

Masarykova univerzita v Brně
Fakulta informatiky



IPv6 aplikace v Linuxu

Bakalářská práce

Michal Škrdla

2003

Prohlašuji, že tato práce je mým původním autorským dílem, které jsem vypracoval samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používal, nebo z nich čerpal, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

Obsah

1	Úvod	1
1.1	Jemný úvod	1
1.2	Jak to funguje	1
1.2.1	Adresace	1
1.2.2	Datagram	2
1.2.3	Připojení k síti	2
1.2.4	Mobilita	3
1.2.5	IPsec	4
2	Portování aplikací pro IPv6	5
2.1	Obecně	5
2.2	Praxe	7
2.3	IPv6 Socket Scrubber	10
2.4	Příklad: portování hry Quake	12
3	Implementace IPv6 v Linuxu	13
3.1	DNS/BIND	13
3.1.1	DNS	13
3.1.2	BIND	13
3.2	SSH/OpenSSH	14
3.3	HTTP/Apache	15
3.3.1	webové prohlížeče	15
3.4	FTP	16
3.4.1	FTP servery	16
3.4.2	FTP klienti	17
3.5	NFS/Samba	17
3.6	SMTP	17
3.6.1	SMTP servery	17
3.6.2	Mail klienti	18
3.7	POP3/IMAP	19
3.8	NTP/SNTP	19
3.9	SNMP	20
3.10	LDAP	20
3.11	SQL	20

4	Distribuce	21
4.1	PLD 1.0	21
4.2	RedHat Linux 9	21
4.3	Mandrake 8.1	21
4.4	Debian	21
5	Závěr	22

Kapitola 1

Úvod

1.1 Jemný úvod

Protokol IPv6 je vyvíjen již od počátku devadesátých let. Tato bakalářská práce se věnuje popisu stádia jeho současné implementace a zároveň metodikou implementace starších protokolů a programů pro využití IPv6.

Jedním z hlavních důvodů vzniku nového protokolu byl stále se zmenšující adresní prostor. Vzhledem k velké časové rezervě provedli vývojáři razantní změny ve struktuře datagramu a ve vlastnostech a funkcích nového protokolu. IPv6 tedy dostal zjednodušené hlavičky, mobilitu, bezpečnostní funkce, automatickou konfiguraci a hlavně rozsáhlý adresní prostor, který by měl být teoreticky nevyčerpatelný.

1.2 Jak to funguje

1.2.1 Adresace

Adresa má délku 128 bitů. Obvykle je zapisována jako osm čtveřic šestnáctkových čísel oddělených znakem `:`. Adresy lze zjednodušeně zapisovat vynecháním prvních nul respektive nahrazením sekvence nul symbolem `::`. Příkladem adresy a jejího následného zjednodušení může být:

```
1234:0000:0000:0000:4567:0000:89ab:cdef
1234:0:0:0:4567:0:89ab:cdef
1234::4567:0:89ab:cdef
```

IPv6 stejně jako jeho předchůdce nepřiděluje adresu počítači ale rozhraní. Avšak nový protokol jde ještě dále a dává rozhraní i několik adres. Jsou nadefinovány tyto základní typy:

unicast je adresou právě jednoho zařízení

multicast je adresou skupiny zařízení, data zaslaná na tuto adresu jsou doručena všem členům skupiny

anycast je adresou skupiny zařízení, ale data odeslaná na tuto adresu se doručují pouze nejbližšímu členu skupiny

1.2.2 Datagram

Datagram IPv6 má pevnou velikost hlavičky, která obsahuje jen nejdůležitější data. Ostatní informace byly vytlačeny do nepovinných rozšiřujících hlaviček. Standardní hlavička tedy obsahuje tyto údaje:

- verze protokolu — 4 bity, obsahuje vždy hodnotu šest
- priorita — 8 bitů
- identifikace toku — 20 bitů
- délka packetu — 16 bitů
- další hlavička — 8 bitů
- dosah — 8 bitů
- zdrojová adresa — 128 bitů
- cílová adresa — 128 bitů

Za úvodní standardní hlavičkou mohou být zřetězeny rozšiřující hlavičky. Pokud již žádná další hlavička nenásleduje je v poli **další hlavička** hodnota 59.

1.2.3 Připojení k síti

Jelikož nejde prohlásit, že od určitého data se bude používat právě a jen IPv6, museli autoři IPv6 vytvořit mechanismy, které dovolí současně používat oba protokoly a zároveň umožní jejich spolupráci. Existuje mnoho návrhů, avšak podle specifických vlastností je lze rozdělit do 3 variant:

Dvojí zásobník - zachovává kompletní funkčnost IPv4 a k té přidává funkce a vlastnosti IPv6, a tak může komunikovat s libovolným počítačem využívajícím libovolný z obou protokolů. Tento způsob je potřebný pro zbylé dvě varianty, protože ty vyžadují aby alespoň některá zařízení podporovala oba protokoly. Pro konfiguraci IPv4 se používá buď ruční konfigurace nebo DHCP, zatímco IPv6 používá automatickou konfiguraci (bezstavová nebo DHCPv6) nebo též nepopulární ruční konfiguraci. Komunikace mezi oběma protokoly probíhá až na aplikační vrstvě. Aplikace obdrží data a ta přetransformuje a odešle pomocí druhého protokolu adresátovi.

