít k vytváření simulací počítačových sítí nad protokolem IPv6 se zachováním plné zpětné kompatibility s IPv4. Díky podpoře scénářů lze zkoumat chování sítě v čase a např. sledovat směrování přes záložní linky po výpadku některého zařízení. Protože konfigurační XML struktura úzce vychází z příkazů reálných zařízení, lze ji generovat i automaticky – tvorba odpovídajícího překladače je předmětem související diplomové práce. Byly také zavedeny některé konvence, které umožní snadnější nasazení implementovaných rozšíření do dalších zařízení, např. modelu přepínače, jehož vývojem se zabývá jiná diplomová práce.

Pro další vývoj se počítá s doplněním jádra směrovacího protokolu OSPFv3 do připraveného modulu. Bylo by také vhodné aktualizovat způsob konfigurace stávajících IPv4 rozšíření, aby byl konzistentní s nově zavedenými zvyklostmi pro IPv6.

Tato práce mi umožnila prohloubit si znalosti o fungovaní protokolů IPv6, NDP a OSPF, a to včetně technik použitých při jejich implementaci. Také jsem se seznámil s prostředím OMNeT++, což mi umožnilo rozšířit své dovednosti v oblasti modelování a simulace. V neposlední řadě jsem získal cennou zkušenost s kolektivní prací na velkém projektu, kde je třeba navazovat na předešlou tvorbu a zároveň dobře připravit podklady pro kolegy, kteří se budou zabývat dalšími rozšířeními.

Literatura

[1] ANSAWiki.
URL https://nes.fit.vutbr.cz/ansa/pmwiki.php?n=Main.HomePage

[2] Implementing OSPF for IPv6 . In *Cisco IOS IPv6 Configuration Guide, Release 12.4*, Cisco Systems, Inc., release 12.4.
URL http://www.cisco.com/en/US/docs/ios/ipv6/configuration/guide/ip6-ospf.html

[3] INET Framework.
URL http://inet.omnetpp.org/index.php?n=Main.HomePage

[4] Carpenter, B.; Moore, K.: Connection of IPv6 Domains via IPv4 Clouds. RFC 3056 (Proposed Standard), Únor 2001.
URL http://tools.ietf.org/html/rfc3056

[5] Cisco Systems, Inc.: *Cisco IOS IPv6 Command Reference* . Červenec 2010.
URL http://www.cisco.com/en/US/docs/ios/ipv6/command/reference/ipv6_book.html

[6] Coltun, R.; Ferguson, D.; Moy, J.; aj.: OSPF for IPv6. RFC 5340 (Proposed Standard), Červenec 2008.
URL http://tools.ietf.org/html/rfc5340

[7] Coltun, R.; Fuller, V.: The OSPF NSSA Option. RFC 1587 (Standard), Březen 1994.
URL http://tools.ietf.org/html/rfc1587

[8] Danko, M.: Modelování směrovacích protokolů OSPF v simulátoru OMNeT++. Bakalářská práce, Brno, FIT VUT v Brně, 2009.
URL http://www.fit.vutbr.cz/study/DP/rpfile.php?id=7989

[9] Deering, S.; Hinden, R.: Internet Protocol, Version 6 (IPv6) Specification. RFC 2460 (Draft Standard), Prosinec 1998.
URL http://tools.ietf.org/html/rfc2460

[10] Hinden, R.; Deering, S.: IP Version 6 Addressing Architecture. RFC 4291 (Draft Standard), Únor 2006.
URL http://tools.ietf.org/html/rfc4291

[11] Hinden, R.; Deering, S.: IP Version 6 Addressing Architecture. RFC 4291 (Draft Standard), Únor 2006.
URL http://tools.ietf.org/html/rfc4291

[12] Hinden, R.; Haberman, B.: Unique Local IPv6 Unicast Addresses. RFC 4193 (Proposed Standard), Říjen 2005.
URL http://tools.ietf.org/html/rfc4193.txt

[13] Huitema, C.: Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs). RFC 4380 (Proposed Standard), Únor 2006.
URL http://tools.ietf.org/html/rfc4380

[14] Moy, J.: OSPF Version 2. RFC 2328 (Standard), Duben 1998.
URL http://tools.ietf.org/html/rfc2328

