VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

KOMUNIKACE POMOCÍ IPV6

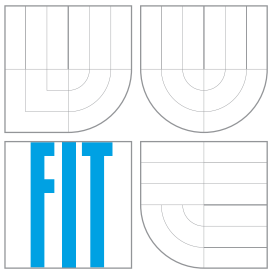
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

VÁCLAV PECHÁČEK

BRNO 2010

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

KOMUNIKACE POMOCÍ IPV6

IPV6 COMMUNICATION LIBRARY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — VÁCLAV PECHÁČEK
AUTHOR

VEDOUCÍ PRÁCE — Ing. ZDENĚK VAŠÍČEK
SUPERVISOR

BRNO 2010

Abstrakt

Práce se zabývá síťovou komunikací na bázi IPv6. Cílem bylo vytvořit knihovnu, která umožní komunikaci nad IPv6 vývojové desce FITkit vybavené rozšiřujícím modulem pro zapojení do sítě typu Ethernet. Součástí výsledného řešení je aplikace demonstrující funkce knihovny. Použitým jazykem je ANSI C.

Abstract

Thesis deals with IPv6-based network communication. The aim was to create a library enabling FITkit development board equipped with Ethernet extension module to act as an IPv6 node. Project includes application to demonstrate available functionality. The library is written in ANSI C.

Klíčová slova

komunikace, síť, vestavný systém, Ethernet, IPv6, FITkit, MSP430, ENC28J60

Keywords

communication, network, embedded system, Ethernet, IPv6, FITkit, MSP430, ENC28J60

Citace

Václav Pecháček: Komunikace pomocí IPv6, bakalářská práce, Brno, FIT VUT v Brně, 2010

Komunikace pomocí IPv6

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Zdeňka Vašíčka.

.....
Václav Pecháček
12. května 2010

Poděkování

Děkuji Ing. Zdeňku Vašíčkovi za podrobné konzultace, rozsáhlou podporu a rady v průběhu celé práce. Především si vážím rychlé a souvislé pomoci při řešení nečekaných problémů stejně jako pečlivého posouzení obsahu práce. Dále děkuji za poskytnutí nově navrženého rozšiřujícího modulu pro zapojení FITkitu do sítě typu Ethernet.

Obsah

Kapitola 1

Úvod

Internet Protocol, version 6 (IPv6) postupně nahrazuje svého předchůdce, IPv4, v roli základního protokolu pro počítačovou komunikaci v lokálním i celosvětovém měřítku. Zároveň využívá celou sadu nových podpůrných mechanismů a jeho implementace tak klade specifické nároky i na obslužné rutiny, které se samotným IP protokolem bezprostředně nesouvisí. [27, 17] Náplní mé práce je komunikace v síti založená právě na tomto protokolu.

Mezi hlavní přednosti IPv6 bývá řazen rozsáhlý adresový prostor a s ním spojená možnost přímé komunikace uzlů kdekoliv v Internetu bez překladových mechanismů. [27, 22] Tyto výhody napomáhají snadnému zapojení většího množství vestavných zařízení, jako jsou např. prvky senzorových sítí, do Internetu. [23] S oblastí úsporných IPv6 uzlů práce úzce souvisí.

Cílem projektu bylo vytvořit knihovnu, která umožní komunikaci nad IPv6 platformě FITkit, rozšířené o modul pro zapojení do sítě typu Ethernet. [10] Obě tyto vývojové desky byly navrženy speciálně pro FIT VUT v Brně, obsahují však běžně dostupné komponenty. Použitým jazykem je ANSI C a knihovnu by tak mělo být možné upravit pro podobná cílová zařízení s minimálním úsilím. Výsledný soubor funkcí jsem nazval *knihovnou NET6*.

Práce je členěna následovně: v druhé kapitole zasadím IPv6 a podpůrné protokoly do rámce referenčního modelu ISO/OSI. Popíši konkrétní mechanismy významné pro praktickou část práce a zmíním jejich vazbu na nižší i vyšší síťové vrstvy. Ve třetí kapitole poskytnu základní přehled o cílové platformě, zejména pak o mikrokontrolérech jako nejvýznamnějších prvcích použitého hardware. Čtvrtá kapitola je věnována samotné knihovně, její struktuře, konvencím a zejména dostupným funkcím. Poslední, pátá kapitola pak představuje návod, který by měl čtenáři usnadnit praktické využití knihovny.

V textu používám následující konvence: důležité pojmy, názvy protokolů i polí jsou uvedeny *kurzívou*. Názvy souborů, identifikátory ze zdrojových kódů a konstanty jsou odlišeny `strojopisně`. Při popisu skupin souborů používám znaku `*` (ve významu obvyklém pro UNIXové systémy), v případě identifikátorů vymezuji obecné části názvu znaky < a > (mezi nimi uvádím krátké vysvětlení). Dále jsem se rozhodl ponechat názvy protokolů, struktur a jejich polí v angličtině. Překlad by mohl být vzhledem k převažujícím anglickým zdrojům nesprávný či matoucí.

Kapitola 2

IPv6 a protocol stack

Internet Protocol, version 6 (IPv6) začala skupina Internet Engineering Task Force (IETF) navrhovat na začátku 90. let 20. století, kdy počet uzlů v Internetu významně stoupal. Předchozí verze, označovaná jako IPv4, se začala jevit jako nedostatečná zejména v souvislosti s tenčící se zásobou volných adres, nevhodným formátem paketů pro vysokorychlostní routing, netransparentním propojváním sítí i koncových bodů a nedostatečnou podporou QoS a toků. Mezi výhody nového protokolu se kromě nápravy zmíněných nedostatků často řadí také jednodušší formát základní hlavičky či snazší automatická konfigurace uzlů. [27, 22]

První oficiální specifikací se stalo RFC 1883 z prosince roku 1995, které bylo záhy nahrazeno o tři roky mladším, dodnes platným *RFC 2460: Internet Protocol, Version 6 (IPv6)*. [17] Tento dokument IETF postupně doprovodila množstvím dalších, které společně vedly k zajištění podpory nové verze IP, ať už modifikací dosavadních protokolů (UDP, DNS, OSPFv3) či tvorbou nových (RIPng, ICMPv6, NDP, DHCPv6). [27]

Má-li tedy implementace síťového rozhraní na daném uzlu podporovat IPv6, promítne se tato skutečnost ve většině protokolů v použité sadě, označované jako *protocol stack*[1]. Právě o prvcích *protocol stacku* knihovny *NET6* pojednává tato kapitola.