Tunelování - je metoda posílání datagramu jednoho protokolu skrz druhý. Datagram původního protokolu se zabalí do dat datagramu druhého protokolu a přenáší se po síti k cílovému stroji, kde se data opět rozbálí do původního datagramu. Mezi navržené metody patří

- 6to4 snaží se o propojení oblastí využívající IPv6 skrz Internet komunikující po IPv4. Této síti stačí jeden směrovač s přidělenou IPv4 adresou. Velkou výhodou tohoto protokolu je jeho nenáročnost. Stačí mu hraniční směrovač a záznamy v DNS.
- 6over4 se zaměřuje na osamocené počítače ve světě IPv4 a umožňuje jim kompletní komunikaci s ostatními počítači na IPv6. Pro správnou funkčnost musí klientský počítač podporovat oba protokoly a datagramy IPv6 jsou tunelovány sítí, která poskytuje připojení. Nevýhodou může být potřeba podpory pro skupinově adresované datagramy, která není běžná u IPv4.
- ISATAP je variantou k 6over4, avšak neklade žádné požadavky na IPv4 síť. Protokol slouží pouze ke komunikaci po lokální síti a venkovní je zajišťována jinými protokoly např. 6to4.
- DSTM předpokládá, že všechny lokální počítače umí komunikovat oběma protokoly, ale nemají přiděleny IPv4 adresy. Tato adresa je mu přidělena pouze na dobu nezbytně nutnou DSTM serverem. Nedostatkem tohoto návrhu je nemožnost navázat spojení z venkovní sítě a neřeší ani vazbu na DNS.
- Teredo řeší problémy spojené s NAT, hlavní ideou je nutnost zahajovat spojení zevnitř sítě k vytvoření vazby a následná manipulace s adresami, aby odpověď byla zaslána s původní cílovou adresou.

Translátoři - slouží k překládání informací mezi sítí pracující pouze na IPv4 a sítí pracující pouze na IPv6. Navrhnutými metodami jsou:

- SIIT používá bezstavový přístup, tudíž každý datagram překládá bez ohledu na předchozí zpracované datagramy. Pro své další potřeby využívá IPv4 překládané adresy. Má přesně definovaná pravidla pro překlady datagramů, ale neřeší vazby mezi adresami obou protokolů a jejich zanesení do DNS.
- NAT-PT je jedním z kandidátů pro široké nasazení. Opět potřebuje jistou škálu IPv4 adres, které poskytuje IPv6 počítačům. Umožňuje dynamické úpravy DNS. Překážkou tomuto řešení jsou některé aplikační protokoly obsahující v datech datagramu IP adresu. Pro tyto případy musí NAT-PT zasahovat do paketů a upravovat je, čímž dochází ke zbytečné zátěži.
- TRT je obdobou NAT-PT, ale svoji práci zahajuje v transportní vrstvě. Místo jednoho spojení mezi dvěma počítači hovořícími různými protokoly naváže spojení s každým z nich a sám se vydává za původního odesílatele.

1.2.4 Mobilita

Mobilita je jednou z důležitých inovací nového protokolu. V dnešní době přenosných zařízení je možnost komunikovat velmi důležitá.

Libovolné zařízení se vždy nachází někde doma, a proto tato **domácí síť** udržuje **domácí adresu** mobilního zařízení, která je uložena v DNS. Pod touto adresou je stroj vždy dosažitelný. Dříve než se stroj vydá na cestu, oznámí tuto skutečnost v domácí síti a zde je vybrán **domácí agent**, který po dobu nepřítomnosti mobilního zařízení přebírá veškerá jeho spojení. Po navázání spojení se ho snaží spojit s právě cestujícím zařízením pomocí

tunelu. To zároveň se svojí odpovědí začne provádět optimalizaci cesty, což je předání současné dočasné adresy původnímu odesílateli pro přímé navázání komunikace. Zároveň musí provést svoji autentizaci vůči adresátovi. Po vyřízení všech formalit odesílatel posílá stále pakety pro původní domácí síť, ale rozšířené o hlavičku směrování, která právě obsahuje vyjednanou dočasnou adresu.

1.2.5 IPsec

Oproti původní IPv4 byla implementace bezpečnostních funkcí jedním z požadavků a proto je tedy povinná. IPsec obsahuje dvě základní metody: šifrování a autentizaci. Šifrování utajuje přenášená data, zatímco autentizace umožní identifikovat odesílatele.

Pro IPsec jsou vyhrazeny dva typy rozšiřujících hlaviček:

Authentication Header Tato hlavička slouží k autentizaci odesílatele. V této hlavičce jsou autentizovaná data zahashovaná pomocí MD5 nebo SHA-1. Oba algoritmy používají dříve dohodnutý sdílený klíč, který se vygeneruje při navazování spojení.

Encapsulating Security Payload Tato hlavička pohltí všechna data datagramu krom hlaviček určených pro směrovače a informací obsahujících data o fragmentaci a zašifruje je. Každá implementace musí obsahovat podporu DES, MD5 a SHA-1.

Hlavičky je možno vkládat buď v transportním režimu, kdy jsou hlavičky standardně připojeny mezi rozšiřující hlavičky, či v tunelujícím režimu, který celý datagram zabalí do nového datagramu a tomu přidá nové hlavičky včetně bezpečnostních.

Při zabezpečené komunikaci se vytvoří virtuální spojení zvané **Bezpečnostní asociace (Security Association, SA)**, které udržuje veškeré informace jako je použitá metoda IPsec, doba životnosti, klíče, čítače, šifrovací algoritmus a další. Pro každý směr komunikace je potřeba mít vlastní bezpečnostní asociaci, tedy v rámci jednoho spojení je nutnost mít jednu asociaci pro příjem a druhou pro odesílání, z čehož vyplývá, že každému směru náleží i jiná dvojice klíčů.

O řízení bezpečnostních asociací se starají pravidla, která se uplatňují na všechny přicházející datagramy. Pokud se pozná, že se jedná o datagram pocházející z komunikace přes IPsec, použije se na něj příslušné pravidlo. Těmto sadám pravidel se říká **databáze bezpečnostní politiky (Security Policy Database, SPD)**.

Pro jejich správu byl navržen automatický protokol ISAKMP, ale je možná i ruční konfigurace. Ruční konfigurace však přináší mnoho úskalí, například při výměně klíčů či správě většího počtu šifrovaných komunikačních kanálů.