[15] Narten, T.; Nordmark, E.; Simpson, W.; aj.: Neighbor Discovery for IP version 6 (IPv6). RFC 4861 (Draft Standard), Září 2007.
URL http://tools.ietf.org/html/rfc4861

[16] Postel, J.: Internet Protocol. RFC 791 (Standard), Září 1981.
URL http://tools.ietf.org/html/rfc791

[17] Rybová, V.: Modelování a simulace návrhových vzorů směrování v počítačových sítích. Bakalářská práce, Brno, FIT VUT v Brně, 2009.
URL http://www.fit.vutbr.cz/study/DP/rpfile.php?id=8351

[18] Scherfel, P.: Simulace chování sítě na základě analýzy konfiguračních souborů aktivních síťových zařízení. Bakalářská práce, Brno, FIT VUT v Brně, 2009.
URL http://www.fit.vutbr.cz/study/DP/rpfile.php?id=7852

[19] Sivák, V.: Model Cisco směrovače v simulačním nástroji OMNeT++. Bakalářská práce, Brno, FIT VUT v Brně, 2009.
URL http://www.fit.vutbr.cz/study/DP/rpfile.php?id=7851

[20] Smith, L.; Lipner, I.: Free Pool of IPv4 Address Space Depleted. Leden 2011.
URL http://www.nro.net/news/ipv4-free-pool-depleted

[21] Suchomel, T.: Rozšíření simulátoru OMNeT++ o filtrovací pravidla ACL. Bakalářská práce, Brno, FIT VUT v Brně, 2009.
URL http://www.fit.vutbr.cz/study/DP/rpfile.php?id=7854

[22] Thomson, S.; Narten, T.; Jinmei, T.: IPv6 Stateless Address Autoconfiguration. RFC 4862 (Draft Standard), Září 2007.
URL http://tools.ietf.org/html/rfc4862

[23] Tlolka, M.: Simulace EIGRP protokolu v prostředí OMNeT++. Bakalářská práce, Brno, FIT VUT v Brně, 2009.
URL http://www.fit.vutbr.cz/study/DP/rpfile.php?id=7853

[24] Varga, A.; aj.: *OMNeT++ User Manual*. OpenSim Ltd., 2010, ver. 4.1.
URL http://www.omnetpp.org/doc/omnetpp41/Manual.pdf

[25] Varga, A.; aj.: *INET framework for OMNeT++*. Únor 2011, ver. INET-20110225.

[26] Weissmann, P.: IPv6 Operating Systems.
URL http://ipv6int.net/systems/index.html

Příloha A

Obsah CD

- `ipv6.pdf` – PDF verze technické zprávy
- `report/` – zdrojové soubory technické zprávy
- `ANSAINET/` – zdrojové soubory frameworku INET s ANSA rozšířeními (z 24. 5. 2011)
- `ANSAINET/src/ansa/ansaDualStackRouter` – model *dual-stack* směrovače
- `ANSAINET/src/ansa/ansaDualStackHost` – model *dual-stack* směrovače
- `ANSAINET/src/ansa/deviceConfigurator/` – zdrojové souboy XML konfigurátoru a XML knihovny
- `ANSAINET/src/ansa/ipSplitter/` – zdrojové soubory modulu `IpSplitter`
- `ANSAINET/src/ansa/ipv6/` – zdrojové soubory směrovací tabulky pro IPv6
- `ANSAINET/src/ansa/ospfv3/` – zdrojové soubory modulu pro směrovací protokol OSPFv3
- `ANSAINET/examples/ansaExamples/IPv6StaticRouting/` – ukázková simulace IPv6 adresace a statického směrování
- `ANSAINET/examples/ansaExamples/IPv6Ospf/` – ukázková simulace OSPFv3 protokolu s objevováním sousedů

Příloha B

Seznam konfigurace pro IPv6

```
<(Device Type) id="(device ID)">
   <Interfaces>
      <Interface name="(interface name)">
         <IPv6Address>(IPv6 address)/(Prefix Length)</IPv6Address>
         <NdpAdvSendAdvertisements>(yes/no)</NdpAdvSendAdvertisements>
         <NdpAdvPrefix>(IPv6 prefix)/(Prefix Length)</NdpAdvPrefix>
         <NdpMaxRtrAdvInterval>(4-1800)</NdpMaxRtrAdvInterval>
         <NdpMinRtrAdvInterval>(3-1350)</NdpMinRtrAdvInterval>
      </Interface>
   </Interfaces>

   <Routing6>
      <Static>
         <Route>
            <NetworkAddress>(IPv6 prefix)/(Prefix Length)</NetworkAddress>
            <NextHopAddress>(IPv6 address)</NextHopAddress>
            <AdministrativeDistance>(1-254)</AdministrativeDistance>
         </Route>
      </Static>
   </Routing6>

   <DefaultRouter>(IPv6 Address)</DefaultRouter>
</(Device Type)>
```