2.1 Referenční model ISO/OSI

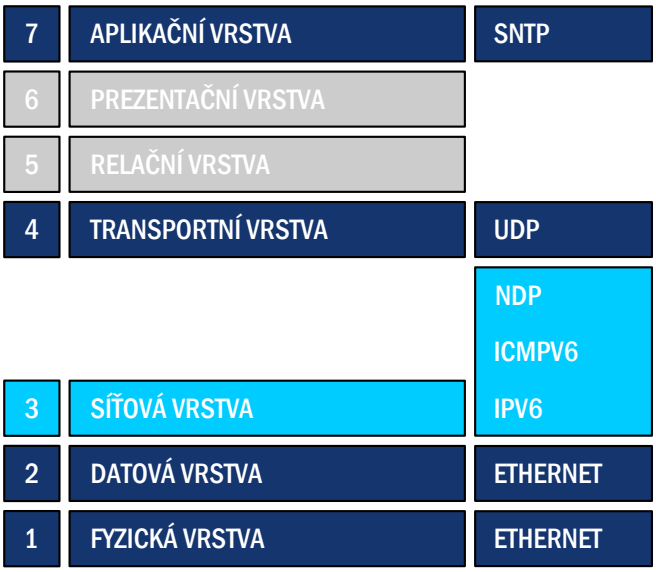
Sedmivrstvý referenční model ISO/OSI umožňuje uvedení použitých síťových protokolů do vzájemného kontextu a vyjádření jejich role při komunikaci. [14] Pro účely této práce nepovažuji za důležitý rozbor první, tedy fyzické vrstvy, kterou na základě standardu Ethernet kompletně obsluhuje hardware rozšiřujícího modulu. Podobná je situace u páté a šesté vrstvy, na nichž se v rámci projektu nevyužívá žádný protokol.

Klíčová je vrstva třetí, do níž se řadí protokoly IPv6, ICMPv6 a NDP. Významná je také linková část standardu Ethernet (2. vrstva) a pro úplnost je třeba zmínit i služby UDP (4. vrstva) a SNTP (7. vrstva) využívané v rámci knihovny především pro demonstrační účely. Grafické znázornění najdeme na obrázku 2.1.

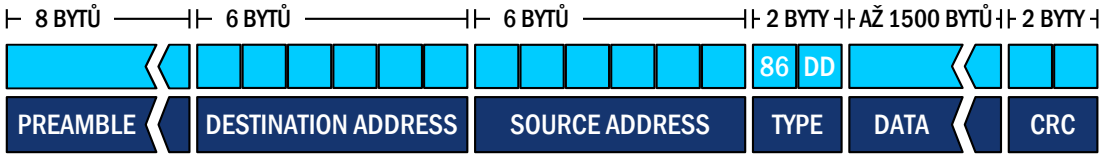
2.2 Ethernet

Standard *Ethernet* umožňuje adresovat dvě a více fyzických rozhraní na jedné lince, mezi nimiž je třeba přenést rámec – jeho základní datovou jednotku. Formát rámce je znázorněn na obrázku 2.2. [28]

[1] Výstižný překlad tohoto spojení není snadné najít. [26]



Obrázek 2.1: Zasazení služeb do referenčního modelu ISO/OSI.



Obrázek 2.2: Struktura Ethernetového rámce.

Přímý vztah k IP vrstvě může mít pole *Destination address*. Podobně jako u IPv4 totiž existuje zobrazení skupinových IP adres do množiny skupinových linkových adres, které se u Ethernetu obecně vyznačují tím, že mají v nejméně významném bitu prvního bytu jedničku. Pro IPv6 byla vyhrazena podmnožina těchto adres začínající dvěma byty `33:33` hexadecimálně, přičemž další 4 byty se přebírají z posledních 4 bytů cílové IP adresy. Koncept všesměrového vysílání (broadcast) s cílovou adresou `FF:FF:FF:FF:FF:FF` se u IPv6 nevyužívá. [27, 18]

Určitý význam má také pole *CRC*. V IPv6 hlavičce bylo totiž pole pro kontrolní součet vypuštěno pro urychlení směrování a autoři návrhu se zaštiťují právě tím, že mnohé linkové vrstvy implementují vlastní mechanismus kontrolního součtu. [27] U Ethernetu je tedy tento předpoklad splněn. 16bitové pole *Type* charakterizuje náplň rámce a v případě IPv6 datagramu má hodnotu `86DD` hexadecimálně. [16] Jiné typy rámců knihovna nepodporuje.

2.3 IPv6

Nyní vysvětlím základní principy samotného IPv6 protokolu, který umožňuje na základě IPv6 adresy najít cestu k cíli, ať už je jím uzel na lince, v Internetu nebo jde např. o zařízení samotné (lokální smyčka).

2.3.1 Obecný formát IPv6 adres a jejich členění

Výrazný nárůst délky adresy z původních 32 bitů u IPv4 na současných 128 bitů u IPv6 s sebou přinesl nutnost nové notace. Hodnoty jednotlivých bytů se zapisují v šestnáctkové soustavě a skupiny po dvou bytech (tedy čtveřice znaků) jsou odděleny dvojtečkou. Protože mnohé adresy obsahují významné množství nul, byla navíc zavedena následující pravidla umožňující zkrácení zápisu:

- počáteční nuly v určité skupině mohou být vynechány
- jednu nebo několik těsně po sobě následujících nulových skupin lze jednou v rámci adresy nahradit dvěma dvojtečkami (`::`)

Podobně jako u IPv4 adres jsou k dispozici tzv. *prefixy*. Ty reprezentují pouze část adresy od jejího počátku po bit určený délkou prefixu, která se uvádí za adresou v desítkové soustavě a je od ní oddělena lomítkem. U IPv6 adres tedy může nabývat hodnot v rozsahu `0-128`. [27, 22] Adresa vyhovuje prefixu, pokud se v celé délce prefixu s prefixem shoduje.

Správu IPv6 adres má na starost organizace Internet Assigned Numbers Authority (IANA). Ta definovala několik obecných skupin vyjádřených pomocí prefixů [4], přičemž v tabulce 2.1 jsou uvedeny pouze ty, které jsou významné pro praktickou část práce.

Prefix	Název skupiny
`::1/128`	lokální smyčka
`fe80::/10`	lokální linkové individuální adresy
`2000::/3`	globální individuální adresy
`ff00::/8`	skupinové adresy

Tabulka 2.1: Významné skupiny IPv6 adres.

