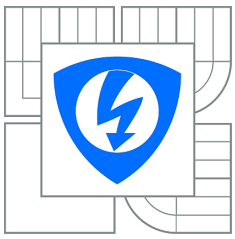
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF CONTROL AND INSTRUMENTATION

NÁVRH TESTPLÁNU PRO IPV6 STACK

IPV6 STACK TEST PLAN DESIGN

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR
JAN MALANÍK

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. PETR PETYOVSKÝ

BRNO 2011

Licenční smlouva
poskytovaná k výkonu práva užít školní dílo

uzavřená mezi smluvními stranami:

1. Pan

Jméno a příjmení:	Jan Malaník
Bytem:	Březnice 530, 76001, Zlín
Narozen (datum a místo):	14.5.1988, Zlín

(dále jen autor)

a

2. Vysoké učení technické v Brně

Fakulta elektrotechniky a komunikačních technologií
se sídlem Technická 3058/10, Královo Pole, 61600, Brno
jejímž jménem jedná na základě písemného pověření děkanem fakulty:
Prof. Ing. Pavel Jura, CSc.

(dále jen nabyvatel)

Čl. 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):

   ☐ disertační práce

   ☐ diplomová práce

   ☒ bakalářská práce

   ☐ jiná práce, jejíž druh je specifikován jako .....

   (dále jen VŠKP nebo dílo)

Název VŠKP:	Návrh testplánu pro IPv6 stack
Vedoucí/ školitel VŠKP:	Ing. Petr Petyovský
Ústav:	Ústav automatizace a měřicÍ techniky
Datum obhajoby VŠKP:	13.6.2011

VŠKP odevzdal autor nabyvateli v[1]:

☐ tištěné formě — počet exemplářů 1

☐ elektronické formě — počet exemplářů 1

[1] hodící se zaškrtněte

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.

3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.

4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Čl. 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.

2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.

3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti

   □ ihned po uzavření této smlouvy

   □ 1 rok po uzavření této smlouvy

   □ 3 roky po uzavření této smlouvy

   □ 5 let po uzavření této smlouvy

   □ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)

4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením §47b zákona č. 111/1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Čl. 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.

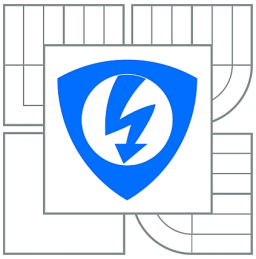
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.

3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.

4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne:

Nabyvatel Autor



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav automatizace a měřicí techniky

Bakalářská práce

bakalářský studijní obor
Automatizační a měřicí technika

Student:	Jan Malaník	***ID:***	106610
Ročník:	3	***Akademický rok:***	2010/2011

NÁZEV TÉMATU:

Návrh testplánu pro IPv6 stack

POKYNY PRO VYPRACOVÁNÍ:

Práce se bude zabývat automatickým testováním IPv6 stacku, který je implementován v jádře OS Linux distribuce Fedora 14. Cílem je navrhnout automatizovaný systém testů založený na rodině testů IPv6 Ready a pomocí něj provést testování systému Fedora 14. Výstupem testů bude seznam chyb, které je nutné opravit.

1. Nastudujte si problematiku testů IPv6 Ready a standardů pro testování softwaru IEEE829.
2. Seznamte se s vybavením na pracovištích BrnoLab a CertLab.
3. Vyberte vhodné testy z rodiny testů IPv6 Ready.
4. Implementujte zvolené automatické testy.
5. Realizujte testování na systému Fedora 14.
6. Navrhněte a vytvořte kompletní testplán na základě IEEE 829.
7. Vyhodnoťte získané výsledky a navrhněte další možná zlepšení.

DOPORUČENÁ LITERATURA:

[1] SATRAPA, Pavel. IPV6 : Internetový protokol IPV6. Vyd. 2. Praha : CZ.NIC, z. s. p. o., 2008. 351 s. Dostupné z WWW: <http://knihy.nic.cz/>. ISBN 978-80-904248-0-7.
[2] JELÍNEK, Lukáš. Jádro systému linux : Kompletní průvodce programátora. Vyd. 1. Brno : Computer Press, a.s., 2008. 686 s. ISBN 978-80-251-2084-2.

Termín zadání:	7.2.2011	***Termín odevzdání:***	30.5.2011

Vedoucí práce: Ing. Petr Petyovský

prof. Ing. Pavel Jura, CSc.
Předseda oborové rady

Abstrakt

V teoretické části mé práce je popsán protokol IPV6, jeho historie, výhody a struktura datagramu. Také je srovnán se starší verzí IPV4. Dále jsou popsány možnosti testování operačního systému Fedora 14 z pohledu síťové komunikace. Pro srovnání různých variant jsou použita hlediska, jako cena, výkon a pohodlí. Jedna kapitola je věnována historickému vývoji a různým používaným rozhraním pro síťovou komunikaci. V závěru teoretické popis normy IEEE 829, která říká, jakým způsobem sestavit testplán.

Praktická část se zabývá vývojem testplánu dle normy IEEE 829. Testplán je vytvořen se zaměřením na stack pro protokol IPV6 použitém ve Fedoře 14. Testovací kritéria budou primárně odvozena ze specifikace IPV6 Ready. IPV6 Ready slouží k harmonizaci všech zařízení, která protokol IPV6 používají.

Summary

My thesis describes IPV6 datagram structure, historical development and differences against IPV4. In other part I describe how it's possible to test network commnunication of Fedora 14 from view of performance, price and availability. One part of my work will be about different kinds of network stack and historical evolution. In the end of theoretical part I will describe testplan based on IEEE 829.

In practical part of my work I will do testplan for IPV6 network stack of Fedora 14. Criteria to pass will be based on IPV6 Ready program. IPV6 Ready program is standard for cooperation between network devices, which using IPV6 protocol.

Klíčová slova

IPv6, stack, testování, testplan, Ipv6-ready, RedHat, RHEL, Fedora, závěrečná práce, VUT v Brně.

Keywords

IPv6, stack, testing, testplan, IPv6-ready, RedHat, RHEL, Fedora, master thesis, BUT Brno

MALANÍK, J.*Návrh testplánu pro IPv6 stack.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2011. s. 43 s. Vedoucí bakalářské práce Ing. Petr Petyovský.

PROHLÁŠENÍ

Prohlašuji,
že svou bakalářskou práci *Návrh testplánu pro IPv6 stack.* jsem vypracoval samostatně a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne ....
(podpis autora)

Obsah

Seznam obrázků

Seznam tabulek

Seznam příkladů

1. Úvod

V teoretické části se zaměřím na popis protokolu *IP6*, jeho základní komunikační jednotky nazývané datagramy, historický vývoj a srovnání s *IPV4*. Rozeberu strukturu jeho základní hlavičky, použití volitelných hlaviček a jejich řetězení. Prozkoumám základní možnosti získání *IPV6* adresy.

V další kapitole popisuji různá síťová rozhraní nazývaná stack. Prozkoumám jejich historii a použití v různých operačních systémech. Dále provádím srovnání různých variant testování síťové komunikace Fedory 14. Srovnání testování lze provést například z pohledu výkonu, ceny nasazení a složitosti přípravy. Na závěr teoretické části rozebírám normu *IEEE 829*. Tato norma určuje, jakým způsobem vytvořit testplán, jaká má být jeho podoba a které náležitosti má testplán obsahovat.

Cílem této práce je vytvoření specializovaného testplánu pro síťové rozhraní (stack) zajišťující přístup k sítím používajícím protokol *IPV6* v operačním systému GNU/Linux Fedora 14. Ten je vyvíjen komunitou za podpory firmy Red Hat, která ho používá pro testování nových vlastností, před jejich nasazením ve své komerční distribuci RHEL (Redhat Enterprise Linux).

Testplán zakládám na obecné předloze dle normy *IEEE 829*. Na základě této obecné varianty vytvořím specializovaný dokument. Ten se zaměří pouze na nízkoúrovňovou konfiguraci a ověření připojení k síti pomocí systémových nástrojů pro protokol *IPV6*. Testplán je psán za účelem použití v rámci *QA* oddělení. Pokusím se o integraci s testy z rodiny certifikace *IPV6 Ready*.

Závěrem této práce je testplán, který je vyzkoušen na linuxové distribuci Fedora 14. Nakonec zhodnotím jeho funkčnost a navrhnu možná rozšíření mé práce.

2. Teoretický úvod

2.1. Protokol IPV6

Protokol *IPV6* je standard sloužící pro komunikaci v největší světové síti Internet. První verze tohoto standardu je z roku 1995, kdy došlo k vydání prvních *RFC*[1]. Je nástupcem starší verze nazvané *IPV4*, která byl používán od vzniku internetu. Oproti starší verzi přináší *IPV6* mnoho vylepšení a nových vlastností. Například došlo ke čtyřnásobnému zvětšení adresního prostoru. Pro představu je to $5x10^{26}$ adres pro každého z 6,5 miliardy lidí žijících na planetě. Mezi další nové vlastnosti patří nové druhy adres, podpora mobility, hierarchické směrování (slouží ke zjednodušení směrovacích tabulek) a další. V současné době (únor 2011) se tlak na zavedení *IPV6* zvýšil, protože v centrálním registru IANA už došly *IPV4* adresy. To znamená, že registrátoři pro jednotlivé světadíly už další *IPV4* adresy nedostanou a národním registrům jednotlivých zemí mohou rozdělit pouze své stávající adresy. Obrázky v této kapitole, jsem převzal z knihy [1], pokud není uvedeno jinak.

V současné době je největší problém, že uživatelé *IPV6* nevyžadují, protože skrz něj není přístupný žádný bonusový obsah, oproti *IPV4* síti. ISP *IPV6* neimplementují, protože ho zákazníci nevyžadují. Další velkou nevýhodou je zpětná nekompatibilita s *IPV4*, což znamená, že při přechodu bude nutné vyměnit většinu koncových zařízení. Velkým problémem je i pomalý vývoj některých standardů například mobilita (možnost použít svou *IPV6* adresu na celém světě) nebo ve světě *IPV4* dobře známý dhcp.

2.1.1. Certifikace *IPV6 Ready*

IPv6 Ready Logo slouží k jednoduchému rozlišení, jak je daná služba nebo produkt připravena na provoz v síti *IPv6*. Poradní výbor kolem *IPV6 Ready Logo* byl založen sdružením *IPV6 Forum* v roce 2002 a slouží k řízení a rozvoji této certifikace. Základním důvodem pro existenci sdružení *IPV6 Forum* je vývoj, politická propagace a koordinace postupného nasazování *IPV6*. Přínos certifikace *IPV6 Ready Logo* je ve třech oblastech uvedených v Tab. 2.2. Pro získání *IPV6 Ready Logo* není nutné stát se členem skupiny *IPV6 Forum*. Existují dvě varianty *IPV6 Ready Logo* (do budoucna je plánována třetí), které lze získat. Pro získání loga je třeba stoprocentně splnit řadu testů. Specifikace jednotlivých skupin testů jsou velmi rozsáhlé. Pro první fázi existuje dokument obsahující testy konfigurace a spolupráce mezi stroji v síti (dokument s názvem *Core_Interoperability_Latest*) a pak rozsáhlejší dokument obsahující testy na správnost

[1] *Request For Comments* je sada norem různé důležitosti (některé závazné, jiné pouze informační) norem definujících základní vlastnosti síťové komunikace. Normy je možné najít na internetových stránkách [4] a výběr norem důležitých pro tuto práci je v elektronických přílohách této práce.

Tab. 2.1: Seznam fází certifikace *IPV6 Ready*.

Fáze 1	Nazývaná též Stříbrná, logo je uvedeno na Obr. 2.1, byla specifikována 1. Září 2003. Pro její splnění je třeba projít přibližně 170 testů[2]. Obsahem testů je kontrola implementace *IPV6*, *ICMPv6*, Objevování sousedů a automatická konfigurace rozhraní.
Fáze 2	Nazývaná též Zlatá, logo je vidět na Obr. 2.1, byla specifikována 15. Února 2005. Pro získání je třeba splnit asi 450 testů[3]. Jejich obsahem jsou části obsažené v normách RFC pod označením *MUST*, *SHOULD*.
Fáze 3	ještě nemá stanovenou barvu loga a zatím nebyla ani spuštěna. Pro její získání bude třeba ověřit funkčnost technologie *IPSEC*.

hlavičky *IPV6* datagramu (dokument s názvem *Core_Conformance_Latest*). Oba tyto dokumenty lze najít v elektronických přílohách této práce nebo na webu [3]. Popis obou fází lze najít v Tab. 2.1

Na světě je v tuto chvíli (březen 2011) 6 oficiálních certifikovaných laboratoří, kde lze získat logo. Čtyři se nacházejí v Asii, po jedné v Spojených státech Amerických a v Evropě.

Obr. 2.1: Stříbrné a Zlaté logo projektu pro testování *IPV6 Ready*. Loga jsou dostupná z domovské stránky projektu [3]

Tab. 2.2: Hlavní cíle programu *IPV6 Ready Logo*

1.	Ověření a srovnání funčnosti jednotlivých implementací *IPv6*.
2.	Poskytnutí testovacích nástrojů.
3.	Zajištění provozu několika testovacích laboratoří umístěných po celém světě pro oficiální certifikaci.

[1]Podle oficiálních stránek projektu[3]. Přesný počet testů lze najít na stránce s technickými informacemi.

2.1.2. Certifikace *IPV6 Ready* - Fáze 1

V 1. fázi testování proběhne nejdříve ověření práce s různými druhy *IPV6* adres. Jsou to lokální linkové, globální unikátní. V první fázi klient nemusí umět pracovat s multicast adresami. Následující kapitola je překladem referenčních dokumentů od sdružení *IPV6 ready* [3], originální dokument lze najít v elektronických přílohách této práce. Jako příklad jsem kompletně přeložil první test, kompletní znění i s přesným postupem dalších testů lze najít na webu [3], nebo v elektronických přílohách této práce.