ISAKMP je protokolem pro automatickou správu bezpečnostních asociací. V první fázi komunikace si ISAKMP vytvoří vlastní bezpečnostní asociace, pomocí kterých bude dále vyjednávat další svoje zprávy. Dále už poskytuje informace ostatním protokolům. Při zahájení komunikace první účastník navrhne možný způsob komunikace s použitelnými šifrovacími algoritmy a druhá strana si vybere nejvhodnější možnost. Spojení pokračuje výměnou klíčů pro zvolené algoritmy.

Výměna klíčů však není obsahem protokolu ISAKMP. Pouze nechává tuto skutečnost na jiných externích technologiích. Zatím se používá **Internet Key Exchange (IKE)**, který funguje na principu Diffie-Hellmanova algoritmu.

Kapitola 2

Portování aplikací pro IPv6

2.1 Obecně

Typická činnost pro server ani klienta se v IPv6 neliší od svého předchůdce. Server nejdříve vytvoří socket, připojí lokální adresu k portu a čeká na spojení. Při pokusu o spojení akceptuje toto spojení a pak zapisuje a čte data.

Klient otevře socket, připojí se k serveru a následně čte nebo zapisuje data.

IP adresa je viditelná pro některé aplikace skrz socketové rozhraní.

Co je tedy potřeba změnit pro univerzálnost kódu?

- část API k rozšíření prostoru pro ukládání IP adresy, jde o novou datovou strukturu
- části aplikace, které manipulují s IP adresou

Změny v API

Je potřeba zpětná kompatibilita s existujícími aplikacemi a zdrojovými kódy, převážně pro IPv4. Pro snadnější portování aplikací je výhodné provést co nejméně změn v API a zároveň umožnit vzájemnou komunikaci mezi oběma protokoly.

	IPv4	IPv6
datové struktury	AF_INET in_addr sockaddr_in	AF_INET6 in6_addr sockaddr_in6
name-to-address functions	inet_aton() inet_addr() inet_ntoa()	inet_pton() inet_ntop()
address conversion functions	gethostbyname() gethostbyaddr()	getipnodebyname() getipnodebyaddr() getnameinfo() getaddrinfo()

Tabulka 2.1: Tabulka úprav

Úpravy v API

- Funkce používající protokolovou adresu jako funkční argument
- Datová struktura pro IPv6 adresy
 - AF_INET6** - nová rodina adres k identifikaci nové IPv6 struktury
 - in6_addr** - struktura pro IPv6 adresy
 - sockaddr_in6** - datová struktura k manipulaci s adresamioznačení nových struktur se liší pouze přidáním číslice „6“ do názvu proměnné, samotné nové struktury se samozřejmě liší obsazeným místem v paměti
- Funkce pro překlad jména na adresu pro oba protokoly
Nové funkce **inet_pton** a **inet_ntop** jsou použitelné pro oba protokoly a slouží k převodu adresy z binárního vyjádření na řetězec respektive obráceně.
- Funkce pro konverzi adres
Funkce **gethostbyname** a **gethostbyaddr** jsou specifické pouze pro protokol IPv4, nové funkce **getipnodebyname** a **getipnodebyaddr** podporují oba protokoly a funkce **getnameinfo** a **getaddrinfo** jsou nezávislé na použitém protokolu, a proto by se měly používat poslední dvě jmenované funkce pro maximální možnou přenositelnost kódu.

Potřebné úpravy na straně aplikací:

Na serveru:

- změna funkcí pracujících se sockety
- přizpůsobení logovacích funkcí, aby mohly manipulovat s delší IP adresou
- rozšíření datových struktur ukládajících IP adresu /v programu, databázích, atd./

U klienta:

- změna funkcí pracujících se sockety
- přizpůsobení logovacích funkcí
- přizpůsobení vzhledu a výstupu pro delší adresy

Některé aplikace používají „:“ k oddělení portu a adresy, ale IPv6 používá tento znak přímo ve své adrese, proto je nutno tuto adresu nějak oddělit. Využívají se hranaté závorky. Je tedy možno použít následující zápis:

```
http://[::1]/index.html
```

2.2 Praxe

Nejprve bych rád ukázal příklad úpravy programu akceptující pouze IPv4 na program akceptující IPv6 spojení. Použil jsem jednoduchý program, který pouze čeká na připojení na zadaném portu. Po připojení vypíše informace o vzdáleném klientovi a opisuje veškerý vstup, co dostane. Zdrojový kód je použit z <http://www.fi.muni.cz/~kas/p077/netread.c>. V dokumentu jsou červeně zobrazeny nahrazené texty a modře nová část kódu. Lze si všimnout, že rozdílů není příliš. Pouze ve specifických částech, kde se pracuje se strukturou adresy, přibývá číslice „6“ označující novou verzi protokolu.

```
#include <netdb.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <arpa/inet.h>

#define BUFSIZE (1<<14)

int main(int argc, char **argv) {
    struct protoent *pe;
    struct servent *se;
    int lsock, msgsock;
    struct sockaddr_in sin;
    struct sockaddr_in6 sin;
    /* Zde je nahrazena původní struktura, která je omezena na IPv4 novou struk-
    turou. */
    unsigned short port;
    int sz;
    int rd, wr;
    char buf[BUFSIZE], *p;

    if (argc != 3) {
        fprintf(stderr, "Usage: %s <protocol> <port>\n", *argv);
        return 1;
    }

    if (!(pe = getprotobyname(argv[1]))) {
        fprintf(stderr, "%s: no such protocol\n", argv[1]);
        return 1;
    }

    if ((lsock = socket(AF_INET, SOCK_STREAM, pe->p_proto)) == -1) {
    if ((lsock = socket(AF_INET6, SOCK_STREAM, pe->p_proto)) == -1) {
```

```

/* Při tvoření socketu oznamujeme, že chceme využívat IPv6. */
    perror("socket()");
    return 1;
}

if (!(se = getservbyname(argv[2], argv[1]))) {
    char *endptr = NULL;
    port = strtoul(argv[2], &endptr, 0);
    if (argv[2][0] == '\\0' || *endptr != '\\0') {
        fprintf(stderr, "%s/%s: no such service\n", argv[2], argv[1]);
        return 1;
    }
    port = htons(port);
} else
    port = se->s_port; /* Already in network order */

sin.sin_family = AF_INET;
sin.sin_addr.s_addr = INADDR_ANY;
sin.sin_port = port;
sin.sin6_family = AF_INET6;
inet_pton(AF_INET6, ":", &(sin.sin6_addr));
sin.sin6_port = port;
/* Jelikož jsme nahradili strukturu pro ukládání adresy musíme naplnit novou
strukturu. */

if (bind(lsock, (struct sockaddr *)&sin, sizeof(sin))) {
    perror("bind()");
    return 1;
}

if (listen(lsock, 128)) {
    perror("listen()");
    return 1;
}

sz = sizeof(sin);
if ((msgsock = accept(lsock, (struct sockaddr *)&sin, &sz)) == -1) {
    perror("accept()");
    return 1;
}

printf("Remote address: %s\n", inet_ntoa(sin.sin_addr));
printf("Remote port: %d\n", ntohs(sin.sin_port));
char straddr[INET6_ADDRSTRLEN];
inet_ntop(AF_INET6, &sin.sin6_addr, straddr, sizeof(straddr));
printf("Remote address: %s\n", straddr);