2.3.2 Adresy pro konkrétní uzel v síti

Před zahájením tvorby vlastních adres potřebuje zařízení nejprve znát *identifikátor rozhraní* o délce 64 bitů [19], který se odvozuje na základě standardu *IEEE EUI-64*. V případě Ethernetu se využívá vhodných vlastností 48bitové MAC adresy, do jejíhož středu se pouze doplní dva byty s hodnotou `FF:FE`. IPv6 si však žádá oproti standardu drobnou odchylku. V nejvýznamnějším bytu je třeba upravit bit představující *příznak globality*, přičemž jeho nastavení na hodnotu `1` značí, že vytvořený identifikátor by měl být celosvětově jednoznačný. Tím vznikne *modifikovaný EUI-64*, který je již pro tvorbu IPv6 adres na daném rozhraní vhodný. [22, 19]

Každý uzel podporující IPv6 si musí sestavit a používat nemalé množství vlastních adres. Obvykle jde přinejmenším o jednu adresu z každé skupiny uvedené v tabulce 2.1.

V první řadě jde o adresy *individuální (unicast)*, určené pro komunikaci mezi dvěma uzly. Kromě adresy lokální smyčky `::1` mezi ně povinně patří *lokální linková individuální adresa*, složená z prefixu `fe80::/64` a identifikátoru rozhraní. Má-li být zařízení dostupné v Internetu, bude potřebovat také alespoň jednu *globální individuální adresu*. Ta bude typicky začínat sekvencí `2001` (prostor přidělovaný regionálním registrátorům), přičemž prvních 64 bitů představujících *prefix sítě* bude opět doplněno identifikátorem rozhraní.

IPv6 uzly jsou povinny naslouchat také hned na několika adresách *skupinových (multicast)*. V první řadě jde o obecné adresy reprezentující všechny uzly v rámci rozhraní (`ff01::1`) a v rámci linky (`ff02::2`). Pro snížení objemu síťového provozu, který musí jednotlivé uzly zpracovávat, zavádí IPv6 navíc tzv. *skupinovou adresu pro vyzývaný uzel*. Vzniká složením prefixu `ff02::1:0:0:0:0/96` a posledních čtyř bytů identifikátoru rozhraní. [27]

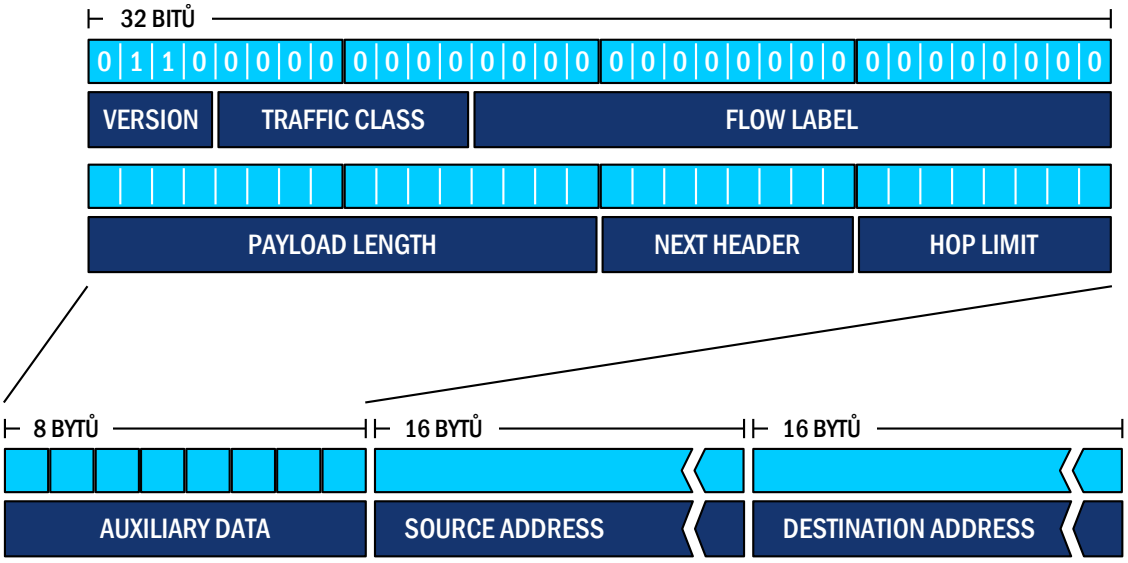
2.3.3 Automatická konfigurace

Vlastní adresy i další údaje potřebné pro zapojení do Internetu lze na každém uzlu nastavit ručně. IPv6 však přichází s výraznou podporou automatické konfigurace zařízení, která může probíhat dvěma způsoby. První je označovaná jako *bezestavová (stateless)* a vyznačuje se tím, že přidělování údajů neřídí žádný centrální prvek (např. server), ale zařízení samotné. Tento způsob, založený na mechanismech NDP (kapitola 2.5) a definovaný v samostatném *RFC 2462: IPv6 Stateless Address Autoconfiguration*, je pro IPv6 uzly dokonce povinný. [29, 13]

Druhá metoda označovaná jako *stavová (stateful)* automatická konfigurace staví na protokolu *DHCPv6* a podobá se mechanismům používaným u IPv4. Tzv. *pronájem adres* v tomto případě řídí DHCPv6 server. [27] Stavová konfigurace však není v rámci knihovny *NET6* využívána.

2.3.4 Základní hlavička

Každý IPv6 datagram začíná hlavičkou o pevné délce 40 bytů, jejíž formát je znázorněn na obrázku 2.3.

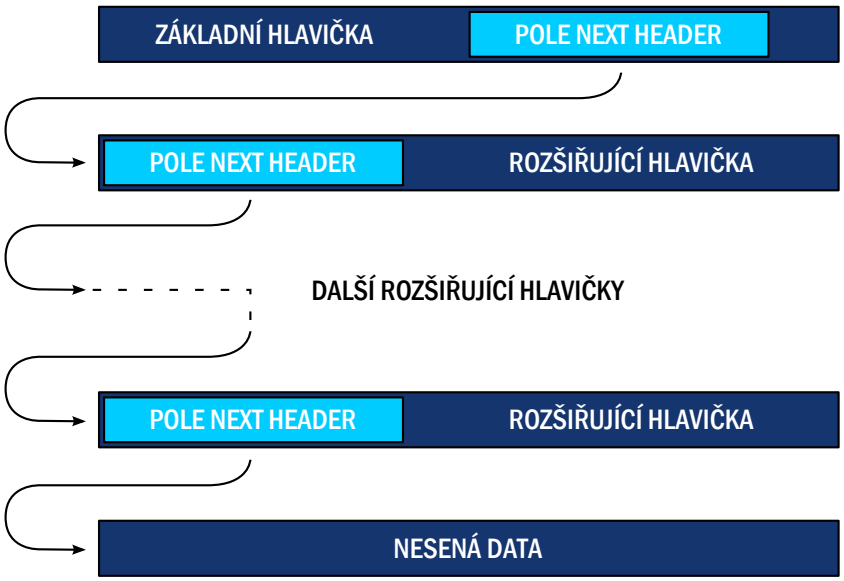


Obrázek 2.3: Formát základní hlavičky IPv6 datagramu.