Testy pro Fázi 1 jsou *Ping test*, *Autokonfigurace a detekce duplicitních adres*, *Zpracování nabídek autokonfigurace od serveru - určení prefixu*, *Zpracování nabídek autokonfigurace od serveru - Router Lifetime*[4], *Přesměrování komunikace přes jiný Směrovač*, *Nastavení MTU*[5] a *Fragmentace*.

V Tab. 2.3 je uveden seznam částí *Ping testu*. Dáleje uveden popis testu a znázornění jeho časového průběhu.

Tab. 2.3: Seznam částí Ping testu v souvislosti s Fází 1,2 certifikace *IPV6 Ready*.

jméno testu	Klienti	Směrovače	Fáze 1	Fáze 2	typ adresy
1a	x	-	x	x	linková lokální
1b	x	-	x	x	globální
1c	x	-	-	x	multicast
1d	x	x	x	x	linková lokální
1e	x	x	x	x	globální
1f	x	x	x	x	multicast
1g	-	x	x	x	linková lokální
1h	-	x	x	x	globální
1i	-	x	x	x	globální

Účel: Ověření funkčnosti *IPV6 ping* v obou směrech (klient vs. klient, klient vs. směrovač)

Požadavky:

- Sledování provozu na síti a filtrace *IPV6* datagramů.
- Zkompilovaný program *ping* podporující *IPV6* protokol.

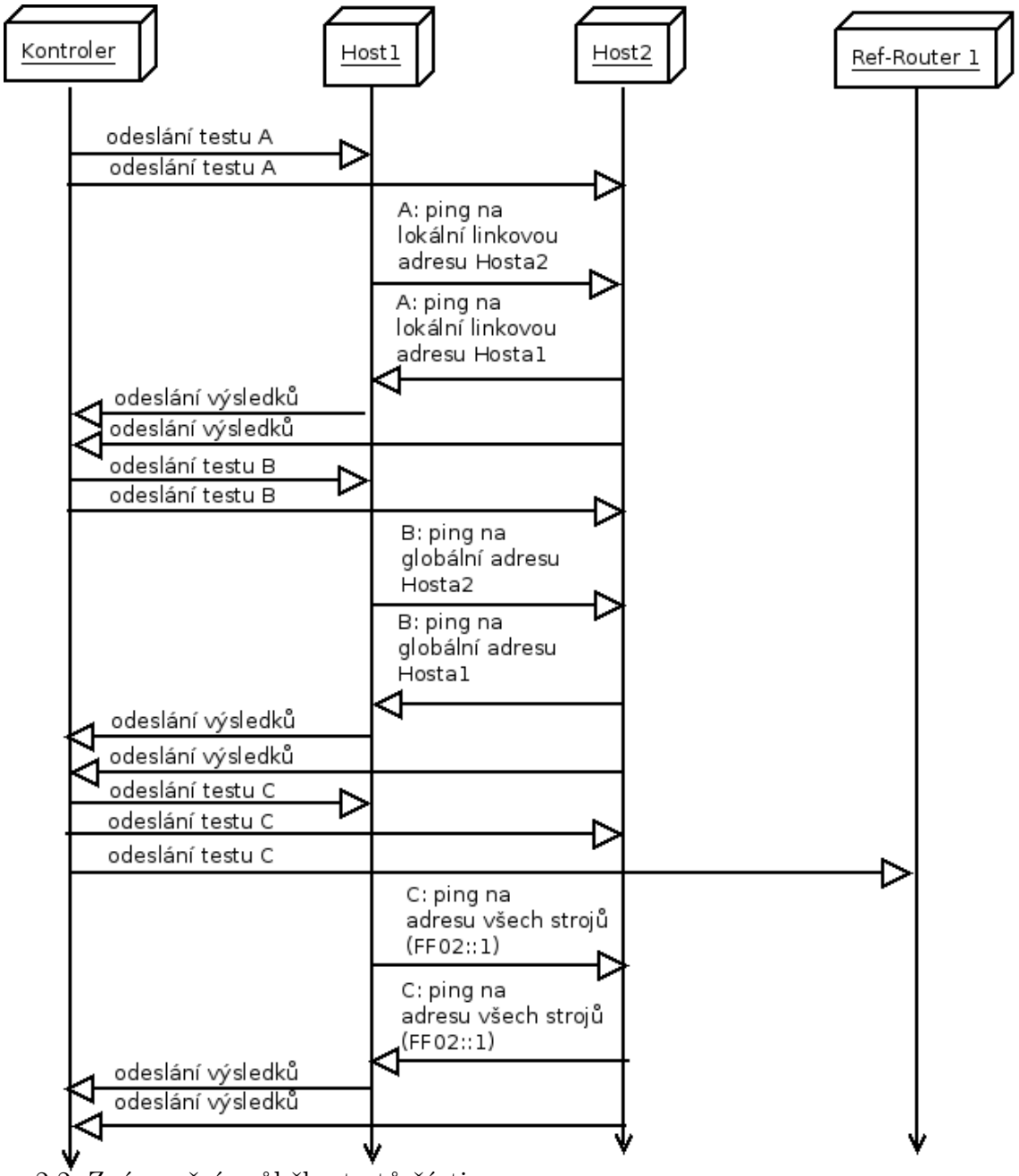
Ve fázi jedna není nutné, aby klienti a směrovače uměli pracovat s multicast adresami, adresou všech uzlů v síti (*ff02::1*) a adresou všech směrovačů v síti (*ffo2::2*).

Diskuze Na každém stroji v síti musí být program, který je schopen odeslat a přijmout *IPV6* provoz, je možné ho rozšířit o testovací grafické rozhraní. Odpověď na *IPV6* multicast adresu je dobrovolná, pokud k ní dojde, musí být odeslána z unikátní adresy,

[4]Udává jak dlouho je daná nabídka na konfiguraci platná

[5]Maximální velikost datagramu, který je možno přenášet

která náleží stejnému rozhraní, na kterém byla přiřazena příjímající *multicast* adresa. Na Obr. 2.2,2.3, 2.4 je znázorněn časový průběh ping testu.



Obr. 2.2: Znázornění průběhu testů části a-c

2.1.3. Certifikace *IPV6 Ready* - Fáze 2

Je mnohem rozsáhlejší a testuje široké spektrum vlastností protokolu *IPV6*. V následující Tab. 2.4 je přehled testovaných vlastností *IPV6*. Detailní informace o těchto funkcích lze nalézt v *RFC*, dostupných v přílohách této práce nebo na webu *ietf* [4]. Protože je rozsah těchto testů velmi široký, neuvádím zde detailní informace. Ty lze najít elektronických přílohách, nebo na webu[3].

Tab. 2.4: Seznam vlastností testovaných v druhé fázi certifikace *IPV6 Ready*

-	Jádro protokolu IPV6
IPSEC	Je protokol sloužící k zabezpečení *IPV6* komunikace pomocí šifrování (zakóduje data tak, aby je nikdo nepovolaný nemohl přečíst) a autentizace (ověří, že data opravdu pocházejí od originálního zdroje). Narozdíl od komunikace protokoly *SSH*, *SSL/TLS*, *IPSEC* zabezpečuje veškerý provoz mezi dvěma klienty, sítěmi nebo mezi klientem a sítí.
MIPV6	Slouží k pohybu klientů skrz *IPV6* sítě a jejich dostupnost na původní adrese. Kompletní znění lze najít v *RFC 3963* a *RFC 3776*.
NEMO	Je rozšířením mobility v *IPV6*. Slouží k udržení spojení pro celou síť mobilních klientů, namísto jednoho stroje. Kompletní znění lze najít v *RFC 3963*.
DHCPV6	Slouží k přidělování *IPV6* adres klientům. Kompletní znění lze najít v *RFC 3315*.
SIP	Slouží k navazování multimediálních spojení, buď mezi dvěma klienty, nebo mezi skupinou. Kompletní znění lze najít *RFC 3261*.
SNMP	Slouží ke sledování a diagnostice sítí. Kompletní znění pro sítě standardu 802.x lze najít v *RFC 4789*.
IMS UE	Slouží k doručování multimedialního obsau koncovým uživatelům, původně byl určen pro přenos pomoci *GSM*.
MLDv2	neboli *Multicast Listener Discovery* Slouží k nalezení optimálních cest pro skupinovou komunikaci. *MLD* se vyskytuje ve dvou verzích, verze jedna je protipólem *IGMPv2*, verze dva odpovídá *IGMPv3*. Výhodou *MLDv2* oproti *MLD* je možnost specifikovat nejen od které skupiny chci dostávat data, ale například i konkrétní stroj. Detailní informace lze najít v kapitole osm, strana 151, knihy IPV6[1] od Pavla Satrapy, nebo v *RFC 3810*.

2.1.4. Struktura datagramu

Protože se *IPV6* adresy a podoba datagramu oproti *IPV4* výrazně změnila, bylo by dobré si je krátce představit. Největší změnou u datagramů je snaha o maximální zjednodušení. K tomu došlo díky nové vlastnosti „řetězení hlaviček“, kdy je většina informací přesunuta do volitelných hlaviček, které se přidávají dle potřeby. Hlavní změnou u adres je jejich prodloužení, což výrazně zvětšilo použitelný adresový prostor.

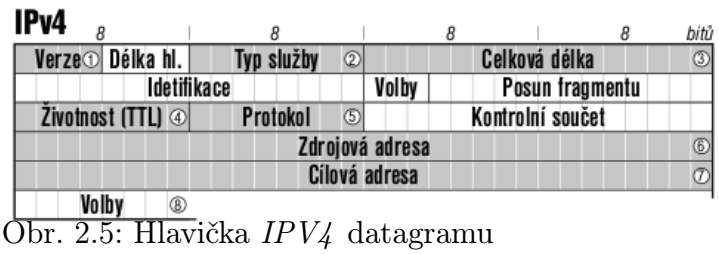
Hlavička datagramu[6] *IPV6* protokolu byla oproti *IPV4* značně zjednodušena. Nyní obsahuje jen základní informace a adresy. Další informace je možné vložit pomocí voli-

[6]Datagram, neboli paket (v angličtině packet) je datová struktura vycházející z ISO/OSI modelu a přijatá pro *TCP/IP* model. Odpovídá datům přenášeným na třetí, neboli síťové vrstvě.

telné položky „další hlavička“, která umožňuje řetězit další hlavičky podle definovaných pravidel a v pořadí, v jakém je budou pravděpodobně stroje po cestě zpracovávat. Proto je například hlavička s informacemi pro adresáta až poslední a po cestě se jejím zpracováním směrovač nemusí zdržovat. A i když se délka adres zvětšila čtyřikrát, celková velikost první hlavičky se zvětšila jen dvakrát z dvaceti bajtů na čtyřicet bajtů. Na Obr. 2.5, 2.6 které jsem převzal z [1] je vidět struktura hlaviček *IPV4* a *IPV6* datagramu. V Tab. 2.5 je popis jednotlivých položek základní hlavičky.

Tab. 2.5: Parametry základní hlavičky protokolu *IPV6*.

Zdrojová a cílová adresa	Udává $2x128$ bitovou adresu odesílatele a příjemce datagramu.
Verze	udává ve všech datagramech verzi dat, kterou obsahují.
Třída provozu	V budoucnosti bude umožňovat třídit provoz podle daných parametrů, což by umožnilo garantovat QOS[7]. V současné době není položka blíže specifikována, pouze se vyžaduje implicitní hodnota nula.
Značka toku	Umožní zrychlení zpracování proudu dat, který bude mít podobné vlastnosti. Například společný odesílatel/příjemce, což umožní rychlejší zpracování datagramu a zvýšení propustnosti routeru.
Délka dat	Je dvoubajtová položka určující velikost datagramu. Základní hlavička se do ní nepočítá, kdežto rozšiřující ano. Pokud je třeba vytvořit delší datagram, je možné použít rozšiřující hlavičku *Jumbo obsah* která bude popsána později.
Další hlavička	Říká, jaká další hlavička má být zpracována. Poslední hlavička obsahuje typ přenášených dat rozdělený podle protokolu.
Dosah	Nahradil dřívější položku TTL[8]. Udává kolik průchodů může datagram absolvovat. Při vynulování je odesílatel pomocí *ICMP* zprávy informován, že datagram nebyl doručen.



Obr. 2.5: Hlavička *IPV4* datagramu

Nyní se podíváme na použití hlaviček a jejich řetězení. Na rozdíl od starší verze protokolu jsou ”provozní informace” protokolu rozděleny do několika hlaviček, každý datagram proto přenáší pouze nezbytně nutné informace (například většina datagramů nepotřebuje informace o šifrování a autentizaci), ikdyž v rámci zvyšování zabezpečení by šifrování mělo být nasazováno co nejvíce. A tyto hlavičky, jak už bylo řečeno dříve, jsou na sebe na-

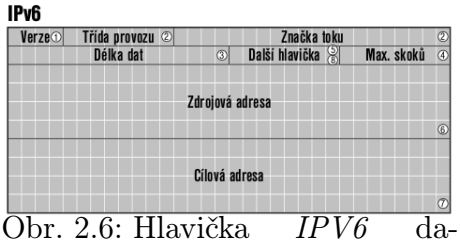
pojeny pomocí položky další hlavička. Možné rozšiřující hlavičky jsou v Tab. 2.6 uvedeny v pořadí v jakém musí být uvedeny za základní hlavičkou.

Tab. 2.6: Rozšiřující hlavičky protokolu *IPV6*.