```

```

    printf("Remote port: %d\n", ntohs(sin.sin6_port));
/* Při výpisu adresy opět musíme uvažovat novou strukturu a tedy přizpůsobit
i část kódu specifickou pro IPv4 */
    fflush(stdout); /* Kdybychom nahodou psali do souboru */

    while((rd = read(msgsock, buf, BUFSIZE)) > 0) {
        for (p = buf; (wr = write(1, p, rd)) > 0; p+=wr)
            if (!(rd -= wr))
                break;
        if (wr <= 0) {
            perror("write()");
            return 1;
        }
    }
    if (rd < 0) {
        perror("read()");
        return 1;
    }
    return 0;
}

```

Jak je vidět v těle programu nedošlo k rozsáhlým změnám. Úpravy byly většinou velmi jednoduché a týkaly se pouze částí, kde se manipuluje s adresou. V druhé části se podíváme na část aplikace **libemi** <http://sourceforge.net/projects/libemi/>, kterou jsem si vybral k úpravám. Několik menších dílčích úprav v kódu přeskočím a zaměřím se přímo na nejdůležitější funkci, a to `tcp_init`, která se stará o připojení k serveru. Upravená funkce je skoro ve všech částech odlišná, což naplňuje očekávání, že univerzální program pro oba protokoly bude odlišný od kódu pouze pro jeden ze dvou protokolů.

```

static SOCKET tcp_init(const char* hostname, unsigned short port) {
static SOCKET tcp_init(const char* hostname, const char* port) {
/* První úpravou je změna prototypu funkce, jelikož v těle použiji funkci
getaddrinfo, která akceptuje jako parametr port typ const char* */
    SOCKET sock;
    struct sockaddr_in addr;
    struct hostent* host;
    struct addrinfo hints, *res;
    memset(&hints, 0, sizeof(struct addrinfo));
/* Opět potřebujeme změnit struktury, ve kterých se ukládají adresy, nyní
však jde o velmi rozdílná datová úložiště. */

    addr.sin_family = AF_INET;
    addr.sin_port = htons( port);
    addr.sin_addr.s_addr = *(unsigned long*)(host->h_addr_list[0]);
    hints.ai_family = AF_UNSPEC;
    hints.ai_socktype = SOCK_STREAM;
/* Znovu musíme vyplnit strukturu informacemi o jaké spojení půjde*/

```

```

host = gethostbyname(hostname);
if (!host)
    return EMI_SOCKETERR;

int n = getaddrinfo(hostname, port, &hints, &res);
if (n != 0) {
    fprintf(stderr, "getaddrinfo error:: [%s]\n", gai_strerror(n));
    return EMI_SOCKETERR;
}
/* Zjistíme informace o počítači, ke kterému se připojujeme, nový kód je
opět od základu různý od původního */

sock = socket(PF_INET, SOCK_STREAM, 0);
if (sock == INVALID_SOCKET)
    return EMI_SOCKETERR;

if (connect(sock, (struct sockaddr*)&addr, sizeof( addr))) {
    tcp_destroy(sock);
    return EMI_SOCKETERR;
}
while (res) {
    sock = socket(res->ai_family, res->ai_socktype, res->ai_protocol);
    if (sock == INVALID_SOCKET)
        return EMI_SOCKETERR;
    if (connect(sock, res->ai_addr, res->ai_addrlen)) {
        tcp_destroy( sock);
        return EMI_SOCKETERR;
    }
    res = res->ai_next;
}
freeaddrinfo(res);
/* Vytvoření socketu a otevření spojení k serveru */
return sock;
}

```

2.3 IPv6 Socket Scrubber

Zdroj: <http://www.sun.com/software/solaris/ipv6/>

Tato utilita od Sun Microsystems prohledává zdrojový kód v C a informuje o částech kódu pro IPv4. Sun distribuuje tuto utilitu pro svůj OS Solaris, ale není problém ji přeložit v libovolném unixovém prostředí.