Příloha C

Seznam konfigurace pro OSPFv3

```
<Router id="(device ID)">
   <Interfaces>
      <Interface name="(interface name)">
         <OspfProcess6>(Process ID)</OspfProcess6>
         <OspfArea6>(Area ID)</OspfArea6>
         <OspfInstance6>(n)</OspfInstance6>
         <OspfNetworkType6>
            (point-to-point/broadcast/non-broadcast/point-to-multipoint)
         </OspfNetworkType6>
         <OspfCost6>(n)</OspfCost6>
         <OspfPriority6>(n)</OspfPriority6>
         <OspfHelloInterval6>(n)</OspfHelloInterval6>
         <OspfRouterDeadInterval6>(n)</OspfRouterDeadInterval6>
         <OspfRetransmitInterval6>(n)</OspfRetransmitInterval6>
         <OspfTransmitDelay6>(n)</OspfTransmitDelay6>
      </Interface>
   </Interfaces>

   <Routing6>
      <Ospf>
         <Process id="(Process ID)">
            <RouterId>(Router ID)</RouterId>
         </Process>
      </Ospf>
   </Routing6>
</Router>
```

Příloha D

Konfigurační soubor IPv6 simulace

```
<Devices>
   <Host id="10.0.0.1">
      <Interfaces>
         <Interface name="eth0">
            <IPAddress>10.0.0.1</IPAddress>
            <Mask>255.255.255.0</Mask>
         </Interface>
      </Interfaces>
      <DefaultRouter>2001:a::1</DefaultRouter>
   </Host>

   <Host id="10.0.0.2">
      <Interfaces>
         <Interface name="eth0">
            <IPAddress>10.0.0.2</IPAddress>
            <Mask>255.255.255.0</Mask>
         </Interface>
      </Interfaces>
      <DefaultRouter>2001:a::1</DefaultRouter>
   </Host>

   <Host id="10.0.0.3">
      <Interfaces>
         <Interface name="eth0">
            <IPAddress>10.0.0.3</IPAddress>
            <Mask>255.255.255.0</Mask>
         </Interface>
      </Interfaces>
      <DefaultRouter>2001:a::1</DefaultRouter>
   </Host>

   <Host id="10.0.1.1">
      <Interfaces>
         <Interface name="eth0">
            <IPAddress>10.0.1.1</IPAddress>
```

```
         <Mask>255.255.255.0</Mask>
         <IPv6Address>2001:b::2/64</IPv6Address>
      </Interface>
   </Interfaces>
   <DefaultRouter>2001:b::1</DefaultRouter>
</Host>

<Router id="192.0.0.1">
   <Interfaces>
      <Interface name="eth0">
         <IPAddress>192.0.0.1</IPAddress>
         <Mask>255.255.255.0</Mask>
         <IPv6Address>2001:12::1/64</IPv6Address>
      </Interface>
      <Interface name="eth1">
         <IPv6Address>2001:13::1/64</IPv6Address>
      </Interface>
      <Interface name="eth2">
         <IPv6Address>2001:a::1/64</IPv6Address>
         <NdpAdvSendAdvertisements>yes</NdpAdvSendAdvertisements>
         <NdpAdvPrefix>2001:a::/64</NdpAdvPrefix>
      </Interface>
   </Interfaces>

   <Routing6>
      <Static>
         <Route>
            <NetworkAddress>2001:b::/64</NetworkAddress>
            <NextHopAddress>2001:12::2</NextHopAddress>
         </Route>
         <Route>
            <NetworkAddress>2001:b::/64</NetworkAddress>
            <NextHopAddress>2001:13::3</NextHopAddress>
            <AdministrativeDistance>20</AdministrativeDistance>
         </Route>
      </Static>
   </Routing6>
</Router>

<Router id="192.0.0.2">
   <Interfaces>
      <Interface name="eth0">
         <IPAddress>192.0.0.2</IPAddress>
         <Mask>255.255.255.0</Mask>
         <IPv6Address>2001:12::2/64</IPv6Address>
      </Interface>
      <Interface name="eth1">
         <IPv6Address>2001:23::2/64</IPv6Address>
```

```
         </Interface>
         <Interface name="eth2">
            <IPv6Address>2001:b::1/64</IPv6Address>
         </Interface>
      </Interfaces>

      <Routing6>
         <Static>
            <Route>
               <NetworkAddress>2001:a::/64</NetworkAddress>
               <NextHopAddress>2001:12::1</NextHopAddress>
            </Route>
            <Route>
               <NetworkAddress>2001:a::/64</NetworkAddress>
               <NextHopAddress>2001:23::3</NextHopAddress>
               <AdministrativeDistance>20</AdministrativeDistance>
            </Route>
         </Static>
      </Routing6>
   </Router>

   <Router id="192.0.0.3">
      <Interfaces>
         <Interface name="eth0">
            <IPAddress>192.0.0.3</IPAddress>
            <Mask>255.255.255.0</Mask>
            <IPv6Address>2001:13::3/64</IPv6Address>
         </Interface>
         <Interface name="eth1">
            <IPv6Address>2001:23::3/64</IPv6Address>
         </Interface>
      </Interfaces>

      <Routing6>
         <Static>
            <Route>
               <NetworkAddress>2001:a::/64</NetworkAddress>
               <NextHopAddress>2001:13::1</NextHopAddress>
            </Route>
            <Route>
               <NetworkAddress>2001:b::/64</NetworkAddress>
               <NextHopAddress>2001:23::2</NextHopAddress>
            </Route>
         </Static>
      </Routing6>
   </Router>
</Devices>
```