0	Volby pro všechny stroje po cestě, neboli *hop-by-hop options*
60	Volby pro první cíl směrování
43	Směrování
44	Fragmentace
51	Autentizace
50	Šifrování obsahu
60	Volby pro skutečný cíl směrování
135	Mobilita

2.1.5. Hlavička fragmentace

Každé přepravní médium pro protokol *IPV6* má danou jistou maximální velikost datagramů, které je schopno přenášet *MTU*[9]. Narozdíl od IPV4 fragmentaci smí provádět pouze odesílající uzel. Pokud během přenosu přes síť IPV6 má některá trasa menší MTU, bude datagram zahozen a odesílatel bude pomocí *ICMP* zprávy (její součástí je i velikost MTU, které chybu způsobilo) vyrozuměn o chybě. Druhým rozdílem oproti *IPV4* je volitelnost, takže pokud není použita hlavička fragmentace, při přepravě datagramu se s ní vůbec neuvažuje. Rozšiřující hlavička pro fragmentaci má *kód 44* v položce *další hlavička* předchozí hlavičky.



Obr. 2.6: Hlavička *IPV6* datagramu

Struktura hlavičky fragmentace je patrná z Obr. 2.7, který jsem převzal z [1]. Na začátku je 8 bitů pro položku *další hlavička*.

Obr. 2.7: Struktura hlavičky Fragmentace

Následuje osmibitová rezerva, která v současné době není využívána. Položka identifikace slouží k jednoznačné identifikaci (pomocí 32bitového celého čísla) skupiny datagramů, patřících k sobě. V rámci dvojice *odesílatel - příjemce* by mělo toto číslo být jedinečné. Každý další fragmentovatelný datagram by měl mít identifikační číslo o jedničku větší a po přetečení se začne znova od nuly. Jednotlivé fragmenty původního datagramu jsou identifikovány pomocí *posunu fragmentu*. Tento pomocí osmice bajtů udávající vzdálenost od začátku fragmentovatelné části původního datagramu, určuje konkrétní umístění

[9] *Max Transfer Unit* neboli maximální velikost datagramu, který bude přenášen.

fragmentu. Na konci prvních 4 bajtů jsou tři bity, z toho dva rezervní, dnes nevyužívané. Poslední bit slouží k signalizaci, zda je tento fragment poslední (poslední bit je roven nule), nebo zda následuje další fragment, pak je poslední bit čtvrtého bajtu roven jedné.

Pokud má dojít k fragmentaci původního datagramu, je datagram rozdělen na dvě části. Nefragmentovatelná, kterou obsahují všechny dílčí fragmenty a fragmentovatelná. V nefragmentovatelné části jsou všechny úvodní hlavičky až po *Směrování*. Zbytek datagramu je považován za fragmentovatelný a je rozdělen do jednotlivých segmentů. Velikost dat ve fragmentovatelné části (ta část, co se liší v jednotlivých fragmentech), je určována jako násobek osmi bajtů a výsledná velikost datagramu (i s povinnou nefragmentovatelnou částí) je menší, než nejmenší *MTU* na trase. Jediná změna v nefragmentovatelné části je přepočítaní skutečné délky fragmentu datagramu a změna hodnoty poslední *Další hlavičky* na kód *44*. Za ni se přidá hlavička *Fragmentace*, která se naplní následovně: Vygeneruje se nový *Identifikátor*, kterým se označí všechny fragmenty. Hodnota další hlavičky se vezme z původního datagramu. Určí se posun fragmentu v původním datagramu, jak bylo popsáno výše. Poslednímu fragmentu se příznak *M* nastaví na nula, ostatním na jedna. Na konec se připojí původní fragment. Na základě údajů z této hlavičky, je adresát schopen fragmenty složit, nechat si znova poslat ztracené části a datagram dále zpracovávat jako jeden celek, proto mohou hlavičky fragmentace zaniknout.

2.1.6. Typy směrování

V nejběžnějším případě probíhá směrování podle cílové adresy v první hlavičce. Pomocí rozšiřující hlavičky *Směrování* je možné zadat průchozí uzly, přes které datagramy musí téct. V současné době se pro běžný provoz používají dva (typ nula a typ dva) a dva *volné* pro testování nových vlastností. V Tab. 2.7 je uveden podrobnější popis.

2.1.7. Připojení k síti

Pro připojení k *IPV6* síti je potřeba získat adresu (používané typy adres jsou v Tab. 2.8. K tomu narozdíl od *IPV4* slouží několik metod. Obdobně jako u starší varianty zůstala možnost manuální konfigurace, která je velmi nepohodlná kvůli délce adresy a velkému riziku překlepu. Nově přibyla možnost bezestavové konfigurace. Ta umožňuje připojení klienta do sítě bez pomoci serveru. Teprve později na žádost uživatelů a kvůli nedostatkům bezestavové konfigurace byla přidána také možnost použít *DHCPV6*. Obě možnosti jsou popsané v kapitole 2.1.7. U *IPV6* se stejně jako u *IPV4* adresy přidělují přímo síťovým rozhraním, nikoliv počítačům. V protokolu *IPV6* ale každé rozhraní dostane přiděleno několik různých adres. V následující Tab. 2.8 jsou uvedeny tři typy adres s různým dosahem a chováním. Oproti *IPV4* zmizely oznamovací, neboli *broadcast adresy*. Byly nahrazeny speciálním druhem adres, nazvaným *skupinové adresy*.

Tab. 2.7: Typy směrování používané v protokolu *IPV6*.

Směrování typ nula	Slouží jako seznam bodů, přes které musí datagram v libovolném pořadí a s libovolným počtem meziskoků projít. Při každém průchodu zadaným uzlem je tento označen a z dalšího rozhodování o trase vynechán. V roce 2007 se objevil bezpečnostní problém, který vedl k odmítnutí směrování typu *0*, takže se dnes už nesmí používat. Problémem je možnost zneužití k zahlcení dané trasy v síti. Útočník použije několik směrovacích tabulek o maximální velikosti, a tím je schopen zahltit i mnohem silnější linku než jakou sám disponuje.
Směrování typu dva	Slouží pro mobilitu. Každá mobilní stanice má svou domovskou adresu a dále druhou, která se mění při pohybu mezi různými *IPV6* sítěmi. Hlavička obsahuje trvalou domácí adresu cílové stanice, ale datagram je doručen na dočasnou adresu. Teprve zde dojde k nahrazení dočasné adresy trvalou adresou a vyšší komunikační vrstvy se nedozvědí, že se mění současná *IPV6* adresa.

Tab. 2.8: Typy adres použitých v protokolu *IPV6*.

Individuální	Jedná se o ip adresy v původním smyslu IPV4, nazývané též unicastové adresy.
Skupinové	Neboli multicast adresy slouží k adresování skupiny počítačů. Data musí být doručena všem členům.
Výběrové	Slouží k identifikaci skupiny pc, ale narozdíl multicast adres, jsou data doručena pouze nejbližšímu členovi. Anglický název je *anycast*.

Existují dva základní způsoby jak získat svoji *IPV6* adresu. Stavová a bezestavová konfigurace. Stavová konfigurace je klasický přístup, jak ho známe z *IPV4*, kde se podobných technologií postupně používalo několik (RARP[10], BOOTP[11], DHCP[12]). V principu jde o zaslání žádosti o konfigurační údaje klientem na obecnou adresu a server mu je ve své odpovědi sdělí.

2.1.8. Bezestavová konfigurace

Narozdíl od starého přístupu *bezestavová konfigurace* využívá úplně nový přístup. Vychází z toho, že v každé síti je směrovač, který ví vše potřebné a vždy po vypršení intervalu údaje rozesílá klientům. Nový klient si na ně může počkat nebo aktivně požádat.

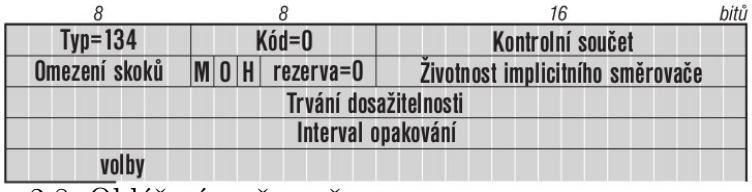
[10] *Reverse address resolution protocol* slouží ke zjištění *IPV4* adresy na základě znalosti vlastní hardware adresy

[11] První protokol sloužící k přidělování síťových adres. Předchůdce *DHCP*, definovaný v RFC 951.

[12] Protokol sloužící k přidělování síťové konfigurace pro klienty, který nahradil *BOOTP*, definovaný v RFC 2131.

Základním prvkem bezestavové konfigurace je *ohlášení směrovače (Router advertisement)*, které v náhodných intervalech posílají všechny směrovače v síti. Náhodné intervaly se používají, proto aby si klienti neustále neměnili konfiguraci.

Na Obr. 2.8, který jsem převzal z [1], je vidět struktura paketu s ohlášením směrovače. Chybějící část tlusté čáry, není chyba grafiky, ale naznačuje, že paket není zobrazen celý. Patří do rodiny servisních paketů *ICMP*. Nejdůležitější položkou paketu je *Životnost implicitního směrovače (Router Lifetime)*, který v jednotkách sekund udává, jak dlouho daný směrovač ještě bude sloužit jako defaultní. Pokud je nula, neměl by nadále být používán. Další údaje slouží k určení maximálního počtu skoků *Omezení poču skoků (Cur hop limit)*. Za ním následuje osm příznaků, ze kterých jsou zatím definované pouze tři. První dva se týkají DHCPV6. Příznak *M (Managed address configuration, stavová konfigurace adres)* oznamuje, že další komunikační parametry přidělí DHCPV6. Za ním následuje příznak *O (Other stateful configuration, stavová konfigurace dalších parametrů)*, který rozhoduje o použití DHCPV6 i pro další parametry sítě (lokální dns, ap.). Možné kombinace příznaků shrnuje Tabulka 2.1.8.



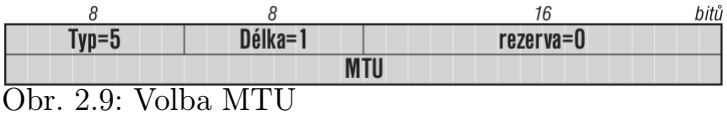
Obr. 2.8: Ohlášení směrovače

Tab. 2.9: Kombinace parametrů pro datagram ohlášení směrovače

M	**O**	Význam
1	-	DHCPV6 poskytne vše
0	**1**	kombinovat s bezestavovou konfigurací (pro adresu, prefix a směrování), DHCPV6 poskytne vše ostatní
0	**0**	DHCPV6 není k dispozici

Příznak *H* slouží k podpoře mobility, kterou se tato práce nezabývá. Další dva údaje pevné části ovlivňují detekci sousedů, jde o časové údaje v milisekundách. První *Trvání dosažitelnosti (Reachable Time)* udává jak dlouho má soused být považován za dostupného po poslední komunikaci. Interval mezi dvěma výzvami k sousedovi se nazývá *Interval opakování (Retrans Timer)*.

Ve volbách může směrovač sdělit svou linkovou adresu, MTU sítě (Obr. 2.9) a použije jednu volbu pro každý prefix logické sítě (Obr. 2.10). Z toho vyplývá, že se už při návrhu *IPV6*, počítalo s více logickými sítěmi na jedné fyzické síti.



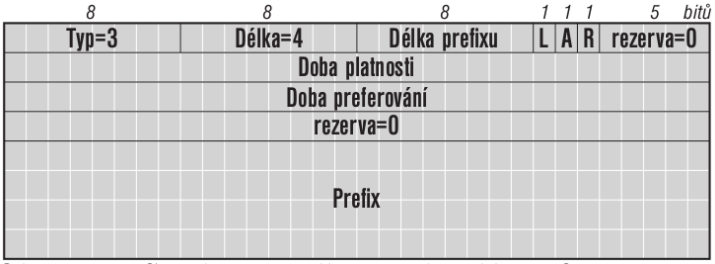
Obr. 2.9: Volba MTU

Doplňkové informace k autokonfiguraci (včetně samotných adres) je možné získat pomocí *DHCPV6*.Další možností jak získat *IPV6* adresu je *DHCPV6 (Dynamic Host Con-*

figuration Protocol), pomocí nějž lze kompletně nastavit údaje potřebné pro připojení k síti. Konfigurace sítě pomocí *DHCPV6* je uvedena v Tab. 2.10.

Tab. 2.10: Průběh komunikace *DHCPV6*.

Objevování (discover)	Klient na všesměrovou adresu pošle žádost obsahující jeho mac adresu.
Nabídka (offer)	Běžně jeden, obecně více serverů, na místní síti klientovi nabídnou konfiguraci.
Požadavek (request)	Klient si vybere vhodnou nabídku a pošle žádost o nabídnuté parametry.
Potvrzení (acknowledge)	Server klientovi potvrdí dočasné zapůjčení (lease) parametrů na zadaný čas, po jeho uplynutí klient musí požádat o prodloužení, nebo nabídku nových parametrů.



Obr. 2.10: Struktura volby pro každý prefix

Ve světě *IPV6* se protokol *DHCP* změnil. Již nejsou využívány všesměrové adresy a klient si sám dokáže vygenerovat lokální linkovou adresu. Při komunikaci protokolem *DHCP* jsou zapotřebí tři typy strojů. Klient je koncové zařízení, které žádá o konfiguraci. Server je centrální zařízení vyřizující žádosti. Mezi nimi může být *relay*, což je zařízení které předává komunikaci, pokud klient a server nejsou na jedné lince. Pokud funkci *relay* a serveru sloučíme, vznikne takzvaný *agent*.

Při používání *DHCPV6* je velmi důležitá identifikace účastníků, k té dříve sloužila mac adresa. Ve světě *DHCPV6* se používá pojem *DUID (DHCP Unique Identifier)* pro identifikaci stroje a *IA (ientity asociation)*, sloužící k identifikaci jednotlivých rozhraní. V současné době existují tři možnosti jak ho vytvořit (standard počítá s možností přidat další varianty). První možností je klasické řešení pomocí mac adresy, toto řešení funguje obdobně jako u *IPV4*. Zde je ovšem problém, že *DUID* by se neměl měnit, což je problém, protože s výměnou síťové karty se změní. Druhou možností je využití linkové adresy a času vytvoření. Pokud je použita tato možnost, je nutné, aby zařízení mělo paměť, kde je možné takový identifikátor uložit. Poslední možností je využití výrobního čísla zařízení a domény výrobce. Pro generování *IA* se používá množina konfiguračních informací, které klient danému rozhraní nastaví. Komunikační fáze se příliš nezměnily. Klient zahájí komunikaci vygenerováním *IA* identifikátorů, pak odešle žádost *(solicid)* na skupinovou adresu agentů uvedenou v Tab. 2.11.