Socket Scrubber generuje tyto soubory:

ipv6.all_files seznam všech souborů, které byly zpracovány

ipv6.last poslední zpracovaný soubor

ipv6.skipped soubory, které nejsou považovány za zdrojový kód

ipv6.primary.results zde jsou zobrazeny všechny části kódu, u kterých se předpokládá nutná změna kódu při portování, příkladem může být výskyt „AF_INET“

ipv6.secondary.results tento soubor obsahuje potenciální části kódu, které by mohly být též změněny, například se zde ale mohou zobrazit i definice „newssocket“, které není třeba měnit

Takto například vypadal výstup ze Socket Scrubberu pro aplikaci **libemi** soubor *ipv6.primary.results*:

```
*****
/home/mis/libemi-1.2/emi.c
*****
44: char origin_addr[32];
87: struct sockaddr_in addr;
99: host = gethostbyname( hostname);
102: sock = socket( PF_INET, SOCK_STREAM, 0);
105: addr.sin_family = AF_INET;
106: addr.sin_port = htons( port);
107: addr.sin_addr.s_addr = *(unsigned long*)(host->h_addr_list[0]);
607: emi->origin_addr,
859: sprintf( emi->buffer+1+13+1, "%.16s/0539/", emi->origin_addr);
896: if( !from || !(*from)) from = emi->origin_addr;
991: if( !from || !(*from)) from = emi->origin_addr;
1069: const char* origin_addr,
1080: emi, hostname, port, password, origin_addr);
1086: !origin_addr) return EMI_INITERR;
1105: strcpy( pp->origin_addr, origin_addr);
```

Velmi jednoduše lze potom nahlédnout do tisíciřádkového kódu a vyhledat místa, která je nutno upravit. Socket Scrubber označil též název proměnné *origin_addr*, kde se však vyskytuje klíčové slovo „addr“, které je částí mnoha struktur. Po těchto nutných úpravách ještě nahlédneme do souboru *ipv6.secondary.results*:

```
*****
/home/mis/libemi-1.2/emi.c
*****
17:#include <sys/socket.h>
59: closesocket( sock);
102: sock = socket( PF_INET, SOCK_STREAM, 0);
```

Zde se již objevují jen zmíněné výskyty slova *socket*, které ovšem není potřeba upravovat a portování aplikace **libemi** je hotové.

2.4 Příklad: portování hry Quake

Quake je velmi známá hra používající Internet pro multi-player. Hra se skládá ze serverové aplikace a klientů, kteří se k serveru připojují. Ve své IPv4 verzi se vyskytly problémy pro hráče, kteří do Internetu byli připojeni přes NAT. Právě zde lze pozorovat výhody IPv6. Kód hry Quake byl uvolněn pro Open Source komunitu v prosinci 1999. První hra využívající IPv6 se uskutečnila 12. ledna 2000.

Úprava kódu zabrala dvěma programátorům 32 hodin. V tomto čase zvládli analyzovat kód, vyhledat místa, která je potřeba upravit, upravit kód a připravit server pro první hru.

Byl změněn pouze jeden soubor, ve kterém byly lokalizovány všechny funkce pracující se sockety, a dále místo pro vyplnění jména serveru, ke kterému se budou klienti přihlašovat.

Z 146500 řádek kódu (včetně komentářů a prázdných řádků) bylo pouze 50 řádků pozměněno a 150 přidáno pro kompatibilitu mezi oběma protokoly.

Patch je dostupný v QuakeForge CVS a na stránkách Viagenié, která tento port připravila:

<http://www.quakeforge.net>

<http://www.viagenie.qc.ca/en/ipv6/quake/ipv6-quake.shtml>

Kapitola 3

Implementace IPv6 v Linuxu

3.1 DNS/BIND

3.1.1 DNS

První RFC (1886: DNS Extensions to support IP version 6) vyšlo v roce 1995 a obsahovalo specifikaci nových záznamů AAAA pro dopředné dotazy a PTR pro zpětné dotazy. Pro zpětné dotazy byla taky definována doména ip6.int.

Další RFC (2673: Binary Labels in the Domain Name System) zjednodušuje zápis reverzních dotazů. Umožňuje zapisovat adresy v přirozeném pořadí a rozdělovat je po jednotlivých bitech. Dotaz tedy má následující tvar:

```
\[xprefix_šestnáctivět/délka_prefixu]
\[x200107180801e010026008fffe5381cf/128].ip6.arpa
\[x026008fffe5381cf/64].\[x200107180801e010/64].ip6.arpa
```

V roce 2000 vyšlo nové RFC (2874: DNS Extensions to Support IPv6 Address Aggregation and Renumbering), které zavedlo další typ záznamů A6 a DNAME. První pro dopředné a druhý pro zpětné dotazy. Aby toho nebylo málo, používá i novou doménu ip6.arpa pro zpětné dotazy. Zároveň toto RFC prohlásilo předchozí za zastaralé a nedoporučovalo ho používat.

Avšak v létě 2001 došlo k dalším změnám, když bylo doporučeno opět používat RFC 1886 a nový návrh byl označen za experimentální. Další RFC (3152: Delegation of IP6.ARPA) pro změnu navrhuje doménu ip6.int a předepisuje doménu ip6.arpa z novějšího návrhu.

Co tedy používat?

Dle doporučení by se měly využívat záznamy AAAA a PTR a doména ip6.arpa. Binární prefixy jsou prohlášeny za experimentální, protože je některé servery neumí interpretovat a označí dotazy obsahující tyto prefixy jako nesprávné. Záznamy A6 a DNAME byly též označeny jako experimentální pro některé nedostatky.

3.1.2 BIND

První podpora IPv6 byla v BINDu již ve verzi 4.9.6. V této verzi podporoval RFC 1886. Verze 9 podporuje oba návrhy a tedy podporuje záznamy AAAA, A6, PTR i DNAME. Zároveň umí provádět převod mezi oběma specifikacemi, čímž zjednodušuje práci správci. Pro povolení IPv6 k naslouchání se musí přidat do konfiguračního souboru tyto řádky:

```
options {
    # ostatní nastavení
    listen-on-v6 { any; };
};
```

Pokud je použito i řízení přístupu pomocí ACL listů, je nutno upravit i je, například:

```
acl internal-net {
    127.0.0.1;
    1.2.3.4/24;
    ::1/128;
    ::ffff:1.2.3.4/128;
};
```

Závěrem je potřeba dodefinovat dopředné a reverzní zóny.