Pole *Version* se vztahuje k IP protokolu a obsahuje tedy konstantu `6`. Hodnoty *Traffic class* a *Flow label* týkající se QoS a podpory toků jsou určeny především nekoncovým

prvkům sítí a v knihovně *NET6* se nevyužívají. Délka dat *Payload length* je vyjádřena v bytech a počítá se od 41. bytu datagramu dál. [17, 18]

Pole *Next header* je prvním znakem významného konceptu rozšiřujících hlaviček. Návrh IPv6 vyšel z předpokladu, že málokterou informaci je třeba umisťovat do každého datagramu, a uvedená základní hlavička proto obsahuje minimum údajů. Je-li třeba přenést nějakou hodnotu navíc, vloží se do pole *Next header* pouze její typ (8bitový kód) a samotná data jsou umístěna až za základní hlavičku v podobě tzv. *rozšiřující hlavičky*. U ní se situace opakuje – kromě potřebných údajů má totiž také vlastní pole *Next header*, které může ukazovat na další rozšiřující hlavičku. Zmíněným způsobem se postupuje, dokud je třeba přikládat další netypické informace. Poté se namísto kódu další hlavičky uvede typ nesených dat, případně zvláštní kód *No next header*. [27, 18, 20] Obecně je tento mechanismus znázorněn na obrázku 2.4. Konkrétním rozšiřujícím hlavičkám a nabídce kódů nesených dat se budu věnovat v následující podkapitole.



Obrázek 2.4: Mechanismus rozšiřujících hlaviček.

8bitová hodnota *Hop limit* říká, přes kolik směrovačů může datagram projít, než bude zahozen, a jde tedy o obdobu *Time to live (TTL)* v IPv4. Zbývající pole *Source address* a *Destination address* pak obsahují odpovídající IPv6 adresy. [17]

2.3.5 Rozšiřující hlavičky

Kompletní seznam kódů rozšiřujících hlaviček i nesených dat poskytuje IANA na webu [4], avšak pro běžné uzly je obvykle významná jen jeho malá část.

Hlavičky *Hop-by-hop options* a *Destination options* představují obecnější struktury, které mohou obsahovat různé nezávislé *volby*. Z úsporných důvodů je povinné umisťovat *Hop-by-hop options* jako první v pořadí, aby směrovače nemusely *Destination options* vůbec zpracovávat. [18, 17] Ostatní rozšiřující hlavičky jsou obvykle konkrétnější a nesou údaje nutné např. pro fragmentaci nebo zabezpečení datagramů.

2.3.6 Neznámé hlavičky a volby

Koncept rozšiřujících hlaviček počítá s tím, že uzel nemusí rozpoznat kód některé z hlaviček obsažených v příchozím datagramu. Může to být způsobeno tím, že ji z nějakého důvodu nepodporuje nebo skutečností, že v době návrhu obslužných rutin vůbec nebyla definována. *RFC 2460* pro tento případ ukládá uzlu jednoznačný postup: celý datagram je třeba zahodit a jeho odesílateli zaslat ICMPv6 zprávu *Parameter problem* (více v kapitole 2.4.2). [17] Správnou hodnotou pro pole *Code* je v tomto případě `1`.

Při zpracování obecnějších hlaviček *Hop-by-hop options* a *Destination options* může dojít k podobnému problému, narazí-li zařízení na volbu s neznámým identifikátorem. V tomto případě jsou požadavky na reakci uzlu rozmanitější a odvíjejí se od dvou nejvýznamnějších bitů v čísle identifikujícím problematickou volbu. [27] Shrnuji je v tabulce 2.2.

Hodnota	Akce
`00`	přeskočit volbu a pokračovat dalšími
`01`	zahodit datagram
`10`	zahodit datagram a odesilateli zaslat zprávu *Parameter problem*
`11`	zahodit datagram a odesilateli zaslat zprávu *Parameter problem* nešlo-li o skupinové vysílání (multicast)

Tabulka 2.2: Chování při nalezení neznámé volby v přijatém datagramu.

2.3.7 IPv6 pseudo-hlavička

Protokoly vyšších vrstev často provádějí kontrolní součet, do něhož kromě vlastních dat zahrnují také určitý soubor údajů souvisejících s IP vrstvou. *RFC 2460* říká, že každý takový protokol musí být upraven pro použití nad IPv6. Ve specifikaci je uveden vzorový set údajů označený jako *IPv6 pseudo-header for UDP and TCP*, využívá jej však např. také ICMPv6. Najdeme zde zdrojovou a cílovou IPv6 adresu datagramu, 32bitové pole obsahující délku dat na vyšší vrstvě a 8bitovou položku *Next header* doplněnou zleva nulami na celkovou délku 32 bitů. Myslí se tím hodnota toho pole *Next header*, které obsahuje právě identifikátor nejbližší vyšší vrstvy. [17]

2.4 ICMPv6

ICMPv6 je pro IPv6 hlavním podpůrným protokolem, který oproti svému předchůdci navrženému pro IPv4 zastává výrazně větší počet úloh. Kromě servisní role pro nové funkce IPv6, jako je zabezpečení či mobilita uzlů, mu byla naložena také správa skupin (vykonávaná dříve za pomoci IGMP), překlad linkových adres (nejpodobnějším předchůdcem by byl ARP) a bezestavová automatická konfigurace zařízení (mechanismus je v IPv6 nový). [22, 27]

V [3] je uvedeno téměř 30 typů ICMPv6 zpráv. V této kapitole se budu věnovat pouze těm, které patří výhradně do ICMPv6. Další podporované zprávy jsou sice na tomto protokolu založeny, ale užívají se v rámci objevování sousedů a jejich rozbor proto bude vhodnější podat v kontextu souvisejícího *NDP (Neighbor discovery protocol)* v kapitole 2.5.

2.4.1 Základní hlavička

Vzhledem k velmi rozličnému účelu nemají ICMPv6 zprávy téměř žádné společné položky a základní hlavička je tak velmi jednoduchou strukturou, znázorněnou na obrázku 2.5.

Obrázek 2.5: Formát základní hlavičky ICMPv6 zprávy.