Tab. 2.11: Skupinové adresy pro získání ipv6 adresy pomocí *DHCPV6*

ff02::1:2	všichni agenti a servery
ff05::1:3	všechny servery

Klient do žádosti přidá též svůj *DUID* a *AI* všech svých rozhraní, pro které chce získat adresy. Součástí žádosti je také jeho linková adresa (s prefixem *fe80::*). Podmínkou funkčnosti je minimálně jeden agent pro každou síť. Na Obr. **??** je vidět struktura paketu pro *DHCPV6* protokol. Oproti starší verzi byl maximálně zjednodušen a zůstaly jen dvě povinné položky. Těmi jsou *typ*, identifikující o jakou zprávu vlastně jde, a *identifikátor relace (Transaction ID)*, používané pro identifikaci jednoho spojení. Ostatní informace se přesunuly do voleb.

V případě, že na lokální síti je server, použije lokální linkovou adresu klienta a odpoví. Server použije typ zprávy *dva*, *ohlášení serveru (advertise)* z Tab. 2.12. Server do odpovědi přidá také nastavení, která by nabídl jednotlivým *IA* a může doplnit preferenci, která udává ochotu serveru s klientem spolupracovat.

Tab. 2.12: Seznam možných voleb pro *DHCPV6* datagram

1	výzva (solicit)
2	ohlášení serveru (advertise)
3	žádost (request)
4	potvrzení (confirm)
5	obnovení (renew)
6	převázání (rebind)
7	odpověď (reply)
8	uvolnění (release)
9	odmítnutí (decline)
10	rekonfigurace (reconfigure)
11	žádost o informace (information request)
12	předání (relay forward)
13	zprostředkovaná odpověď (relay reply)

Pokud žádost dorazí ke zprostředkovateli (*Relay*), ten ji přepošle na servery z předem definovaného seznamu. Přeposlání výzvy (*solicit*) proběhne pomocí nové zprávy typu předání (*relay forward*). Server odpoví na výzvu pomocí zprostředkované odpovědi (*relay reply*). Zprostředkoval zprávu vybalí a pošle na lokální adresu klienta. Ze všech odpovědí které dorazí, si klient vytvoří seznam a vybere nejvhodnější konfiguraci (měl by zvolit tu s nejvyšší preferencí serveru, v případě shody si může vybrat dle svého uvážení).

Po výběru serveru nastává druhá fáze. Klient odešle žádost (*request*), ve které uvede DUID serveru získané z ohlášení. Protože zatím stále oficiálně nezná parametry místní sítě, pošle ji na všeobecnou adresu všech *DHCPV6* agentů. Servery, kterých se žádost netýká ji samozřejmě ignorují. Server odpoví pomocí zprávy typu odpověď (*reply*). Při sestavování

parametrů pro klienta použije hlavní fyzickou síť, kde se klient nachází a DUID klienta. Protože klient vybíral z pozitivních reakcí serverů, je odmítnutí nepravděpodobné.

Klient si zaslané adresy může ověřit pomocí standartního mechanismu pro detekci duplicitních adres a pokud detekuje, že adresy už někdo používá, může je odmítnout pomocí odmítnutí (*decline*).

Podobně jako u *IPV4* je síťová konfigurace přidělována na předem definovaný čas. Pokud klient chce své parametry používat i nadále musí požádat o prodloužení. Jeho stávající parametry může potvrdit, jak původní server, který mu je přidělil pomocí zprávy obnovení (*renew*), tak i nový server, pomocí zprávy převázání (*rebind*). Pokud klient končí v síti, měl by nahlásit uvolnění adres (*release*). V případě, že klient neuvolnil svou adresu a pouze se dočasně odpojil od sítě (například v případě suspendu, nebo restartu), musí ověřit platnost své konfigurace pomocí potvrzení (*confirm*). Server zodpovědný za danou konfiguraci žádost buď odmítne, nebo potvrdí.

Nejběžnějším případem je žádost o konfiguraci ze strany klienta. Specifikace *IPV6* ovšem pamatuje i na možnost, že komunikaci potřebuje zahájit server, například v případě změny síťových parametrů (změna výchozí brány). Tyto situace jsou ošetřeny pomocí zprávy rekonfigurace (*reconfigure*). Žádost posílá individuálně každému klientovi, kterého se změna týka (jejich adresy má uloženy v databázi přidělených údajů). Klient odpoví pomocí požadavku na obnovení svých parametrů a server v odpovědi sdělí novou konfiguraci.

2.1.9. Základy routování

Elementární routování musí zvládat každý stroj připojený do nějaké sítě, *IPV6* není výjimkou. Základem je směrovací tabulka obsahující několik záznamů, které jsou vidět v tabulce 2.13. Směrovací tabulka je závislá na použitém operačním systému, například *Microsoft Windows Vista* automaticky vytváří i *tunelová zařízení*[13], proto je jeho směrovací tabulka rozsáhlejší. Zadaný cíl pro směrování může ležet buď přímo v dané síti, pak jsou data předána přímo jemu, nebo je ve směrovací tabulce uveden stroj, kterému data poslat a on zařídí doručení. Záznam číslo jedna se musí vyskytovat na každém zařízení. Jde o lokální smyčku (*loopback - lo*)[14] Druhý a třetí záznam zajišťují směrování vlastní lokální linkové adresy a strojů připojených k místní síti. Druhý záznam, vychází z hardware adresy síťové karty a ukazuje na rozhraní *zpětné smyčky*, protože komunikace neopouští klientský stroj. Třetí záznam určuje, že komunikace pro místní síť (která končí

[13]Počítač, který nemá přímý přístup(nativní připojení) k *IPV6*, se může připojit pomocí *IPV4* k serveru, který *IPV6* připojení má, a komunikace bude probíhat skrz tento server.

[14]Jedná se o základní síťové zařízení, vyskytující se ve všech zarízeních používajících síť. Používá se například k testování konfigurace nebo komunikaci s lokálně běžícími servery.

Tab. 2.13: Příklad směrovací tabulky klienta

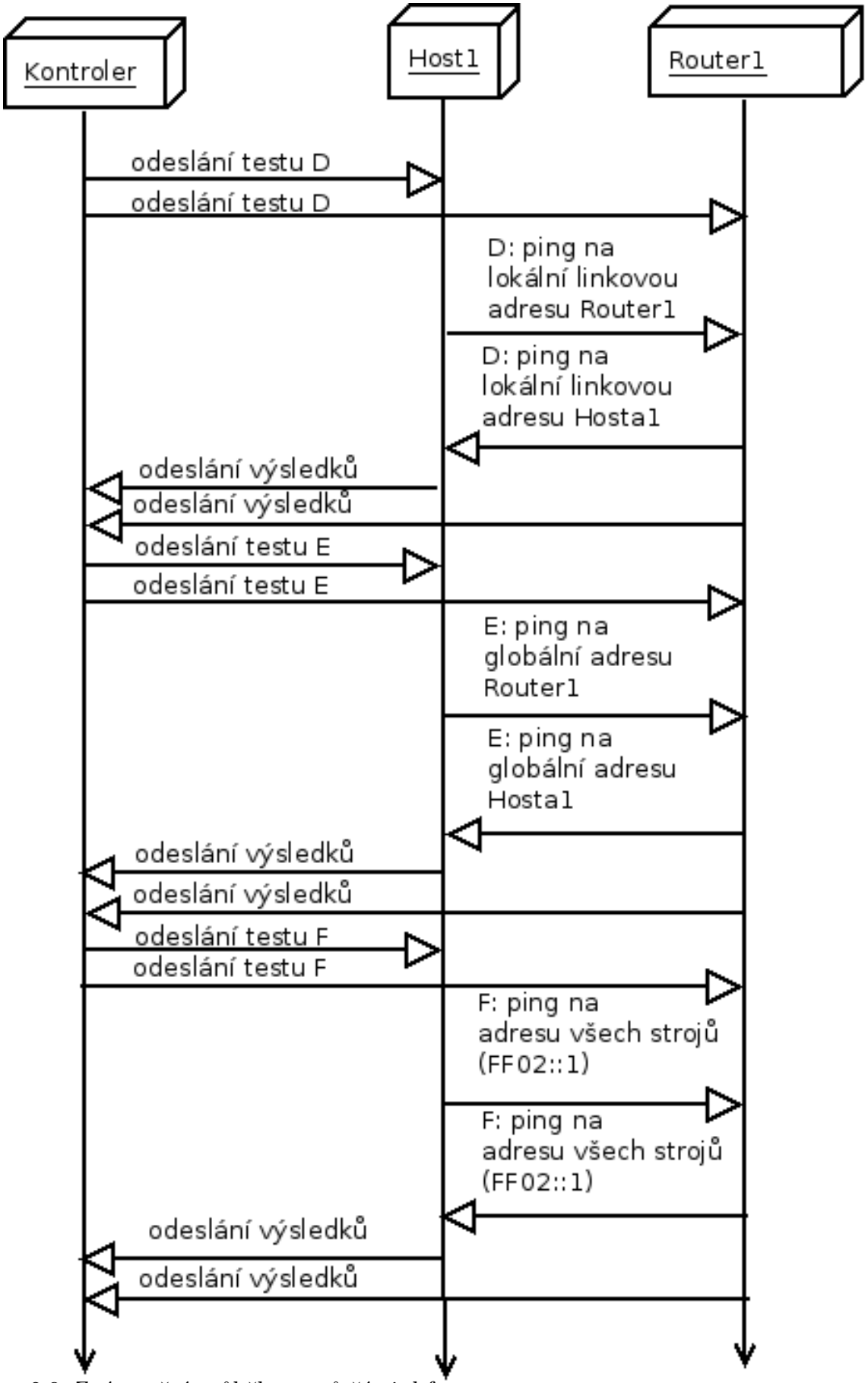
číslo záznamu	cílová adresa	adresa pro předání	rozhraní
1	::1/128	::	lo
2	fe80::1020:3040:5060	::	lo
3	fe80::/64	::	eth0
4	2001:15c0:65ff:17a::1020:3040:5060/128	::	eth0
5	2001:15c0:65ff:17a::/64	::	eth0
6	ff00	::	eth0
7	::/0	fe80::1010:1010:1010	eth0

hraničním směrovačem)[15] se má poslat přímo skrz rozhraní eth0[16]. Čtvrtý a pátý záznam v tabulce řeší to samé pro globální adresy počítače. Šestý záznam v tabulce určuje, že skupinově adresované datagramy (které mají prefix *ff00::*) mají být předány přímo na rozhraní eth0. Poslední záznam v tabule udává, co se má stát s daty, ktetým nevyhovuje žádný předchozí záznam. Říká se mu implicitní záznam (*default route*) a udává adresu, na kterou poslat veškerou zbývající komunikaci.

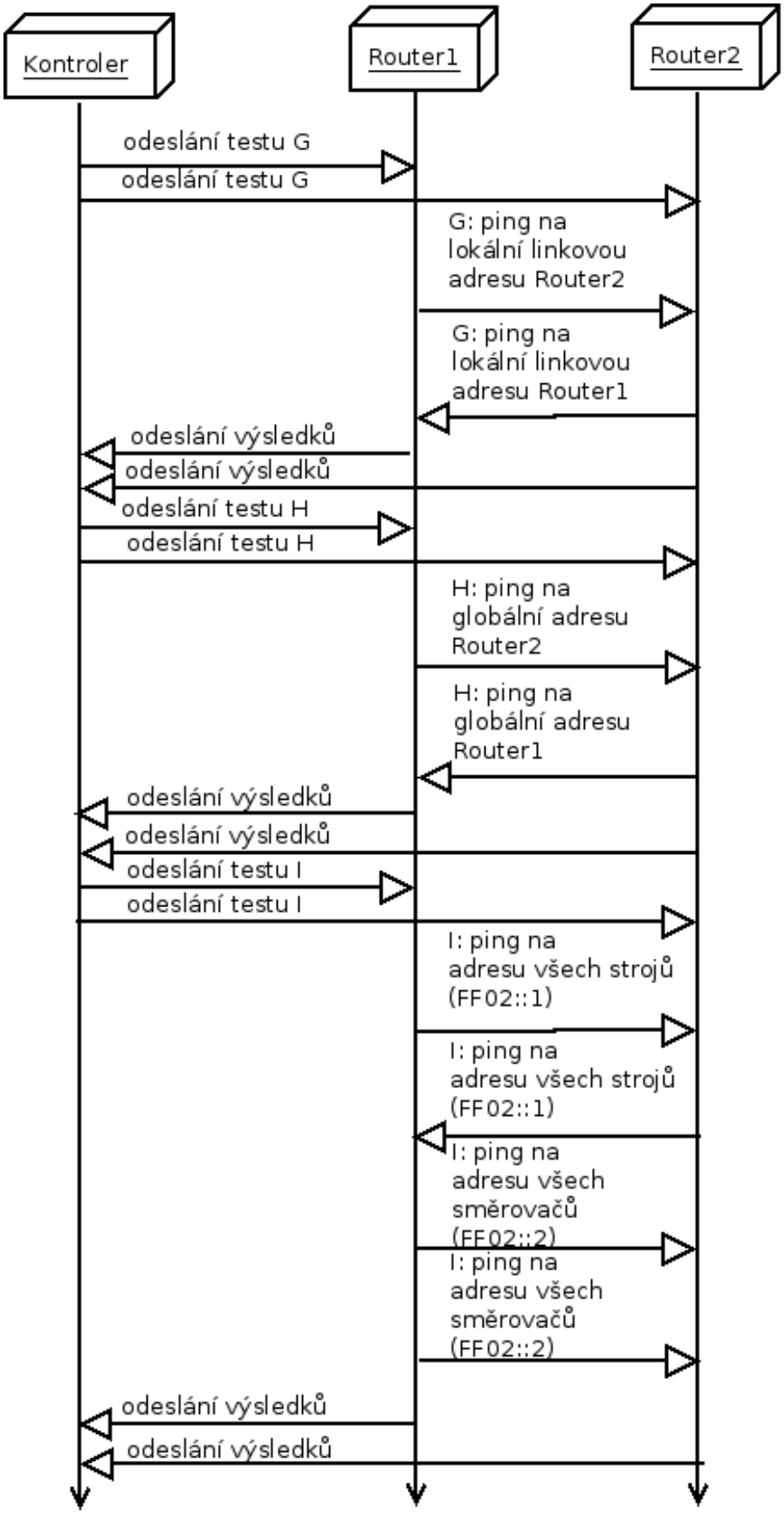
Při odeslání datagramu se vždy projde směrovací tabulka a vyberou se záznamy odpovídající cílové adrese. Vzhledem k tomu, že jich může být více, vybere se ten, jehož síťová maska je nejdelší shodná s adresou cíle datagramu.