Dopředná zóna:

```
$TTL 86400
@      SOA tapka.ip6.skrdla.net. root.skrdla.net. (
        2003062603; 3H; 15M; 1W; 1D; )
      IN  NS      tapka
tapka  IN  AAAA    2001:718:801:e010:260:8ff:fe53:81cf
hannah IN  AAAA    2001:718:801:e010:2d0:59ff:fe65:14b7
debian IN  AAAA    2001:710:801:e010:2d0:59ff:fe65:14b8
```

Reverzní zóna:

```
$TTL 86400
@      SOA tapka.ip6.skrdla.net. root.skrdla.net. (
        2003062602; 3H; 15M; 1W; 1D; )
      NS      tapka.ip6.skrdla.net.
f.c.1.8.3.5.e.f.f.f.8.0.0.6.2.0 PTR tapka.ip6.skrdla.net.
7.b.4.1.5.6.e.f.f.f.9.5.0.d.2.0 PTR hannah.ip6.skrdla.net.
8.b.4.1.5.6.e.f.f.f.9.5.0.d.2.0 PTR debian.ip6.skrdla.net.
```

3.2 SSH/OpenSSH

OpenSSH bylo jedním z prvních projektů, které obsahovalo podporu IPv6 již v roce 2000. Standardně je ssh server nastaven pro akceptování obou protokolů. Přepínači -4 resp. -6 lze zvolit pouze IPv4 resp. IPv6. Pokud náhodou ssh démon není schopen přijmout IPv6 spojení, stačí pozměnit řádek v souboru *sshd_config* z:

```
ListenAddress 0.0.0.0 na ListenAddress ::
```

Tento zápis v Linuxu znamená „akceptuj připojení z obou protokolů“, zatímco BSD systémy používají tuto syntaxi pouze ve významu „poslouchej jen IPv6 adresy“. Proto v BSD musí být zápsány oba řádky.

3.3 HTTP/Apache

Verze 1.3 standardně neobsahuje podporu IPv6. Patch je možné stáhnout <ftp://ftp.kame.net/pub/kame/misc/>, ale pro využití IPv6 se doporučuje upgrade na verzi 2.x. Tyto verze již standardně obsahují podporu nového protokolu.

Po úspěšné instalaci stačí pozměnit konfiguraci v souboru *httpd.conf*

Nejprve řekneme webserveru, že má poslouchat na IPv6 socketu:

```
Listen [::]:80
```

Dále když už náš webserver akceptuje IPv6 můžeme nastavit i jednotlivé virtuální servery.

Jedinou změnou je rozšíření tagu `VirtualHost`:

```
<VirtualHost [2001:718:801:e010:260:8ff:fe53:81cf]:80 1.2.3.4:80>  
    ServerName www.ip6.skrdla.net  
    ...  
</VirtualHost>
```

3.3.1 webové prohlížeče

Aktuální verze všech oblíbených prohlížečů, krom linksu, podporují IPv6.

Opera

Opera ovládá IPv6 teprve od současné beta verze 7.20

Zdroj: <http://www.opera.com>

Aktuální verze: 7.20b

Mozilla

Zdroj: <http://www.mozilla.org/>

Aktuální verze: 1.4

Konqueror

Zdroj: <http://www.kde.org/>

Aktuální verze: 3.1.4

Netscape Navigator

Zdroj: <http://www.netscape.com/>

Aktuální verze: 7.1

Lynx

Zdroj: <http://lynx.browser.org/>

Aktuální verze: 2.8.4

3.4 FTP

Původní návrh implementace protokolu FTP pracoval pouze s 32-bitovými adresami. Proto RFC 2428:FTP Extensions for IPv6 and NATs přináší rozšíření, pomocí kterého lze využívat FTP i ve světě IPv6. RFC 2428 nahrazuje příkazy **PORT** a **PASV** novými rozšířenými příkazy **EPRT** a **EPSV**, které obsahují informaci o použitém protokolu. Příkaz EPRT má tuto syntaxi:

```
EPRT<space><d><net-prt><d><net-addr><d><tcp-port><d>
```

Za klíčovým slovem **EPRT** následuje mezera (ASCII 32), dále je použit oddělovač (ASCII 33 až 126, doporučen je znak „|“ - ASCII 124).

net-prt identifikuje použitý protokol (1 pro IPv4, 2 pro IPv6)

net-addr je zápisem IP adresy dané normou použitého protokolu

tcp-port obsahuje číslo portu, na kterém host očekává datové připojení

Příkladem může být:

```
EPRT |2|2001:718:801:e010:260:8ff:fe53:81cf|5282|
```

Formát příkazu **EPSV** je stejný. Odpovědí na tento požadavek může být:

```
Entering Extended Passive Mode (||6446|)
```

tedy ve tvaru

```
<informace o přechodu do pasivního módu> (<d><d><d><tcp-port><d>)
```

3.4.1 FTP servery

ProFTPD

ProFTPD ve verzi 1.2.9 neobsahuje oficiální podporu IPv6. Poslední dostupný patch je pro verzi 1.2.5 od PLD Teamu. V nových verzích 1.4 a testovacích verzích 1.3 se již s využitím IPv6 počítá. Oficiální podpora IPv6 byla přidána ve verzi 1.2.9rc2.

Zdroj: <http://www.proftpd.org>

Aktuální verze: 1.2.9

Patch pro verzi 1.2.5: <http://cvs.pld.org.pl/SOURCES/proftpd-1.2.5-v6-20020808.patch.gz>.

vsftpd

vsftpd obsahuje podporu IPv6 již od verze 0.2.