V poli *Type* najdeme číslo reprezentující příslušnou zprávu. Nejvýznamnější bit zároveň naznačuje povahu obsahu, kterým může být hlášení o chybě (hodnota `0`) nebo provozní informace (hodnota `1`). Identifikátory běžných informačních zpráv proto v příslušném seznamu najdeme od čísla `128` dále. [3] Pole *Code* může přesněji specifikovat charakter zprávy, ale často se vůbec nevyužívá. Kontrolní součet (*Checksum*) je vypočten nad celou zprávou předřazenou IPv6 pseudo-hlavičkou (kapitola 2.3.7). Za zmíněnými třemi poli již najdeme tělo zprávy. [15, 27, 20]

2.4.2 Hlášení chyb

Knihovna pracuje s jedinou chybovou zprávou, *Parameter problem* (typ `4`), kterou uzel vysílá v případě, že při zpracování příchozího datagramu narazí na logickou chybu (nikoliv tedy např. v důsledku neplatného kontrolního součtu). Problém lze upřesnit použitím jednoho z následujících kódů v základní hlavičce zprávy:

Kód	Význam
`0`	obecná chyba v některé IPv6 hlavičce
`1`	neznámá rozšiřující hlavička
`2`	neznámá volba v rozšiřující hlavičce

Tabulka 2.3: Kódy pro upřesnění významu zprávy *Parameter problem*.

Tělo zprávy začíná 32bitovou položkou *Pointer*, která udává počet bytů od začátku datagramu k místu, na němž byla chyba nalezena. Celý zbytek ICMPv6 zprávy pak tvoří data převzatá z původního datagramu počínaje právě problematickým bytem. Zpráva se zašle uzlu, který chybový datagram odeslal. [15, 20]

2.4.3 Echo (ping)

Nejjednodušší nástroj pro ověření dosažitelnosti určitého uzlu nedoznal v ICMPv6 žádné významné změny. Tazatel vyšle ICMPv6 zprávu *Echo request* (typ `128`) a očekává odpověď v podobě *Echo reply* (typ `129`). Shodný formát zpráv začíná 16bitovými poli *Identifier* a *Sequence number* (pro rozlišení jednotlivých pokusů o zjištění dosažitelnosti) a pokračuje nepovinnou sekvencí libovolných testovacích dat. Má-li být odezva považována za platnou,

musejí se všechna pole v odpovídajícím páru *Echo request* a *Echo reply* zpráv shodovat. [22, 18, 20]

2.5 NDP

Neighbor discovery protocol (NDP) není zcela samostatným protokolem, neboť ke své činnosti využívá podmnožinu zpráv protokolu ICMPv6. Protože však pokrývá komplexnější soubor funkcí v porovnání s hlášením chyb či nástrojem *echo*, má vlastní název a především specifikaci v podobě *RFC 2461: Neighbor Discovery for IP Version 6 (IPv6)*. [25]

NDP poskytuje nástroje pro zjištění stavu místní sítě. Sousedy v názvu protokolu se totiž myslí uzly sdílející s daným zařízením linku, což jsou kromě běžných uzlů obvykle také směrovače. Jejich přítomnost a dostupnost, linkové adresy, prefixy podsítí, životnost datagramů a některé časové proměnné patří mezi informace, která může uzel prostřednictvím NDP získat. [18, 27]

2.5.1 Hledání sousedů

Hledání sousedů je obecnějším mechanismem, který lze použít pro všechny uzly v síti a hlavním předávaným údajem jsou linkové adresy. Zakládá se na ICMPv6 zprávách *Neighbor solicitation* (typ `135`) a *Neighbor advertisement* (typ `136`).

Pomocí první z nich žádá vyzyvatel určitého souseda, aby se mu ohlásil. Jediným povinným parametrem je v tomto případě IPv6 adresa cíle. Vyzyvatel však může připojit volbu typu *Source address*, kterou sdělí cíli svoji linkovou adresu. Cílovou adresou pro odesílaný datagram je skupinová adresa pro vyzývaný uzel, kterou je třeba sestavit pro daný cíl na základě pravidel uvedených v kapitole 2.3.2.

Správnou reakcí na uvedenou výzvu je právě druhý typ zprávy, *Neighbor advertisement*. Vyzývaný soused v ní kromě vlastní IPv6 adresy připojuje povinnou volbu typu *Target address*, ve které sděluje vlastní linkovou adresu. Tento datagram se již posílá na individuální adresu tazatele, který tak získá všechny informace potřebné ke komunikaci s daným sousedem v rámci 2. a 3. vrstvy. Uzel se může rozhodnout vyslat zprávu *Neighbor advertisement* i v případě, že nebyl nikým tázán. Cílovou adresou takového datagramu je pak skupinová adresa pro všechny uzly na dané lince (`ff02::1`). [27]

Informace získané prostřednictvím hledání sousedů by si měl uzel ukládat do *Neighbor cache*, struktury obsahující dvojice linkových a síťových adres pro jednotlivé sousedy. U každé dvojice si navíc zařízení musí uchovávat její stav (*Reachability state*), přičemž *RFC 2461* jich nabízí pět. Ve stavu *Incomplete* není linková adresa známá a zjišťuje se. Záznam *Reachable* značí položku s platnou linkovou adresou, která se po vypršení určité doby změní na *Stale*. V tomto stavu není položky třeba a platnost linkové adresy je proto irelevantní. Jakmile vznikne požadavek na využití záznamu, změní se jeho stav na *Delay* a pokud není v krátké době platnost potvrzena, vyzve uzel souseda k ohlášení a přepne jej na *Probe*. V případě, že je soused dále neaktivní, je položka z *Neighbor cache* odstraněna. Uvedený soubor pravidel se nazývá *Neighbor unreachability detection (NUD)*. [25]

Pro ověření dosažitelnosti souseda, který již je veden v *Neighbor cache*, není třeba opakovaného příjmu zprávy *Neighbor advertisement*. Stav *Reachable* může nastavit či prodloužit vyšší vrstva, pokud získá o dosažitelnosti souseda jasný doklad (jako příklad se uvádí navázání TCP spojení). [25]

2.5.2 Detekce duplicitních adres

Duplicate Address Detection (DAD) je specifickým případem hledání sousedů. Jakmile si uzel vytvoří po připojení do sítě svoji linkovou adresu (kapitola 2.3.2), je povinen si ověřit, že tutéž adresu nevyužívá jiný uzel. Prakticky to provede tak, že se pokusí najít souseda s adresou, kterou si chce sám přidělit. Najde-li takového, prohlásí vytvořenou linkovou adresu za neplatnou, což mu znemožňuje jakýkoliv další provoz v síti. Pokud se naopak žádný uzel neozve, může uzel vygenerovanou linkovou adresu používat a zároveň má možnost si později sestavit adresu globální, kterou už ověřovat nemusí, pokud ji založí na stejném identifikátoru rozhraní. [29]

2.5.3 Hledání směrovačů

Role směrovačů byla v IPv6 posílena o znalost důležitých parametrů místní sítě a právě pomocí NDP je od nich mohou „běžné uzly“ (*hosts*) získat. Slouží k tomu zprávy *Router advertisement* (typ `134`). [25] Obsahují nemalé množství údajů (zejména časových), jichž knihovna využívá, ale jejich popis by byl příliš rozsáhlý.