[15]Každá síť je připojena do dalších sítí pomocí jednoho nebo více hraničních směrovačů. Každé rozhraní takového směrovače je přiřazeno do jiné logické sítě a skrze ně jsou mezi těmito sítěmi předávana data.

[16]Nelze použít výraz zařízení, protože na jednom zařízení může být několik rozhraní, například pro každou logickou síť.



Obr. 2.3: Znázornění průběhu testů části d-f



Obr. 2.4: Znázornění průběhu testů části g-i

2.2. Síťové rohraní - Stack

Síťový stack slouží k připojení počítače do sítě. Rozlišuje se pro jaký protokol je určeno. Dnes se nejběžněji používá síťový stack pro protokol *IPv4* a *IPv6*. Protože jsou tyto protokoly podobné, je možné použít takzvaně duální (hybridní) stack, který umožňuje přístup k oběma protokolům zároveň. Struktura síťového stacku odpovídá struktuře protokolu pro který je určen, tj. je rozdělen do vrstev a komunikují spolu vždy pouze 3 nejbližší vrstvy (střední vrstva využívá služeb nižší a poskytuje své služby vrstvě vyšší). V operačním systému obvykle bývá rozdělen na dvě části, nižší část (ovladače) komunikuje s hardware počítače a posílá připravená data do sítě. Vyšší část poskytuje *API*[17] pro přístup k síti uživatelským aplikacím. To umožňuje využívat jednotné rozhraní všem programům nainstalovaným v operačním systému. Takové řešení má spoustu výhod:

- Jednodušší práce pro autory aplikací. Není nutné, aby každá aplikace měla vlastní stack.
- Jednotné rozhraní umožňuje jednodušší správu, v případě nalezení chyby nebo přidání nových vlastností je možné u Unixových operačních systémů aktualizovat na jednom místě.
- Je možné dosáhnout vyššího zabezpečení. Operační systém může určovat, která aplikace bude mít přístup k síti a v jakém rozsahu. Například uživatelská aplikace může jen číst/zapisovat data, ale aplikace běžící s právy root[18], může sledovat i provoz dalších aplikací a poskytovat administrátorovi detailní informace.

Během vývoje počítačových sítí vznikly různé síťové stacky, dnes běžně potkáváme dva. Základním je *Berkeley sockets*, který je implementován ve všech Unixových operačních systémech včetně Linuxu. Z *Berkeley sockets*2.2.1 byl později odvozen *Winsocks* používaný u MS Windows.

2.2.1. Rozhraní *Berkeley sockets*

Existují dva základní druhy socketu, které se liší typem datového provozu, pro který jsou používány. Dříve byl častěji používán proudový (*stream socket)*. Používá se k sestavení spojení mezi dvěma stroji a souvislou komunikaci. To umožňuje obousměrnou

[17] *Aplication programing interface* je jednotné rozhraní, díky kterému mohou aplikace využívat vlastnosti operačního systému. Bývá navrženo jako knihovna, kterou autor aplikace může přidat do svého programu a začít využívat její funkce. Tato knihovna může být do programu přidána hned při kompilaci (statická knihovna), což je přístup používaný u MS Windows. Druhou možností je načtení při startu programu (dynamická knihovna), tento přístup používají operační systémy založené na Unixu.

[18] Nejvyšší oprávnění, která může uživatel Linuxového/Unixového systému mít, obdoba administrátora u MS Windows.

komunikaci. Včetně kontroly správnosti přijatých dat a možnosti požádat o nové zaslání v případě chyby. Druhou možností je datagramový socket. Při jeho použití se sestaví pouze balík dat (datagram) a odešle se. Narozdíl od *stream socketu*, se neprovádí kontrola odeslaných a přijatých dat. Dříve bylo jeho použití velmi omezené (*DNS* dotazy nebo kontrola času pomocí *NTP*). V současné době se jeho význam zvyšuje kvůli stoupající popularitě multimediálního obsahu na internetu a použití internetové telefonie. Zde je důležitý pravidelný proud dat a občasnou chybu, při špatném přenosu, lidský mozek nepostřehne. Na druhou stranu, zastavení přehrávání nebo výpadek telefonního hovoru, lidský mozek vnímá. Proto je raději kapacita linky použita k přenosu užitečných dat (i když budou chybná), než k přenosu dat kontrolních.

Spojová komunikace probíhá u klienta v šesti krocích, které jsou vidět v Tab. 2.14. V praxi se kroky tři až pět mohou provádět opakovaně. Po vytvoření socketu je možné používat běžné funkce jako u normálního souboru, tj. *read()*, *write* pro čtení a zápis dat z/do socketu.

Tab. 2.14: Použití technologie *stream socket* na straně klienta.

1.	Vytvoření socketu
2.	Případné přiřazení názvu (např. čísla portu a internetové adresy)
3.	Připojení k serveru
4.	Přenos dat
5.	Ukončení spojení se serverem
6.	Uvolnění (zrušení) socketu

Na straně serveru je situace trochu složitější. Postup je vidět v Tab. 2.15. Narozdíl od klienta server musí počítat s větším počtem připojení. To vede k tomu, že server musí mít frontu čekajících klientů. Protože tato fronta nemůže mít nekonečnou délku, může se stát, že pokus o spojení bude v případě jejího zaplnění zamítnut. Existuje několik řešení tohoto problému. Nejčistší řešení je obsluha všech klientů v jediném vlákně, společně s čekáním na další klienty. Toto řešení je velmi obtížné na korektní naprogramování, protože v linuxu chybí univerzální čekání na události. Dříve se velmi často používalo řešení pomocí samostatných procesů[19], které je velmi nevýhodné kvůli vysoké režii a paměťovým nárokům. V současné době se nejčastěji používá řešení pomocí vláken[20], toto je rychlé a úsporné, ale vyžaduje pečlivé zamykání sdílených[21] prostředků systémů.

Detailní informace k této problematice lze najít v knize *Jádro systému Linux*[2], v kapitole deset.

[19]Každý proces má vlastní adresový prostor a nevidí data ostatních procesů. Vzniká pomocí systémového volání *fork*.

[20]Jeden proces se rozdělí do více paralelně běžících vláken. Tato sdílejí jednotný adresový prostor.

[21]Sdílené prostředky jsou takové, které využívá více vláken. Pro zajištění exkluzivního přístupu se používá zamykání, která zabrání dalšímu vláknu v přístupu.

Tab. 2.15: Použití technologie *stream socket* na straně serveru.

1.	Vytvoření socketu
2.	Pojmenování
3.	Konfigurace socketu do naslouchacího režimu.
4.	Vyčkávání na připojení klienta.
5.	Obsloužení klientských komunikačních požadavků.
6.	Ukončení spojení.
7.	Uvolnění (zrušení) socketu

2.2.2. Rozhraní *Winsocks*

Počátky přístupu k sítím v raných verzích Windows byly ve znamení externích firem, takzvaných *3rd part*[22] výrobců. Důvodem této situace bylo podcenění situace a významu počítačových sítí ze strany výrobce operačního systému, společnosti Microsoft. Tato síťová rozhraní ostatních výrobců poskytovala navzájem nekompatibilní *API* pro uživatelské aplikace. Tato řešení byla velmi drahá, protože každému výrobci se musely zaplatit náklady na výzkum a vývoj (*R&D cost - Research and Development cost*).

Rozhraní *winsocks* vzniklo jako sjednocující nad různými implementacemi síťových stacků. Je založeno nad *Unixovým* řešením *Berkeley Socket*, popsaným v kapitole 2.2.1. První verzí MS Windows byly Windows NT 3.1 vydané v polovině roku 1993. Detailní pojednání o historii lze najit na webu [5] a elektronických přílohách této práce. Informace o použití, vývoji a referenční manuál lze najít na webu Microsoft MSDN[6].

[22] *3rd part* znamená třetího dodavatele programů (v tomto případě knihovny pro přístup k síti), k původním dvěma členům obchodního vztahu (zákazník vs. dodavatel operačního systému).

2.3. Testovací laboratoře

Ve chvíli, kdy testujeme nové vlastnosti nějakého programu, je vhodné provádět to ve specializovaném prostředí. Použití produkčních prostředků je krajně nevhodné, protože v případě závažného problému může způsobit finanční ztráty (nedostupnost serveru, náhodné odpojení uživatelů, ztráta dat). V případě použití testovacího prostředí máme k dispozici absolutní volnost co se týče konfigurace strojů, zatížení sítě a času pro testování (není nutné se ohlížet, zda jsou klienti připojeni k serveru a jak velkou část síťové linky můžeme obsadit).

Protože laboratoře *BrnoLab* a *CertLab* jsou interní prostředky firmy *Red Hat*, nemohu uvést detaily o jejich možnostech ani je použít. Uvedu proto v následujících kapitolách další možnosti testování včetně obecných informací o *IPV6 Ready* laboratořích.

2.3.1. Virtualizované prostředí

Při tvorbě virtuálního testovacího prostředí se v prvé řadě musíme rozhodnout, jaký výkon, vlastnosti a uživatelský komfort vyžadujeme. V dnešní době jsou běžně dostupné procesory s podporou virtualizace přímo v hardware. V takovém případě ji označujeme jako nativní nebo též úplnou. Druhou možností je paravirtualizace. V tomto případě virtualizovaný stroj (quest) nemá přímý přístup k hardware počítače a pro jeho běh je třeba speciální jádro. Projekt paravirtualizace se v Linuxu nazývá *Xen*[17] a v této práci se jím nebudu dále zabývat. Jednou z možností pro nativní virtualizaci je projekt *Virtualbox*, dostupný z domovské stránky projektu[18].

Při pokusu o tvorbu testovacího prostředí pro praktickou část mé práce, jsem se pokusil použít virtualizaci pomocí *QEMU/KVM*[19]. Jedná se o dva projekty umožňující oba druhy (úplnou i paravirtualizaci) virtualizace. *Qemu* je emulátor hardware počítače, který pomocí *KVM* získá přístup k hardware prostředkům počítače, na kterém běží. Pro použití *KVM* je třeba mít procesor s instrukční sadou[23], která ji podporuje. Celá technologie nativní virtualizace se u Intelu nazývá *VT-X* a u AMD *AMD-V*.

2.3.2. Instalace virtuálních strojů

Pro instalaci a administraci virtuálních strojů máme dvě možnosti. První je použití grafické aplikace *virt-manager* nebo nástrojů pro příkazovou řádku. Druhá možnost je praktičtější, protože tvorba strojů je automatizovatelná pomocí skriptů. Ve spojení s

[23]Nejběžněji se setkáme s virtualizací na klasických stolních pc arhitektury *X86*, nebo *X86_64*. Jedná se o starší třicetidvou bitovou architekturu a šedesátičtyř bitovou architekturu vyvinutou firmou *Intel* a později licencovanou dalším výrobcům. U procesorů značky *Intel* se instrukční sada pro úplnou virtualizaci nazývá *vmx* u *AMD* se nazývá *svm*.

technologií *kickstart*[24] lze dosáhnout plně automatické instalace i pro velmi rozsáhlé sítě. Pro usnadnění tvorby virtuálních strojů a sítí jsem vytvořil jednoduchý skript. N věm lze nastavit parametry guest stroje. Skript je možné nalézt v elektronických přílohách této práce ve složce bin pod názvem *make_machine.sh*. Nastavení testovací sítě, které bude použito v této práci je možné najít v adresáři bin v souboru *testnet.xml*.

Pro účely této práce jsem se pokusil vytvořit dva virtuální stroje s názvy Test1 a Test2. První má nainstalovánu Fedoru 14 pouze po *init 3*[25], druhý je nainstalován včetně grafického rozhraní. Detailní informace k vytváření virtuálních strojů jsou mimo rozsah této práce. Podrobnější informace lze najít v manuálových stránkách nástrojů *virt-install, virsh*, které jsou součástí elektronických příloh této práce, nebo na webu [19].

Protože jsem pro virtualizaci použil stroj s nainstalovaným OS Fedora 14, narazil jsem na závažný problém s podporou *IPV6* ve virtualizační knihovně *libvirt*[26]. Je to dáno tím, že ve Fedoře 14 je použita pouze verze 0.8.3. Díky tomu není možné Fedoru 14 použít jako základní operační systém pro virtualizaci v tomto konkrétním nasazení. Problémem je nekorektní konfigurace síťových rozhraní virtuálních strojů při použití *IPV6*. Kompletní podpora *IPV6* byla totiž přidána do verze 0.8.7. Detailní informace o konfiguraci lze najít na webu[25]. Informace o chybě jsou v Bugzille firmy Red Hat[24].

Rozhodl jsem se proto použít fyzický stroj s nainstalovanou Fedorou 14 a otestovat ji, proti implementaci síťového stacku na vzdáleném stroji s nainstalovaným operačním systémem [7].