Zdroj: <http://vsftpd.beasts.org/>

Aktuální verze: 1.2.0

wu-ftpd

wu-ftpd je za posledních 5 let označováno jako nejvíce zranitelný a nejméně bezpečný ftp server. Přesto se stále vyskytuje v současných distribucích. IPv6 přímo nepodporuje, avšak existuje patch od PLD Teamu, který příliš nemění zdrojový kód a doplňuje podporu o IPv6.

Zdroj: <http://www.wu-ftpd.org/>

Aktuální verze: 2.6.2

Patch: <http://cvs.pld.org.pl/SOURCES/wu-ftpd-ipv6.patch>

3.4.2 FTP klienti

ftp

Na ftp klienta z Linuxového Netkitu také existuje patch, který umožní připojovat se k ftp serverům využívajícím IPv6.

Zdroj: <ftp://ftp.uk.linux.org/pub/linux/Networking/netkit>

Aktuální verze: 0.17

Patch: <ftp://ftp.linux-ipv6.org/pub/usagi/stable/split>

NcFTP

Populární sada programů NcFTP nepodporuje v základní distribuci IPv6, avšak na stránkách KAME, lze nalézt patch pro verze $\leq 3.1.5$.

Zdroj: <http://www.ncftp.com/ncftp/>

Aktuální verze: 3.1.6

Patch: <ftp://ftp.kame.net/pub/kame/misc>

Konqueror

HTTP/FTP prohlížeč integrovaný v KDE podporuje, stejně jako celé prostředí KDE, IPv6 naprosto bez problému již od verze 2.2.

Zdroj: <http://www.kde.org/>

Aktuální verze: 3.1.4

3.5 NFS/Samba

V současné době neexistuje port na IPv6 pro žádnou z aplikací: nfsd, mountd, mount.

Pro Sambu existuje patch, který ovšem není zařazen do hlavního vývojového kódu, kvůli nedostatku podpory ze strany Samba klientů od Microsoftu.

Patch pro Sambu: <http://v6web.litech.org/samba/>

3.6 SMTP

3.6.1 SMTP servery

sendmail

Nejrozšířenější, nejrobustnější a nejznámější MTA /mail transport agent/, mnohými zatracován pro spoustu bezpečnostních chyb a složitou správu, obsahuje velmi dobrou podporu IPv6.

Při zápisu adres z rozsahu IPv6 je potřeba před zmíněnou adresu dávat prefix: „IPv6:“

Zdroj: <http://www.sendmail.org/>

Aktuální verze: 8.12.10

postfix

Systém podstatně jednodušší na správu, rychlejší a bezpečnější než sendmail, za ním zao-
stává. Přímá podpora IPv6 v dodávaných zdrojových kódech není, ale Dean Strik navazuje
na projekty KAME a PLD Teamu a píše patche na všechny nové verze postfixu. Krom pod-
pory IPv6 přidává do tohoto systému i podporu přenosu dat přes TLS.

Zdroj: <http://www.postfix.org/>

Aktuální verze: 2.0.16

Patch: <http://www.ipnet6.org/postfix/>

qmail

MTA od Dana Bernsteina, vyznačuje se modulárností s vysokým důrazem na bezpečnost.
Autor qmail neobdařil podporou IPv6 „díky“ svému přesvědčení o tomto protokolu, ale i
přesto existuje několik patchů.

Zdroj: <http://www.qmail.org/>

Aktuální verze: 1.03

Patch: <http://pyon.org/fujiwara/>

3.6.2 Mail klienti

KMail

Součástí grafického prostředí KDE, které se již od verze 2.2 pyšní podporou IPv6, taktéž
nezůstává pozadu a obsahuje vše, co je třeba.

Zdroj: <http://www.kde.org/>

Aktuální verze: 3.1.4

pine

Zdroj: <http://www.washington.edu/pine/>

Aktuální verze: 4.56

Patch: <http://www.ngn.euro6ix.org/IPv6/pine/>

mutt

mutt nemá s novým protokolem žádné problémy.

Zdroj: <http://www.mutt.org/>

Aktuální verze: 1.4.1

Mozilla

Mozilla mail klient zvládá SMTP, POP3 i IMAP4 nad IPv6.

Zdroj: <http://www.mozilla.org/>

Aktuální verze: 1.4

Opera

Mail klient v Opeře by /stejně jako celá Opera/ měl mít možnost využívat IPv6 od beta verze 7.20b. Dřívější verze 7.x podporu nemají.

Zdroj: <http://www.opera.com/>

Aktuální verze: 7.11

3.7 POP3/IMAP

U všech zmíněných POP3/IMAP démonů jsem nenarazil na žádný závažný problém, který by omezoval IPv6.

solidpop3d

Zdroj: <http://solidpop3d.pld.org.pl/>

Aktuální verze: 0.15

courier-pop3d

Zdroj: <http://courier.sourceforge.net/>

Aktuální verze: 0.42.2

courier-imapd

Zdroj: <http://courier.sourceforge.net/>

Aktuální verze: 0.42.2

cyrus-imapd

Poslední stabilní verze ještě neobsahuje IPv6, ale v nové beta verzi již tato podpora je.

Zdroj: <http://asg.web.cmu.edu/cyrus/imapd/>

Aktuální verze: 2.2.1-BETA

Patch pro verzi 2.1.15: <http://www.imasy.or.jp/~ume/ipv6/>

3.8 NTP/SNTP

Network Time Protocol a Simple Network Time Protocol jsou protokoly pro synchronizaci času po internetu. Port na IPv6 připravila kanadská firma Viagénie. Nejprve vytvořila server i klienta pro jednodušší protokol SNTP <http://www.viagenie.qc.ca/en/ipv6/ntp6/sntp-ipv6-1.0.tar.gz>. Tento zdrojový kód je nezávislý na použitém protokolu. Po úspěšném dokončení tohoto projektu došlo i na NTP. Programátoři této firmy vytvořili server démona, který je napsán pouze pro využití s protokolem IPv6.