Příloha E

Konfigurační soubor OSPF simulace

```
<Devices>
   <Router id="10.0.0.1">
      <Interfaces>
         <Interface name="eth0">
            <IPAddress>10.0.0.1</IPAddress>
            <Mask>255.255.255.0</Mask>
            <IPv6Address>2001:db8:a1::1/64</IPv6Address>
            <OspfProcess6>1</OspfProcess6>
            <OspfArea6>0</OspfArea6>
            <OspfPriority6>10</OspfPriority6>
         </Interface>
      </Interfaces>
      <Routing6>
         <Ospf>
            <Process id="1">
               <RouterId>10.0.0.1</RouterId>
            </Process>
         </Ospf>
      </Routing6>
   </Router>

   <Router id="10.0.0.2">
      <Interfaces>
         <Interface name="eth0">
            <IPAddress>10.0.0.2</IPAddress>
            <Mask>255.255.255.0</Mask>
            <IPv6Address>2001:db8:a2::1/64</IPv6Address>
            <OspfProcess6>1</OspfProcess6>
            <OspfArea6>0</OspfArea6>
            <OspfPriority6>0</OspfPriority6>
         </Interface>
      </Interfaces>
      <Routing6>
         <Ospf>
            <Process id="1">
```

```
            <RouterId>10.0.0.2</RouterId>
         </Process>
      </Ospf>
   </Routing6>
</Router>

<Router id="10.0.0.3">
   <Interfaces>
      <Interface name="eth0">
         <IPAddress>10.0.0.3</IPAddress>
         <Mask>255.255.255.0</Mask>
         <IPv6Address>2001:db8:a3::1/64</IPv6Address>
         <OspfProcess6>1</OspfProcess6>
         <OspfArea6>0</OspfArea6>
         <OspfPriority6>8</OspfPriority6>
      </Interface>
   </Interfaces>
   <Routing6>
      <Ospf>
         <Process id="1">
            <RouterId>10.0.0.3</RouterId>
         </Process>
      </Ospf>
   </Routing6>
</Router>

<Router id="10.0.0.4">
   <Interfaces>
      <Interface name="eth0">
         <IPAddress>10.0.0.4</IPAddress>
         <Mask>255.255.255.0</Mask>
         <IPv6Address>2001:db8:a4::1/64</IPv6Address>
         <OspfProcess6>1</OspfProcess6>
         <OspfArea6>0</OspfArea6>
         <OspfPriority6>5</OspfPriority6>
      </Interface>
   </Interfaces>
   <Routing6>
      <Ospf>
         <Process id="1">
            <RouterId>10.0.0.4</RouterId>
         </Process>
      </Ospf>
   </Routing6>
</Router>

<Router id="10.0.0.5">
   <Interfaces>
```

```
         <Interface name="eth0">
            <IPAddress>10.0.0.5</IPAddress>
            <Mask>255.255.255.0</Mask>
            <IPv6Address>2001:db8:a5::1/64</IPv6Address>
            <OspfProcess6>1</OspfProcess6>
            <OspfArea6>0</OspfArea6>
            <OspfPriority6>8</OspfPriority6>
         </Interface>
      </Interfaces>
      <Routing6>
         <Ospf>
            <Process id="1">
               <RouterId>10.0.0.5</RouterId>
            </Process>
         </Ospf>
      </Routing6>
   </Router>
</Devices>
```