Soustředím se proto na volbu *Prefix information*, která je klíčová pro automatickou konfiguraci uzlů. Nejvýznamnějším polem je *Prefix*, jehož obsah v kombinaci s hodnotou *Prefix length* a 8bitovým souborem příznaků *LAR* může poskytovat uzlu důležitou informaci. Je-li totiž nastaven bit *L* (v souboru příznaků je první), jde o *lokální prefix*. Zasílá-li tedy zařízení datagram na adresu vyhovující tomuto prefixu, nemusí užívat směrovače, protože cíl je i přes poskytnutou globální individuální adresu dostupný na stejné lince. Je-li navíc nastaven bit *A* (v souboru druhý), lze tento prefix, který nesmí podle *RFC 4291* [19] být delší než 64 bitů, použít pro sestavení globální adresy uzlu v rámci *bezestavové automatické konfigurace* (více v kapitole 2.3.3). Stačí posledních 64 bitů vyplnit vlastním identifikátorem rozhraní. Neposkytuje-li žádný směrovač v síti žádný prefix s příznakem *A*, je uzel odkázán na stavovou konfiguraci. [25, 27, 20]

Routery obvykle zprávy *Router advertisement* vysílají samy na periodické bázi. Uzel o ně však může požádat prostřednictvím zprávy *Router solicitation* (typ `133`). Ta neobsahuje žádná pole kromě základní hlavičky, tazatel může pouze připojit svoji linkovou adresu pro usnadnění komunikace. [22]

Kapitola 3

Hardware

Jádrem mé práce je knihovna v jazyce C. Vzhledem k určité vazbě na použitý hardware však považuji za vhodné podat alespoň základní informace o jeho konfiguraci. Klíčovými částmi systému jsou dva mikrokontroléry (MCU). Hostitelský čip představuje *MSP430F2617* společnosti Texas Instruments (dále MSP), hostovaný čip potom *ENC28J60* společnosti Microchip (dále ENC). [7, 2] Každý z nich je osazen na zvláštní vývojové desce FIT VUT v Brně.

Platforma *FITkit v2.0* hraje v systému roli základní desky, neboť poskytuje zdroj napájení, hostitelský mikrokontrolér, programovatelné hradlové pole (dále FPGA) a USB rozhraní. [10] FPGA není pro činnost systému esenciální, usnadňuje však připojování rozšiřujících modulů, protože jeho vývody míří jak k MSP, tak k pinům vhodným pro zasunutí přídavných desek. Registr, který je v něm v rámci projektu syntetizován, tak vytváří potřebný „můstek“. V případě potřeby by však bylo možné jej nahradit holými vodiči.

Rozšiřující modul pro síť Ethernet, menší desku obsahující především čip ENC a zásuvku RJ45, lze pomocí 20pinového konektoru nasadit přímo na FITkit v místě, kde je vyvedeno napájení, uzemnění a porty FPGA. Vedle koncovky RJ45 najdeme ještě dvě výrazné diody v barvách typických pro zakončení Ethernetu, tedy zelenou a oranžovou.

3.1 Fyzické propojení

V základním provedení pro komunikační kanál mezi MSP a ENC postačuje šest vodičů. Jejich označení a význam uvádím v tabulce 3.1. Můstek syntetizovaný v FPGA je navržen tak, aby uvedenou logiku přesně replikoval a význam i směr signálů tak zůstal zachován.

Signál	Význam	Směr
`SCK`	hodinový signál SPI	MSP → ENC
`MOSI`	přenos dat z řídícího prvku SPI	MSP → ENC
`MISO`	přenos dat do řídícího prvku SPI	MSP ← ENC
`CS`	aktivace řízeného prvku pro komunikaci na SPI	MSP → ENC
`INT`	přerušení	MSP ← ENC
`RST`	reset	MSP → ENC

Tabulka 3.1: Význam signálů ve fyzickém propojení jednotlivých prvků hardware.

3.2 Hostitelský mikrokontrolér MSP430F2617

16bitový RISC čip MSP nabízí 92 kB paměti typu flash a 8 kB paměti RAM, což je díky schopnostem čipu ENC pro potřeby knihovny poměrně dostačující. Většiny speciálních periferií, které MSP nabízí, není potřeba. Kód využívá jednoho časovače a dvou portů, přičemž jeden z nich je konfigurován pro rozhraní `USCI_B` v režimu SPI.

Vzhledem k uvedeným nárokům by nebylo obtížné najít do role hostitelského mikrokontroléru produkt srovnatelných parametrů[1]. Čip MSP tedy není žádným specifickým předpokladem pro správnou funkci knihovny a nebudu se mu podrobněji věnovat.

3.3 Hostovaný mikrokontrolér ENC28J60

Mikrokontrolér ENC je narozdíl od MSP úzce specializovaným čipem, jehož schopnosti knihovna plně využívá. Nabízí totiž široké spektrum funkcí, vztahujících se k fyzické a linkové vrstvě Ethernetu. Výrazně tím šetří prostředky hostitelského MCU, zejména v oblasti RAM.

3.3.1 Organizace paměti

Dvoukanálová paměť RAM s podporou přímého přístupu (DMA) o velikosti 8 kB je určena výhradně Ethernetovým rámcům, přičemž je rozdělena na dvě sekce – pro jejich příjem (*receiver buffer*) a odesílání (*transmit buffer*). Velikost a umístění těchto bufferů lze nastavit podle potřeby.

Konfigurační a stavové údaje jsou rozprostřeny mezi více než stovku nezávislých registrů. Většina z nich se označuje jako *kontrolní* a najdeme je ve čtyřech *bankách*, mezi nimiž je třeba se vhodně přepínat. Všechny mají velikost 1 byte a vyžaduje-li ENC pro nějaký účel 2bytovou hodnotu (např. adresování paměti RAM), je rozdělena do dvou registrů. Zvláštní postavení mají registry související s řízením fyzické vrstvy, tzv. *PHY registry*. Jsou 16bitové, nacházejí se v odděleném segmentu paměti ENC a přístup k nim je ve srovnání s kontrolními registry komplikovanější.

3.3.2 Řízení

MSP komunikuje s ENC s výjimkou signálů `RST` a `INT` (kapitola 3.1) výhradně pomocí rozhraní SPI. Na něm je definováno 7 příkazů, určených v drtivé většině pro práci s paměťovými prvky ENC. Najdeme je tabulce 3.2.

Příkazy pro čtení spočívají v zaslání 8bitového kódu příkazu a následného příjmu dat odpovídající délky. U příkazů pro zápis musí naopak hostitelský mikrokontrolér sám zařadit ihned za 8bitový kód příkazu potřebná data. Metody pro přístup k různým sadám registrů ENC skrývají nejednu výjimku, omezení či specifický požadavek. Detaily najdeme v technické dokumentaci k ENC.