2.3.3. Minilab s dedikovaným hardware

Sestavení vlastního testovacího minilabu může být finančně značně náročná věc, záleží na požadavcích na testovací stroje. Pro běžné testování by stačily dva stolní pc a jeden směrovač, v případě větších testovacích sítí (například pro vlan[27], nebo bonding[28]) by bylo vhodné oddělení ovládacího stroje od testovacího prostředí pomocí nastavitelného přepínače (managed switch)[29]. Návrh na strukturu takové testovací sítě je vidět na Obr. 2.11. Vlevo je zapojení pro testování klientských stanic vpravo je zapojení pro směrovače.

[24] Technologie[22] vyvinutá firmou *Red Hat* pro automatizovanou instalaci *Red Hat Enterprise Linux*. Konfigurační data lze generovat buď manuálně, nebo pomocí nástrojů (*Anaconda, Cobbler*).

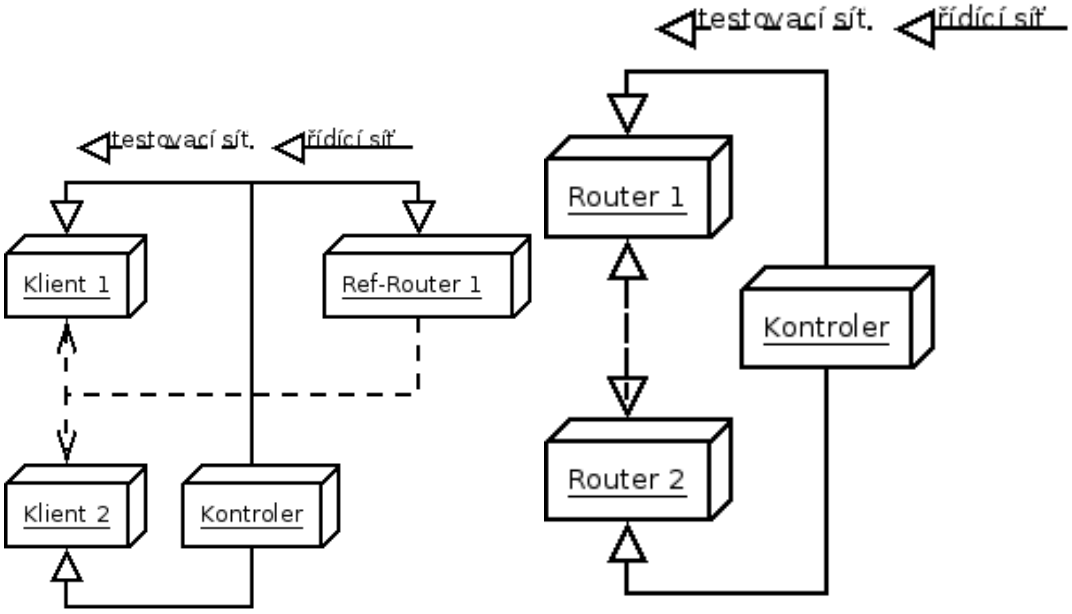
[25] Úroveň běhu linuxového systému, pod *init 3* je kompletní multiuživatelský systém v příkazovém řádku, pod *init 5* je kompletní systém včetně grafických aplikací.

[26] Základní knihovna sloužící k virtualizaci pomocí technologie *KVM*

[27] Nad jednou fyzickou sítí může být několik logických sítí

[28] Několik síťových rozhraní na jednom stroji je propojeno do jedné sítě, buď z důvodu vyšší propustnosti, nebo redundance (nefunguje-li jedno rozhraní, funguje alespoň druhé.)

[29] Přepínač, který umožňuje nastavit pro každý svůj port parametry běžně dostupné na směrovači (například firewall, zařazení do vlan sítě) a tím umožňuje vyšší optimalizaci provozu.



Obr. 2.11: Zapojení testovací sítě

2.3.4. Certifikované laboratoře *IPV6 Ready*

Na světě existuje šest certifikovaných laboratoří pro získání loga *IPV6 Ready*.

První z nich je *TIPI*[20] (testing internet protocols interoperatibility), je součástí projektu *Dionysos* (dependability, interoperability and performance analysis of networks) a spadá pod organici *IRISA*[21]. *Irisa* je francouzsko-britskou výzkumnou organizací, která zastřešuje mnoho významných Evropských výzkumných projektů. Výzkumná laboratoř *TIPI* se zabývá výzkumem nejen v oblasti *IPV6*. Jako příklad její další činnost je vytváření *ETS* (Spustitelných testů) generovaných z Abstraktních zadání (ATS) nebo ve vývoji distribuovaného testování. Další laboratoře jsou rozmístěny po celém světě, čtyři jsou v Asii a jedna v USA, odkazy na ně lze najít na webu [3] v sekci kontakty Schválené laboratoře (Approved Labs).

3. Realizované řešení

3.1. Testplán

V dnešní době vzniká stále složitější software. Nové funkce jsou přidávány stále rychleji a do starších verzí programů jsou backportovány[1] nejen tyto nové vlastnosti ale i opravy. Proto je nutné mít jednotnou metodiku jak zajistit, aby se jednou odhalené chyby nešířily do dalších verzí. Protože každý testovaný program je jiný, firma provádějící vývoj má své specifické požadavky, existuje obecná norma *IEEE 829*[2],která specifikuje co testplán může a co musí obsahovat.

3.1.1. Šablona pro tvorbu testplánu

Testplán může obsahovat mnoho položek, které v konkrétním případě nemusí mít využití, je proto nutné pro každý testovaný produkt sestavit speciální testplán. Ten bude splňovat požadavky zadavatele na kvalitu, rychlost testování a seznam vlastností, které mají být hlídány. V následujících řádcích se pokusím okomentovat anglický článek [11], který popisuje, co by testplán podle normy *IEEE 829* měl obsahovat.

1. **Název testovaného produktu, Autor, Obsah, Úvod** - Obsahuje základní informace o testovaném produktu, autorovi a struktuře dokumentu.

2. **Cíle a Úkoly** - Popisuje cíle testování, zodpovědnost jednotlivých týmů, definuje, zda dokument může být použit jako servisní garance. Mezi úkoly můžeme zařadit například jednotlivé fáze testování (hlášení chyb nebo doplňkové testování po vydání).

3. **Účel** - obsahuje detailní popis všech funkcí, zda budou, nebo nebudou testovány, říká jestli už existují nějaké dílčí testovací postupy. Zároveň je dobré definovat způsoby jak kontaktovat lidi zodpovědné za jednotlivé funkce testovaného programu, nebo knihovny.

[1] Postup kdy jsou do starší verze programu přidány nové vlastnosti, nebo opravy chyb, které by jinak byly dostupné jen pro novou verzi. Tato technika se používá velmi omezeně, mimo programů s placenou podporou, protože je pro vývojáře a testery časově velmi náročná. Jako příklad volně dostupných backportovaných aplikací lze uvést repozitář (server sloužící k instalaci a aktualizaci programů používaný v GNU/Linuxu[10]) *Debian Backports*[8], který přidává do stabilní verze Debianu[7] nové vlastnosti a dostupné jinak jen v testovací verzi (například nové verze kancelářského balíku nebo multimediálního přehrávače). Komerční podporu pro udržování starších verzí programů nabízí například společnost Red Hat[9] a jejich distribuce *Redhat Enterprise Linux - RHEL*.

[2] Narozdíl od norem *RFC* vydávaných *IETF*[4], nejsou normy *IEEE* veřejně přístupné (bez registrace a platby), proto jsem ji nezahrnul do příloh, ale během tvorby této práce jsem ji měl k nahlédnutí ve firmě Red Hat. Pro vytvoření detailnější představy o obsahu lze použít článek[11]

4. **Rozdělení testů**

   4. 1 **Požadavky pro vydání** - V jednotlivých fázích vývoje[3] je nutné definovat, jak dlouho bude vývoj trvat, nebo testování funkcionality v dané fázi do produktu přidané.

   4. 2 **Testování systémové integrace** - Sekce udává jak bude probíhat spolupráce s dalšími komponentami prostředí, ve kterém produkt poběží. Další věcí je, kdo napíše skripty[4] pro jednotlivé části testů.

   4. 3 **Testy výkonu a stability** - V této části je rozbor takzvaných zátěžových testů. Udává, jak který projekt definuje výkon a jakými prostředky ho otestovat.

   4. 4 **Regresní testování** - Slouží k odhalení, zda se v programu znova nevyskytují už jednou opravené chyby.

5. **Požadavky na testovací hardware a prostředí** - Udává jaké vybavení je třeba pro otestování produktu (software i hardware).

6. **Časový rozvrh testování** - Udává časovou náročnost testování. Obsahem kapitoly je obsazení hardware prostředků, software licencí (v případě že je jich zakoupen pouze omezený počet) a zaměstanců.

7. **Kontrolní mechanismy** - Udává jakým způsobem bude probíhat hlášení chyb, ať už automaticky nebo manuálně. Též se zabývá požadavky na změny ve vlastnostech programu, definuje kdo má právo tyto změny navrhovat a schvalovat.

8. **Vlastnosti ne/zahrnuté do testování** - Udává seznam jednotlivých částí programu s ohledem na nutnost jejich testování.

[3] *Alpha, Beta, Nightbuild* Termíny udávající v jaké fázi vývoje se daný program nachází. *Alpha* říká, že se jedná o náhled nově vyvíjených vlastností, předpokládá se výskyt kritických chyb. Verze je nevhodná pro běžné uživatele. Ti nejsou schopni se s výskytem chyby vyrovnat, nahlásit ji vývojářům a případně provést obnovení poškozených dat ze zálohy. U *Beta* verze by měl být ukončen vývoj nových vlastností a pouze se opravují nahlášené chyby. Termín *nightbuild* udává aktuální verzi programu, která neprošla žádným testováním. Může tudíž obsahovat jakoukoliv chybu, protože většinou vzniká automaticky z kódu ten den přidaného do vývojového repozitáře (Server sloužící k instalaci programů operační systém Linux.).

[4] Skript je program, napsaný v interpretovaném jazyce. To znamená, že pro jeho běh není třeba překladač a skript vykonává interpreter příkaz za příkazem z textového souboru. Z toho vyplývá, že v případě větvení kódu program může pokračovat, a když je v jedné větvi chyba, interpreter ji najde, až na chybný příkaz narazí. U některých programovacích jazyků je možné skript zkompilovat a spouštět ho i bez přítomnosti interpretu daného jazyka. V linuxu se nejběžněji vyskytuje skriptovací jazyk *Bash*[12], příkladem programovacího jazyka s možností kompilace je *Python*[13]

9. **Požadavky na lidské a finanční zdroje** - Udává, kteří zaměstanci se budou věnovat kterým částem testování. Kdo je zodpovědný za plánování, vývoj, spouštění a vyhodnocování testů. Kdo zajistí testovací prostředí.

10. **Dokumenty** - Udává které ze čtyř možných dokumentů budou vytvořeny (Testplán, Testovací příklady, Seznam problémů při testování, Seznam výsledků testování).

11. **Závislosti** - Udávají na kterých skutečnostech testování závisí (termín dokončení, dostupnost testovacího prostředí).

12. **Předpokládané problémy** (*Risks assumptions*) Udává s jakými problémy se během testování počítá, například jak se bude řešit zpoždění ve vývoji testů.

13. **Nástroje** - Udává seznam nutných použitých nástrojů použitých pro testování, včetně bugzilly[5].

14. **Schvalovací proces** - Udává seznam osob, zodpovědných za schválení a případné revize testplánu.

3.1.2. Návrh Tesplánu

V této kapitole se pokusím navrhnout testplán pro *IPV6 stack* použitý v kernelu Linuxové distribuce *Fedora 14* [15]. Od oficiálního *Vanilla Kernelu*[14] vyvíjeného Linusem Torvaldsem se liší v některých komunitně zařazených patch[6] souborech.

1. **Název testovaného produktu** - GNU/Linux Fedora 14

2. **Autor** - Jan Malaník

3. **Úvod** - Tento testplán slouží pro testování kernelu linuxové distribuce Fedora 14 z pohledu základní funkčnosti, výkonu a hledání regresí.

4. **Cíle a úkoly** - Testplán je primárně určen pro *QA* vývojové oddělení.

5. **Rozdělení testů** - Testy by měly být provedeny pro všechny architektury, pro které je Fedodra 14 dostupná. V současné době je to *i686* a *x86_64*.

   5. 1 **Ověření základní funkčnosti** - K tomuto účelu bude použit Ping test z kapitoly 2.1.2 založený na doporučení pro testování *IPV6 Ready Fáze 1*. Test bude

[5] Nástroj pro hlášení chyb v produktu, sledování jejich stavu, přidělování vývojářům a členům *QA* týmu. Samozřejmě včetně záznamu o tom jakým způsobem byla chyba uzavřena (například není chyba, bude opraveno v příští verzi, bude opraveno).

[6] Patch je soubor obsahující nějakou změnu v programu, například slouží k doplnění nových vlastností produktu, případně opravě chyb.

založen na programu *Nettest* vyvíjeného firmou Red Hat[7] pro účely automatického síťového testování.

5. 2 **Testy bezpečnosti** - Ověření základní funkčnosti *ip6tables*[8].

5. 3 **Výkonové testy** - Budou založeny na programu *Netperf* [16].

5. 4 **Regresní testy** - Mohou být implementovány v libovolném programovacím jazyce, jejich jediným výstupem je jedna nebo nula pro úspěch a neúspěch. Pro účely ladění mohou mít možnost zapnutí debug výpisů[9].