Příloha F

Výpis sousedů R2 v OSPF simulaci

```
Neighbor 10.0.0.3
   In the area 0 via interface FastEthernet0/0
   Neighbor: interface-id 4, link-local address FE80::21F:CAFF:FE05:27B0
   Neighbor priority is 8, State is 2WAY, 2 state changes
   DR is 10.0.0.1 BDR is 10.0.0.5
   Options is 0x000013 in Hello (V6-Bit E-Bit R-bit )
   Dead timer due in 00:00:33
   Neighbor is up for 00:00:44
   Index 0/0/0, retransmission queue length 0, number of retransmission 0
   First 0x0(0)/0x0(0)/0x0(0) Next 0x0(0)/0x0(0)/0x0(0)
   Last retransmission scan length is 0, maximum is 0
   Last retransmission scan time is 0 msec, maximum is 0 msec
Neighbor 10.0.0.4
   In the area 0 via interface FastEthernet0/0
   Neighbor: interface-id 4, link-local address FE80::21F:CAFF:FE05:C3E0
   Neighbor priority is 5, State is 2WAY, 2 state changes
   DR is 10.0.0.1 BDR is 10.0.0.5
   Options is 0x000013 in Hello (V6-Bit E-Bit R-bit )
   Dead timer due in 00:00:32
   Neighbor is up for 00:00:45
   Index 0/0/0, retransmission queue length 0, number of retransmission 0
   First 0x0(0)/0x0(0)/0x0(0) Next 0x0(0)/0x0(0)/0x0(0)
   Last retransmission scan length is 0, maximum is 0
   Last retransmission scan time is 0 msec, maximum is 0 msec
Neighbor 10.0.0.1
   In the area 0 via interface FastEthernet0/0
   Neighbor: interface-id 4, link-local address FE80::21F:CAFF:FE05:28D0
   Neighbor priority is 10, State is FULL, 6 state changes
   DR is 10.0.0.1 BDR is 10.0.0.5
   Options is 0x000013 in Hello (V6-Bit E-Bit R-bit )
   Options is 0x000013 in DBD (V6-Bit E-Bit R-bit )
   Dead timer due in 00:00:39
   Neighbor is up for 00:00:47
   Index 1/1/1, retransmission queue length 0, number of retransmission 0
   First 0x0(0)/0x0(0)/0x0(0) Next 0x0(0)/0x0(0)/0x0(0)
```

```
   Last retransmission scan length is 0, maximum is 0
   Last retransmission scan time is 0 msec, maximum is 0 msec
Neighbor 10.0.0.5
   In the area 0 via interface FastEthernet0/0
   Neighbor: interface-id 4, link-local address FE80::21F:CAFF:FE05:6A10
   Neighbor priority is 8, State is FULL, 6 state changes
   DR is 10.0.0.1 BDR is 10.0.0.5
   Options is 0x000013 in Hello (V6-Bit E-Bit R-bit )
   Options is 0x000013 in DBD (V6-Bit E-Bit R-bit )
   Dead timer due in 00:00:38
   Neighbor is up for 00:00:49
   Index 2/2/2, retransmission queue length 0, number of retransmission 0
   First 0x0(0)/0x0(0)/0x0(0) Next 0x0(0)/0x0(0)/0x0(0)
   Last retransmission scan length is 0, maximum is 0
   Last retransmission scan time is 0 msec, maximum is 0 msec
```