3.3.3 Dostupné funkce

Prvním důležitým mechanismem je odesílání rámců. Do příslušné části paměti ENC hostitelský MCU nejdříve pomocí příkazu blokového zápisu zapíše požadovaný obsah, předřazený jedním kontrolním bytem. Následně sdělí ENC umístění těchto dat a požádá o odeslání.

[1] Mohlo by jít například o PIC24FJ64GA002 společnosti Microchip. [1]

Zkratka	Název	Význam
RCR	Read control register	čtení kontrolního registru
RBM	Read buffer memory	čtení souvislé části paměti RAM
WCR	Write control register	zápis do kontrolního registru
WBM	Write buffer memory	zápis do souvislé části paměti RAM
BFS	Bit field set	nastavení bitu v kontrolním registru
BFC	Bit field clear	vynulování bitu v kontrolním registru
SRC	Software reset command	požadavek na reset čipu

Tabulka 3.2: Seznam příkazů pro ovládání mikrokontroléru ENC přes rozhraní SPI.

Hardware ENC sám dokáže vypočítat a zařadit za rámec kontrolní součet. Po dokončení pokusu o odeslání navíc připojí ještě 6bytový *status vector*, jehož rozborem může hostitelský MCU zjistit, zda byl paket odeslán, proč případně odeslán nebyl a některé doplňující informace.

Příjem rámců probíhá podobným způsobem. ENC opět nabízí šestibytový *status vector*, který je tentokrát umístěn před rámcem. Za ním již najde hostitelský MCU samotná data. Na příchod paketu ENC dokáže upozornit pomocí signálu `INT` či nastavením příslušných příznaků v kontrolních registrech.

Mezi funkce ENC patří dále sada filtrů. Ty umožňují přijímat či zahazovat rámce na základě několika kritérií. Knihovna využívá *Unicast filter* a *Multicast filter*, pomocí nichž zahazuje rámce, jejichž cílová MAC adresa není skupinová a zároveň se neshoduje s vlastní MAC adresou zařízení. Doplňuje je *CRC filtrem*, který brání příjmu rámců s nesprávným kontrolním součtem.

Mezi další využívané schopnosti ENC patří zjišťování stavu fyzické linky, jehož změnu ENC dokáže zařadit mezi zdroje přerušení, a možnost konfigurace připojených LED diod pro nejrůznější účely. Kopírování velkých bloků dat v rámci paměti ENC značně zjednodušuje funkce *DMA copy*, která odlehčuje jak hostitelskému MCU, tak rozhraní SPI.

Kapitola 4

Návrh a implementace knihovny

Knihovna *NET6* je rozdělena na tři logické části, u nichž můžeme rozeznat určitou hierarchii. Na nejnižší úrovni jde o sadu podpůrných funkcí a nastavení. Jedná se převážně o funkce obecného charakteru. Výše bychom mohli postavit balík zpřístupňující především funkce mikrokontroléru ENC. Hlavní část práce pak tvoří kód pracující nad jednotlivými vrstvami *protocol stacku*. Uvedená logická struktura je znázorněna na obrázku 4.1.

Obrázek 4.1: Hierarchická struktura knihovny NET6.

Nezbytným základním kamenem pro celou knihovnu *NET6* je knihovna *libfitkit*, která zpřístupňuje zejména funkce pro komunikaci s terminálem a definice usnadňující práci s čipem MSP. Nad knihovnou naopak pracuje demonstrační aplikace sestávající z hlavní smyčky aplikace, obsluhy přerušení, uživatelského rozhraní a příkladů využití knihovny. Tyto prvky jsou na obrázku 4.1 znázorněny šedou barvou.

4.1 Podpůrný kód

V první části knihovny najdeme zdrojové a hlavičkové soubory s podpůrným kódem.

4.1.1 Základní nastavení

Soubor `base.h` obsahuje především direktivy preprocesoru, jimiž se dá snadno a z jediného místa řídit překlad kódu pro různé cíle. Najdeme mezi nimi přiřazení logických vývodů konkrétním vývodům MSP či makra pro ovládání registru v FPGA, ale také nastavení

rozsahu ladicích výstupů na terminál, MAC adresy či konfiguraci paměti ENC. Několik definic, zejména typových, je určeno pro zvýšení celkové čitelnosti kódu.

V této části bych se rád zmínil o nastavení MAC adresy, jejíž význam se táhne celou knihovnou. Rozšiřující modul není skutečnou, celosvětově unikátní MAC adresou vybaven. Při vývoji reálného produktu by bylo možné použít standardní postup vycházející z kódu výrobce, který však není v případě mé práce aplikovatelný. Knihovna proto pracuje s náhodně zvolenou hodnotou, kterou lze přenastavit právě ve zmíněném souboru.

4.1.2 Tisk ladicích informací

Dvojice souborů `dbg.*` poskytuje nástroje usnadňující zasílání ladicích informací na terminál. Ty mohou být užitečné nejen pro kontrolu správné funkce knihovny či aplikace, ale také pro sledování a rozbor IPv6 provozu v síti.

4.1.3 Kalendář

Vzhledem k vazbě některých síťových protokolů na nejrůznější časové proměnné jsem se rozhodl vytvořit poměrně obecný *kalendář*, který je umístěn v souborech `cal.*`. Tento kód umožňuje vkládání, případné odkládání a především spouštění časově vázaných událostí. Údaje vedené ke jednotlivým záznamům jsou uvedeny na následujícím obrázku:

Obrázek 4.2: Struktura záznamu v kalendáři.

Aktivační čas se skládá se ze dvou částí, přičemž první je uvedena v jednotkách označených jako `2s` a druhá v jednotkách označených `tck`. Ty byly navrženy na základě stavby použitého časovače čipu MSP, který každé 2 sekundy přeteče a během každé takové periody provede určitý počet tiků. Při přetečení je kalendář žádán, aby snížil hodnotu `2s` u všech záznamů o jedničku. Údaj je tedy relativní vzhledem k aktuálnímu času – má-li být událost spuštěna v rámci třetí následující periody, bude mít ve `2s` části aktivačního času hodnotu `3`. Druhá část, tedy tck, naopak udává absolutní čas spuštění události v rámci periody. Má-li čítač hodnotu `200` a událost má být spuštěna po dosažení `500` tiků, bude mít záznam v `tck` části aktivačního času hodnotu `500`. Po provedení jedné události nastaví MSP časovač na další událost v dané periodě, pokud taková v kalendáři existuje. Konstrukce časovače MSP umožňuje oba potřebné momenty, tedy přetečení i navýšení čítače o určitou hodnotu, definovat jako zdroje přerušení. Uvedený formát aktivačního času neubírá kalendáři na jeho obecnosti. Rozhraní kalendáře totiž poskytuje také funkce pro převod údaje v milisekundách, jejichž úprava může odlišný charakter jiného čítače reflektovat.