6. **Požadavky na testovací hardware a prostředí** - Pro účely vývoje testů a během tvorby Fedory 14 postačí testování ve *virtuální prostředí*[10]. Důležitou podmínkou pro volbu virtualizačního prostředí je možnost nastavení počtu síťových karet (jsou nutné dvě, jedna pro testovací síť, druhá pro ovládání testovacích strojů). Do virtuálních strojů bude nainstalována Fedora 14 v konfiguraci pro uživatelskou stanici a pro server (směrovač). Pro tuto konfiguraci není nutné instalovat grafické aplikace. Při závěrečném testování je nutné použití reálného hardware. Je vhodné použití specializovaného minilabu obsahujícím několik strojů (pro základní testování postačuje jeden směrovač a dva klienti). Každý stroj bude opět obsahovat 2 síťové karty, jednu pro ovládání druhou pro testování. V případě problémů s virtualizací, nebo nedostupnosti minilabu, je možné testování na jednom fyzickém stroji, oproti jinému stroji s libovolnou implementací *IPV6* stacku.

7. **Vlastnosti ne/zahrnuté do testování** - Nebudou testovány klientské aplikace (Skype, Icq, Firefox). Testován však bude kernel se zaměřením na základní funkčnost sítě. V pozdějších verzích bude doplněna řada dalších testů například pro kontrolu bezpečnostních prvků (*ip6tables*), regresní testy, výkonové a zátěžové testy, pro ověření stability

8. **Dokumenty** - Jako výstup mé práce vznikne tento Tesplán.

9. **Nástroje**

[7] V době psaní této práce (duben - květen 2011) program prochází interním schvalovacím řízením pro zveřejnění pod licencí GPL. Proto zatím není k dispozici veřejná adresa pro stažení, ale v elektronických přílohách této práce, lze najít snapshot zdrojových kódů.

[8] Základní nástroj pro nastavování pravidel firewallu pro Linux.

[9] Ladící výpisy slouží pro zobrazení detailních informací o běhu programu pro účely testování a hledání chyb v samotném programu.

[10] Při běhu ve virtuálním prostředí se na fyzický počítač s hardware podporou virtualizace nainstaluje libovolný GNU/Linux s funkční *IPV6* implementací. Tomuto stroji se říká *host* (anglický termín pro výraz hostitel). Každému vytvořenému virtuálnímu počítači se pak říká *quest*. Takových virtuálních počítačů může být vytvořeno libovolné množství, počet spuštěných záleží na výkonu *host* počítače (velikost RAM, počet procesorů).

9. 1 **Ping6** - Je instalován jako součást distribuce. Slouží k ověření dostupnosti síťového stroje a zjištění času nutného pro jeho dosažení skrz síť.

9. 2 **Ip6tables** - Je instalován jako součást distribuce. Slouží k nastavování firewallu.

9. 3 **Nettest** - V době psaní této práce ještě nemá veřejné www stránky. Bude použit jako framework pro tvorbu základního testu. Jakou součást vzniku tohoto testplánu, bude nutné vytvořit modul podporující ping pomocí protokolu *IPV6* a napsat xml soubor popisující průběh *Ping testu*, založeného na certifikaci *IPV6 Ready*.

9. 4 **Netperf** - Je dostupný ze stránek [16]. Poslouží pro výkonové a zátěžové testy stability.

9. 5 **Python** - Je dostupný ze stránek [13]. Je programovací jazyk použitý při tvorbě programu *Nettest*.

9. 6 **Ip** - Je instalován jako součást distribuce v balíčku *iproute2*. Slouží pro nastavování síťových adres a směrovacích cest, při manuální konfiguraci sítě.

10. **Schvalovací proces** - Osoby zodpovědné za ověření testplánu:

Na základě tohoto testplánu upravím modul pro program *Nettest*, aby podporoval protokol *IPV6*, napíšu *xml* soubor, popisující průběh testů a provedu testování síťového stacku Fedory 14.

3.2. Test pro stack Fedory 14

Pro účely testování stacku Fedory 14, jsem využil základní test ze specifikace *IPV6 Ready* upravený pro mé potřeby. Jeho kompletní struktura je vidět na Obr. 2.2, 2.3, 2.4. Pro účely otestování klientské stanice jsem použil pouze části A, B. Pro implementaci jsem si vybral aplikaci *Nettest* vyvíjenou týmem kolem Jiřího Pírka. Je napsána jako framework[11] pro rozsáhlé síťové testování. Umožňuje testovat inicializaci síťových zařízení, generování provozu a jeho parsování. Protože je aplikace zatím ve vývoji, nemá domovskou stránku a do elektronických příloh mé práce přikládám snapshot[12] zdrojových kódů.

3.2.1. Program *Nettest*

Program *Nettest* je složen z několika částí. Základem je kontroler, který spouští pokyny zadané formou xml[13] souboru. Kontroler zpracuje zadání úkolu, dle specifikace se připojí na stroje použité k testování, spustí na nich program *nettestslave*, ktery poslouchá kontroler pomocí *RPC*[14] a provádí jeho pokyny. Po skončení příkazu odešle výsledek a podle nastavení může skončit. *Nettestslave* je možné spustit na daném testovacím stroji i manuálně, ale pak je potřeba dát si pozor na dostatečná oprávnění procesu, protože ikdyž nettest používá přihlášení přes *root*[15], některé programy nemusí s právy běžného uživatele správně běžet, například *tcpdump*[16]. Také je třeba si dát pozor na nastavení firewall na testovacích strojích, aby povoloval *RPC* spojení.

V Př. 3.1 je vidět xml soubor specifikující průběh testování. Jeho obsahem je v části "machines" specifikace jednotlivých testovacích strojů, včetně možnosti nastavení rozhraní. To bohužel není možné, protože pro testování jsem použil reálné stroje (z toho jeden vzdálený s referenční distribucí Debian Squezze), s jednou síťovou kartou, takže v případě chybné konfigurace by nebylo možné se připojit na stroj a provést opravu. V další části je třikrát zavolán modul *IcmpPing*, kterému jsou předány parametry cílová adresa pro ping, verze protokolu, která se má použít a počet odeslaných datagramů. Jako cílová adresa je postupně použita globální adresa vzdáleného stroje, lokálního stroje a lokální linková adresa stroje, který je defaultní branou pro testovaný stroj s Fedorou 14. Důležitým parametrem je hodnota *limit_rate* udávající nutnou úspěšnost test, pro splnění v procentech. Pokud se všechny tři příkazy provedou, je možné test prohlásit za úspěšný.

[11] Vývojové *API* v programovacím jazyce, které obsahuje základní, běžně používané funkce.

[12] Aktuální stav zdrojových kódů k určitému dni.

[13] *Extensible Markup Language* je programovací jazyk založený na textu formátovaném do tagů (značek), je určen pro strojové zpracování.

[14] *Remote procedure call* slouží ke spouštění vzdálených procesů.

[15] Uživatel s nejvyššími oprávněními v linuxu.

[16] Program sloužící ke sledování síťového provozu, podle zadaných kritérií. Domovská stránka je na webu [23].

Př. 3.1: Provedení prvních tří částí ping testu, pomocí *xml* specifikace úlohy pro program *Nettest*

```
1  <nettestrecipe>
2    <machines>
3    <!-- V casti machines se speficikuji ip adresy pro testovaci stroje, vcetne heslo uzivatele root, zde by take bylo
         mozne provest nastaveni sitovych rozhrani-->
4      <machine id="1">
5        <netmachineconfig>
6          <info hostname="127.0.0.1" rootpass="xyz"/>
7        </netmachineconfig>
8        <netconfig>
9        </netconfig>
10     </machine>
11     <machine id="2">
12       <netmachineconfig>
13         <info hostname="malanik.org" rootpass="zyx"/>
14       </netmachineconfig>
15       <netconfig>
16       </netconfig>
17     </machine>
18   </machines>
19   <command_sequence>
20   <!-- V teto casti kazdy prikaz znamena provedeni jedne casti Ping testu. Vola se test s nazvem IcmpPing a predavaji
         se mu parametry ktere budou vysvetleny mimo komentar ke zdrojovemu kodu -->
21     <command machine_id="1" type="test" value="IcmpPing" >
22       <options>
23         <option name="addr_family" value="6"/>
24         <option name="addr" value="2002:d450:4a62:1::1" />
25         <option name="count" value="1"/>
26         <option name="limit_rate" value="100"/>
27       </options>
28       </command>
29     <command machine_id="2" type="test" value="IcmpPing" >
30       <options>
31         <option name="addr_family" value="6"/>
32         <option name="addr" value="2001:67c:1220:c1b1:21d:9ff:fe50:171c" />
33         <option name="count" value="1"/>
34         <option name="limit_rate" value="100"/>
35       </options>
36     </command>
37     <command machine_id="1" type="test" value="IcmpPing" >
38       <options>
39         <option name="iface" value="eth0"/>
40         <option name="addr_family" value="6"/>
41         <option name="addr" value="fe80::21b:3fff:fe56:8a00" />
42         <option name="count" value="1"/>
43         <option name="limit_rate" value="100"/>
44       </options>
45     </command>
46   </command_sequence>
47 </nettestrecipe>
```

3.2.2. Implementace *Ping testu*

Implementace *Ping testu* je založena na příkladu testovacího modulu aplikace *Nettest* a upravena pro podporu *ping6*. Zdrojovy kód je vidět v Př. 3.2. Zajímavou částí je funkce

_compose_cmd(), kde jsem přidal rozpoznání typu protokolu pomocí parametru *family* a přiřazení odesílacího zařízení, pokud je použito. V hlavní funkci se provede zavolání sestaveného příkazu, rozparsování jeho výstupu pomocí regulárních výrazů a výpočet úspěšnosti testu.

Př. 3.2: Implementace *Ping6 testu*

```
__author__ = """
jpirko@redhat.com (Jiri Pirko)
jmalanik@redhat.com (Jan Malanik)
"""

import logging
import re
from Common.TestsCommon import TestGeneric
from Common.ExecCmd import exec_cmd

class TestIcmpPing(TestGeneric):
    def _compose_cmd(self)                                  #V teto funkci dochazi k sestaveni prikazu ping,
        addr = self.get_mopt("addr", opt_type="addr")       # oproti puvodni implementaci jsem pridal
        family= self.get_mopt("addr_family", opt_type="6" ) #parametr "addr_family", ktery urcuje
        if family == "6":                                   # v jakem protokolu chceme provest ping a podle toho
          cmd = "ping6 %s" % addr            # zvolime program (ping, ping6)
        elif family =="4":
          cmd = "ping %s" % addr
        else:
          return self. set_fail ("unknown ip family version")
        iface = self.get_opt("iface" )
        if family == "6" and iface:
            cmd += " -I %s" % iface
        else:
            cmd += ""
        count = self.get_opt("count")
        if count:
          cmd += " -c %s" % count
        interval = self.get_opt("interval")
        if interval:
            cmd += " -i %s" % interval
        return cmd

    def run(self):                              #V teto funkci dojde ke spusteni ping prikazu
        cmd = self._compose_cmd()
        print cmd
        limit_rate = self.get_opt("limit_rate", default=80)
        data_stdout = exec_cmd(cmd, die_on_err=False)[0]
        """
Provede se parsovani vystupu pomoci regularnich vyrazu . V pripade ze hledane retezce nebubdou nalezeny, test skonci s
      chybou, znamena to, ze vystup programu ma neocekavanou vystupni syntaxi.
"""
        stat_pttr1 = r'(\d+) packets transmitted, (\d+) received'
        stat_pttr2 = r'rtt min/avg/max/mdev = (\d+\.\d+)/(\d+\.\d+)/(\d+\.\d+)/(\d+\.\d+) ms'
        match = re.search(stat_pttr1, data_stdout)
       if not match:
            return self. set_fail ("expected pattern not found")

        trans_pkts, recv_pkts = match.groups()
        rate = int(round((float(recv_pkts) / float(trans_pkts)) * 100))
        logging.debug("Transmitted \"%s\", received \"%s\","
```

```
51                      "rate␣\"%d%%\",␣limit_rate␣\"%d%%\""
52                              % (trans_pkts, recv_pkts, rate, limit_rate))

54          match = re.search(stat_pttr2, data_stdout)

56          if match:                       #V pripade shody se provede parsovani vystupnich hodnot
57              tmin, tavg, tmax, tmdev = [float(x) for x in match.groups()]
58              logging.debug("rtt␣min␣\"%.3f\",␣avg␣\"%.3f\",␣max␣\"%.3f\",␣"
59                      "mdev␣\"%.3f\"" % (tmin, tavg, tmax, tmdev))

61          if rate < limit_rate:               #Provede se vypocet uspesnosti pingu a podle toho test vrati ne/uspech
62              return self.set_fail("rate␣is␣lower␣that␣limit")

64          return self.set_pass()
```

3.2.3. Použití testu na stack Fedory 14

Nyní mám připraven modul *IcmpPing*, s podporou protokolu *IPV6* a *xml* soubor udávající průběh testu. Z kořenového adresáře programu nettest mohu proto nyní provést spuštění testu pomocí *./nettestctl.py -r ping.xml -e run*. Pomocí přepínače *-r ping.xml* určím jaký *xml* soubor má být spuštěn. Přepínač *-e* udává, že na testovacích strojích si kontroler sám má spustit program obsluhující testy. Na závěr parametr *run* říká, že test chci spustit. Další možné parametry lze získat při spuštění bez parametrů. V příkladu 3.3 je vidět průběh testu.