Tento server je k dispozici na adrese: ntp.ipv6.viagenie.qc.ca

Patch pro ntp klienta verze 4.1.0 od firmy Viagénie:

<http://www.viagenie.qc.ca/en/ipv6/ntp6/ntp-4.1.0-ipv6.patch.tar.gz>

Nová, zatím experimentální, verze **ntpd** 4.1.80-rc1 již obsahuje nativní podporu IPv6.

Zdroj: <http://www.ntp.org/>

3.9 SNMP

V současné době již existují některé IPv6 MIBy¹, ale jsou jen přepracované z IPv4. MIB používající multicast v IPv6 zatím neexistuje.

SNMP informace jsou nyní přenášeny přes IPv4 spojení, což je převážně zapříčiněno nedostatečnou podporou v komerčních řešeních. Tento stav je ale velmi nevýhodný, protože IPv6 zůstává velmi závislá na svém předchůdci. Zároveň existuje velmi mnoho zařízení, která je v budoucnosti potřeba spravovat právě pomocí SNMP, jako příklad mohu uvést: tiskárny, switche, bezdrátové access pointy atd.

Jedním z prvních projektů podporujících IPv6 je NET-SNMP:

Zdroj: <http://net-snmp.sourceforge.net>

3.10 LDAP

OpenLDAP obsahuje podporu IPv6. Pokud není podpora implicitně zapnutá, stačí překompilovat zdrojové kódy s přepínačem `--enable-ipv6`

3.11 SQL

Prvním databázovým strojem podporující IPv6 se stal 17.11.2003 PostgreSQL ve své nové verzi 7.4.

V ostatních databázových strojích, jako je mySQL nebo Oracle, zcela chybí. Přestože existují patche pro současné verze mySQL, vývojáři je stále nezařadili do oficiálních zdrojových kódů.

¹MIB = Management Information Base

Kapitola 4

Distribuce

4.1 PLD 1.0

Tato nepříliš známá polská distribuce disponuje v současnosti nejlepší podporou IPv6. Mnoho patchů vyšlo právě z této dílny.

Distribuce PLD obsahuje všechny dostupné aplikace, které využívají nový protokol.

4.2 RedHat Linux 9

RedHat Linux je další distribucí, která je velmi dobře vybavena pro IPv6. Konfigurace je zde velmi jednoduchá v **initscripts**. Redhat obsahuje pouze aplikace, které mají oficiální IPv6 patche, ostatní programy je potřeba opatchovat a zkompileovat individuálně. Vzhledem k tomu, že PLD vychází právě z RedHatu, je možné použít i instalační balíčky nebo alespoň zkompileovat balíčky se zdrojovými kódy z PLD Linuxu.

Defaultní nastavení je bez IPv6.

4.3 Mandrake 8.1

Distribuce Mandrake je velmi podobná RedHat Linuxu, avšak přijde mi více uživatelsky orientovaná. IPv6 zde také není defaultně zapnuta, a je proto potřeba nastavit síťové nastavení v konfiguračních souborech, které jsou zpracovávány jiným programem a byly hůře čitelné.

4.4 Debian

Debian Woody 3.0 obsahuje nejhorší podporu IPv6 ze všech zmíněných distribucí. Jelikož se ale jedná o distribuci, vzhledem k ostatním, poměrně starou není se čemu divit. Pokud však uživatel povolí používání vývojových a testovacích balíčků, které lze stáhnout například pomocí apt-get, dostává velmi rozmanitou paletu nástrojů pro IPv6, čímž pravděpodobně předčí obě výše zmíněné distribuce a stane se srovnatelným oponentem PLD Linuxu.

Kapitola 5

Závěr

V této bakalářské práci jsem se pokusil obsáhnout potřebné informace o současném stavu IPv6 v linuxových distribucích. Většina aplikací nyní počítá s využitím nového protokolu, a proto je jejich kód již připraven pro přechod na protokol IPv6.

Z distribucí si nejlépe vede polská distribuce PLD, která je zaměřena právě na podporu IPv6 a mnoho jejích tvůrců pracuje na úpravách současných aplikací. Další velké distribuce, jako jsou RedHat, Mandrake, SuSE a Debian, nemají s IPv6 problémy, avšak některé specifické aplikace je potřeba opatchovat nebo použít testovací verze.

Portování aplikací je velmi jednoduché. Práce se sockety je velmi podobná, přibývá pouze několik nových funkcí. K vyhledání kódu, který je nutno upravit, lze použít utilitu Socket Scrubber, díky níž nemusí programátor procházet celý kód, ale může se zaměřit pouze na podstatné části. Změny nejsou příliš rozsáhlé, většinou se jedná o zlomek procenta celé aplikace.

A jaká je budoucnost IPv6? Zatím se stále jedná o vyvíjený protokol, kterému stále ještě chybí mnoho detailů k dokonalosti, zejména nezáměr některých velkých společností brzdí vývoj a využití tohoto protokolu. Po vyřešení důležitých částí, jako je například mobilita, IPsec, multihoming a dalších, dojde jistě k masovému rozšíření a využívání všech výhod, které nám poskytne.

Zároveň je velkým problémem i přístup velkých výrobců nezávislého software, jako je například Oracle, kteří nevěnují IPv6 pozornost.

Literatura

- [1] <http://www.deepspace6.net>. Aktuální informace o dění ve světě IPv6.
- [2] <http://www.ipv6.org>. Oficiální stránka projektu.
- [3] http://www.sun.com/software/solaris/ipv6/porting_guide_ipv6.pdf. Příručka o portování IPv6 od SUNu.
- [4] <http://www.tldp.org/howto/linux+ipv6-howto/>. IPv6 HOWTO.
- [5] Satrapa Pavel. *IPv6*. Neortex, 2002.