Př. 3.3: Výstup programu *Nettest* při spuštění *Ping6 testu*

```
1  29/05 18:49:26|      (127.0.0.1) NetTestControll:0044| INFO: Remote app exec on machine 127.0.0.1
2  29/05 18:49:36|      (127.0.0.1)       SshUtils:0368| INFO: Copy to remote machine (127.0.0.1) pass.
3  29/05 18:49:40|      (127.0.0.1)       SshUtils:0153| INFO: Got shell prompt -- logged in
4  29/05 18:49:45|      (127.0.0.1)       SshUtils:0153| INFO: Got shell prompt -- logged in
5  29/05 18:49:45|      (127.0.0.1) NetTestControll:0044| INFO: Remote app exec on machine malanik.org
6  29/05 18:49:56|      (127.0.0.1)       SshUtils:0368| INFO: Copy to remote machine (malanik.org) pass.
7  29/05 18:50:01|      (127.0.0.1)       SshUtils:0153| INFO: Got shell prompt -- logged in
8  29/05 18:50:07|      (127.0.0.1)       SshUtils:0153| INFO: Got shell prompt -- logged in
9  29/05 18:50:07|      (127.0.0.1) NetTestControll:0066| INFO: Connecting to RPC on machine "127.0.0.1"
10 29/05 18:50:07|      (127.0.0.1) NetTestControll:0066| INFO: Connecting to RPC on machine "malanik.org"
11 29/05 18:50:07|      (127.0.0.1) NetTestControll:0083| INFO: Setting logging server on machine "127.0.0.1"
12 29/05 18:50:07|      (127.0.0.1) NetTestControll:0083| INFO: Setting logging server on machine "malanik.org"
13 29/05 18:50:07|      (127.0.0.1) NetTestControll:0092| INFO: Setting netconfigs on machine "127.0.0.1"
14 29/05 18:50:07|      (127.0.0.1) NetTestControll:0092| INFO: Setting netconfigs on machine "malanik.org"
15 29/05 18:50:08|      (127.0.0.1) NetTestControll:0158| INFO: Executing command: [type "test", machine_id "1", value "
       IcmpPing"]
16 29/05 18:50:08|      (127.0.0.1) NetTestControll:0158| INFO: Executing command: [type "test", machine_id "2", value "
       IcmpPing"]
17 29/05 18:50:08|      (127.0.0.1) NetTestControll:0158| INFO: Executing command: [type "test", machine_id "1", value "
       IcmpPing"]
18 29/05 18:50:08|      (127.0.0.1) NetTestControll:0108| INFO: Clearing netconfigs on machine "127.0.0.1"
19 29/05 18:50:08|      (127.0.0.1) NetTestControll:0108| INFO: Clearing netconfigs on machine "malanik.org"
20 29/05 18:50:08|      (127.0.0.1)       nettestctl :0063| INFO: ========== SUMMARY
       ==========
21 29/05 18:50:08|      (127.0.0.1)       nettestctl :0069| INFO: *PASS* ping.xml
```

```
22 29/05 18:50:08|        (127.0.0.1)        nettestctl :0070| INFO:
     ================================================
```

Ve výstupu programu je vidět, že došlo k připojení na oba testovací stroje (127.0.0.1 a malanik.org), přihlášení a spuštění logovacího serveru, který zaznamenává průběh testu a odesílá výsledky. Dále dojde ke spuštění jednotlivých částí testu a nahlášení jejich výsledků. Protože všechny části testu proběhly úspěšně, program vypsal *PASS*, čímž oznámil úspěch a skončil.

Protože test pro stack Fedory 14 proběhl úspěšně můžeme říct, že rozhraní pro přístup k síti pomocí protokolu *IPv6* je funkční.

4. Závěr

V této bakalářské práci jsem se v první kapitole[2.1] věnoval problematice *IPV6* protokolu, podrobně jsem se seznámil se strukturou datagramu a zjistil rozdíly oproti *IPV4* datagramu. Také jsem věnoval pozornost možnostem získání certifikace *IPV6 Ready* Tu je možno získat, v jedné z šesti laboratoří na světě. V další kapitole[2.2] jsem se zabýval průzkumem síťového stacku v operačním systému GNU/Linux, jeho historickým vývojem a oddělením síťového stacku pro operační systém *MS Windows* s *Berkeley sockets*.

Dalším bodem[2.3] mého zadání bylo seznámení s laboratořemi *BrnoLab* a *CertLab*. Protože tyto jsou majetkem firmy Red Hat a informace o nich jsou chráněny *NDA* smlouvou, vytvořil jsem kapitolu, kde jsem se zabýval obecnou problematikou testovacích laboratoří. V ní jsem popsal možnosti minilabu složeného z reálných komponent. V této variantě se cena testování odvíjí od požadavků na testovaná zařízení. Oproti tomu je mnohem jednodušší variantou virtuální testování. Dnes je možné bez problémů zajistit stolní *X86* počítač s podporou plné virtualizace, která umožňuje přímý přístup ke komponentám pc bez ztráty výkonu. Na starším hardware je možné použít paravirtualizaci a emulovat reálná zařízení za pomoci speciálně upraveného linuxového jádra. Pro potřeby své bakalářské práce jsem vytvořil jednoduchý skript, zrychlující vytváření virtuálních strojů.

Protože virtualizační knihovna (libvirt) použitá ve Fedoře 14, na stroji, který měl běžet jako virtualizační host (stroj poskytující své služby virtuálním strojům), má nedostatečnou podporu *IPV6*, nebylo možné použít připravené virtuální prostředí vytvořené pomocí *KVM*. Detaily o tomto problému lze najít v kapitole [2.3.2]. Rozhodl jsem se proto použít Fedoru 14 nainstalovanou na fyzickém stroji a otestovat ji oproti vzdálenému stroji s nainstalovaným operačním systémem Debian.

V praktické části[3.1] jsem vytvořil testplán pro linuxový síťový stack Fedory 14, dle normy *IEEE829*. Tesplán je zaměřen pouze na testování nízkoúrovňových záležitostí. Nezabývá se testováním uživatelských aplikací, jako například Mozilla Firefox, nebo přenos videa/zvuku. Vytvořil jsem pouze základní dokument. Nezabýval jsem dalšími dokumenty, které dle normy mohou vzniknout. Zde by bylo možné mou práci dále rozšířit.

Dále jsem v kapitole[3.2] implementoval test dle specifikace *IPV6 Ready* a využil jej jako podmínku pro splnění testplánu. Test je založen na úpravě modulu pro program *Nettest*. Standartní modul podporuje ping pouze pomocí protokolu *IPV4*. Po přidání podpory *IPV6* ho bylo možné jednoduše použít pro účely této práce. Specifikaci průběhu testu jsem provedl pomocí *xml* souboru, obsahujícího seznam příkazů pro program *Nettest*.

Na mou práci by bylo možno navázat v několika směrech. V případě použití virtuálních strojů, by bylo příhodné využít technologie *Kickstart* pro plně automatizovanou instalaci testovacích laboratoří v kombinaci s mým instalačním skriptem. V případě nasazení pro větší laboratoře by také bylo vhodné zjistit, jaké existují pokročilé systémy nejen pro

instalaci, ale i správu nainstalovaných strojů (přidělení uživateli jen na určitý čas, hlášení chyb, sledování vytíženosti) a zvážit nasazení takového dlouhodobě vyvíjeného programu. Hodně práce nabízí implementace dalších testů. Můj testplán lze rozšířit například o testy pro odhalení regresí, testy pro pokročilá nastavení linuxového firewallu (ip6tables) nebo o daslší testy, které jsou součástí certifikace *IPV6 Ready*, například, parsování hlavičky datagramu, autentizace a šifrování *IPSEC* nebo pro podpora mobility.

Z pohledu mého testplánu lze konstatovat, že *IPV6* stack Fedory 14, je funkční, protože test proběhl v pořádku. Jak sem zmínil výše, jsou zde problémy na úrovni systémových knihoven. Tyo knihovny nenabízí úplnou funkcionalitu poskytovanou síťovým stackem. V případě vytvoření testů, pro méně používané protokoly využívající *IPV6*, bych byl pravděpodobně schopen sestavit seznam bugů, které je nutné opravit.

5. Seznam použitých zkratek a symbolů

Stack	Kompletní sada programů a knihoven pro připojení k síti.
Interface	Síťová karta (fyzická, nebo logická) sloužící k připojení pc do sítě.
Rozhraní	Český překlad pro výrazy *Stack* a *Interface*, lze použít v obou významech.
rhel	Red Hat Enterprise Linux
API	Application Programming Interface
bit	základní jednotka v IT vyjadřuje jedničku nebo nulu
byte	osm bitů
R&D	Research and Development
MTU	Maximální velikost datagramu, který je možno přenést
QA	Quality Assurance
QE	Quality Engineering
KVM	Kernel-based Virtual Machine
NDA	Nondisclosure agreement
ETS	Executable test suite
ATS	Abstract test suite
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
RFC	Request for Comments
DHCP	Dynamic Host Configuration Protocol
DHCPv6	Dynamic Host Configuration Protocol V6
ICMP	Internet Control Message Protocol
ICMP6	Internet Control Message Protocol V6

IPV4	Internet Protocol V4
IPV6	Internet Protocol V6
NTP	Network Time Protocol
DNS	Domain Name System
BOOTP	Bootstrap Protocol (definován v *RFC 951*)
RARP	Reverse Address Resolution Protocol
ARP	Address Resolution Protocol
IPSEC	IP Security
MIPV6	Mobile Internet Protocol V6
NEMO	Network Mobility Support in *IPV6*
SIP	Session Initiation Protocol
SNMP	Simple Network Management Protocol
IMS	IP Multimedia Subsystem
MLDV2	Multicast Listener Discovery V2

Literatura

[1] SATRAPA, Pavel. IPv6 : Internetový protokol IPV6. Vyd. 2. Praha : CZ.NIC, z. s. p. o., 2008. 357 s. Dostupné z WWW: <http://knihy.nic.cz/>. ISBN 978-80-904248-0-7.

[2] JELÍNEK, Lukáš. Jádro systému Linux : Kompletní průvodce programátora. 1. Brno : Computer Press, a. s., 2008. 686 s. ISBN 978-80-251-2084-2.

[3] IPv6 Ready [online]. 2010 [cit. 2011-04-21]. Dostupné z WWW: <http://www.ipv6ready.org/>.

[4] The Internet Engineering Task Force (IETF) [online]. c2011 [cit. 2011-05-03]. Dostupné z WWW: <www.ietf.org>.

[5] YOUNG, Warren. Tangentsoft Escape Gravity : Winsock Programmer's FAQ [online]. c2009 - c2011, 28. ledna 2010 08:27 MST [cit. 2011-05-07]. The History of Winsock. Dostupné z WWW: <http://tangentsoft.net/wskfaq/articles/history.html>.

[6] Microsoft MSDN [online]. c2011, Build date: 4/28/2011 [cit. 2011-05-07]. Getting Started with Winsock. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/ms738545(v=vs.85).aspx>.

[7] Debian GNU/Linux [online]. c1997 - c2011, Last Modified: Sat, May 7 07:41:22 UTC 2011 [cit. 2011-05-07]. Dostupné z WWW: <http://www.debian.org>.

[8] Debian GNU/Linux [online]. c2011, Last edited Wed Oct 6 21:24:15 2010 [cit. 2011-05-07]. Debian Backports. Dostupné z WWW: <http://backports.debian.org/>.

[9] Red Hat : The world's Opensource Leader [online]. c 2011 [cit. 2011-05-07]. Dostupné z WWW: <http://www.redhat.com/>.

[10] PŘIBYL, Adam. GNU/Linux [online]. c 2011 [cit. 2011-05-07]. Operační systém se vším,co potřebujete. Dostupné z WWW: <http://www.linux.cz/>.

[11] Software Testing Help [online]. c2007 [cit. 2011-05-07]. Test plan sample: SoftwareTesting and Quality assurance Templates. Dostupné z WWW: <http://www.softwaretestinghelp.com/test-plan-sample-softwaretesting-and-quality-assurance-templates/>.

[12] Bash - GNU Project - Free Software Foundation (FSF) : Bash [online]. c 2009, Date: 2006/11/20 09:03:24 [cit. 2011-05-27]. Dostupné z WWW: <http://www.gnu.org/software/bash/>.

[13] Python Programing language - Official website [online]. c 2011 [cit. 2011-05-27]. Dostupné z WWW: <http://www.python.org/>.

[14] The Linux Kernel Archives [online]. June 8, 2011 [cit. 2011-05-27]. Dostupné z WWW: <http://kernel.org/>.

[15] Fedora Project Homepage [online]. c 2011 [cit. 2011-05-27]. Dostupné z WWW: <http://fedoraproject.org/>.

[16] JONES, Rick . The temporary Netperf Homepage [online]. c 2011 [cit. 2011-05-27]. Dostupné z WWW: <http://www.netperf.org/netperf/>.

[17] Welcome to xen.org, home of the Xen® hypervisor, the powerful open source industry standard for virtualization [online]. c 2011 [cit. 2011-05-27]. Dostupné z WWW: <http://www.xen.org/>.

[18] Virtualbox [online]. c 2011 [cit. 2011-05-27]. Dostupné z WWW: <http://www.virtualbox.org/>.

[19] KVM : Kernel Based Virtual Module [online]. c 2011 [cit. 2011-05-27]. Dostupné z WWW: <http://www.linux-kvm.org>.

[20] Irisa [online]. C 2008 [cit. 2011-05-27]. TIPI - Testing Internet Protocols Interoperability. Dostupné z WWW:<http://www.irisa.fr/tipi/wiki/doku.php>.

[21] Irisa (Institut de Recherche en Informatique et Systèmes Aléatoires) [online]. c 2010 [cit. 2011-05-27]. Dostupné z WWW: <http://www.irisa.fr/english/home.html>.

[22] Fedora Project [online]. c 2011 [cit. 2011-05-28]. Anaconda/Kickstart - FedoraProject. Dostupné z WWW: <http://fedoraproject.org/wiki/Anaconda/Kickstart>.

[23] TCPDUMP/LIBPCAP [online]. c 2010 [cit. 2011-05-29]. Dostupné z WWW: <http://www.tcpdump.org/>.

[24] Red Hat Bugzilla [online]. c 2011 [cit. 2011-05-29]. Bug 567124 . Dostupné z WWW: <https://bugzilla.redhat.com/show_bug.cgi?id=567124>.

[25] Libvirt: The virtualization API [online]. c 2011 [cit. 2011-05-29]. Network XML format. Dostupné z WWW: <http://libvirt.org/formatnetwork.html>.