Úloha, která má být provedena, je vedena jako obecný ukazatel na funkci s jedním 8bitovým parametrem (položka `task`). Ten se při volání funkce naplňuje identifikátorem subjektu, který je také součástí aktivačního záznamu (pole `subj`). Uvedené řešení umožňuje využití služeb kalendáře nejen knihovně, ale i samotné aplikaci.

Kalendář lze charakterizovat jako frontu, jejíž velikost se dynamicky mění. Prvky jsou za sebe navázány pomocí ukazatele v poli `next` přesně tak, jak jdou za sebou jejich aktivační

časy. Jakmile čítač dojde k aktivačnímu času záznamu v hlavičce fronty, provede se daná úloha a záznam je z fronty odstraněn. Výjimkou je situace, kdy bylo záznamu nastaveno *Odložení* (pole `postp_2s`). Nenulová hodnota v tomto poli způsobí, že záznam není spuštěn, ale jeho aktivační čas je namísto toho posunut daný počet period a zároveň je přeřazen na odpovídající ve frontě. Požadavek na tuto funkci je u některých síťových protokolů velmi častý. Odkladu o dobu kratší jedné periodě naopak třeba není a tato funkce proto není podporována, jak je uvedeno v programové dokumentaci.

4.2 Komunikace s rozšiřujícím modulem

Druhou část knihovny tvoří sada funkcí pro ovládání rozšiřujícího modulu. Snadno je odlišíme podle počátečního písmena `e`, odvozeného od označení čipu ENC.

4.2.1 Rozhraní SPI

Soubory `espi.*` umožňují řízení rozhraní SPI. V zásadě tu najdeme funkce pro otevření (inicializaci) a uzavření tohoto komunikačního kanálu na rozhraní `USCI_B` čipu MSP. Síťová část knihovny však prakticky využívá pouze funkci `espi_transmit()`, která provede obousměrný přenos 1 bytu dat mezi MSP a ENC a využívá se tedy jak pro vysílání, tak pro příjem dat.

4.2.2 Rozhraní čipu ENC

Funkce pro obsluhu komunikačního rozhraní mikrokontroléru ENC se nacházejí v souborech `enci.*`. Zpřístupňují všechny příkazy, které ENC na rozhraní SPI nabízí, a navíc usnadňují některé příbuzné operace. Jde zejména o zápis a čtení 16-bitových hodnot v oddělených kontrolních registrech, přepínání bank a také přístup k *PHY registrům* pomocí zvláštního interního rozhraní ENC označeného jako *Media Independent Layer (MII)*.

V této části knihovny se nachází také rozsáhlé definice adres v měřítku bytů i bitů, vztahujících se ke kontrolním registrům. Zatímco bytové adresy jsou uvedeny všechny, bitové jsou vzhledem k jejich celkovému počtu zavedeny jen v případě, že jsou knihovnou *NET6* využívány.

4.2.3 Řízení čipu ENC

Vzhledem k tomu, že mnohé operace potřebné k řízení čipu ENC lze realizovat pouhým nastavením bitu, zápisem nebo čtením některého z paměťových prvků, je soubor rutin umístěný v souborech `enc.*` méně obsáhlý, než by se dalo vzhledem ke schopnostem čipu očekávat.

Největší podíl na objemu kódu v této části mají inicializační funkce. Před samotným příjmem či odesíláním rámců je totiž nutné nastavit nemalé množství detailů. V hlavičkovém souboru najdeme deklaraci funkce `enc_init()`, která tyto kroky voláním příslušných podprocedur sjednocuje. Nejdříve nakonfiguruje LED diody, přičemž zelená signalizuje stav fyzické linky a oranžová aktivitu přijímače. Dále nastaví MAC podvrstvu tak, aby využívala mód *full duplex*, u odesílaných rámců automaticky doplňovala kontrolní součet a odmítala rámce s délkou přes 1518 bytů, které by neodpovídaly standardu Ethernet. Pokračuje rozdělením paměti na *transmit buffer* a *receive buffer*. Po provedení konfigurace filtrů povolí příjem rámců. V závěru inicializuje přerušení tak, aby byl vývod `INT` nastaven do logické nuly v případě, že dojde k příjmu rámce nebo ke změně stavu fyzické linky.

Důležitá je funkce `enc_send()`, která žádá o vyslání rámce připraveného na dané adrese v *transmit bufferu*. Po nastavení příslušných hodnot vloží jedničku do bitu *Ready to send* v kontrolním registru `ECON1` a čeká na jeho uvolnění, aby pokus o odeslání vyhodnotila na základě *status vectoru*. Vyslání rámce obvykle trvá jednotky či desítky cyklů čipu MSP, tedy řádově jednotky milisekund[1]. Z příčiny, kterou se mi bohužel nepodařilo zjistit, ENC někdy setrvává v uvedeném stavu mnohanásobně delší časový interval i v řádu desítek sekund. Funkce `enc_send()` proto obsahuje pojistku, která po určité době nejen informuje o neúspěšném odeslání a pokus ukončí, ale zároveň provede oddělený *reset transmitteru* ENC. Následující pakety jsou pak opět zpracovány bezchybně.

K dispozici jsou dále funkce pro reset mikrokontroléru hardwarovým i softwarovým způsobem, kopírování dat mezi dvěma místy v paměti ENC pomocí funkce *DMA copy* a zjištění aktuálního stavu fyzické linky.

4.3 Komunikace pomocí IPv6

Tato podkapitola sdílí název s titulem práce, neboť je jejím jádrem. Právě ve zdrojových souborech začínajících písmenem `n` (od slova *network*) je rozmístěn kód, umožňující vykonávat funkce potřebné k provozu *protocol stacku* popsaného v kapitole 2.

Zvláštní postavení má dvojice souborů `net6.*`, které obsahují základní funkcionalitu celé knihovny a zároveň poskytují významnou část rozhraní, pomocí něhož mohou aplikace využívat jejích služeb. V tomto bodě zmíním, že na základě upřesnění zadávajícího Ústavu počítačových systémů FIT VUT v Brně neměla být cílem práce knihovna jako samostatný celek, ale jako součást ukázkové aplikace pro cílovou architekturu se zřetelným rozhraním a možnostmi dalšího využití či rozšíření. Právě zmíněné rozhraní najdeme v hlavičkovém souboru `net6.h`. Každá další dvojice `*.c` a `*.h` souborů reprezentuje implementaci jednoho standardu či protokolu.

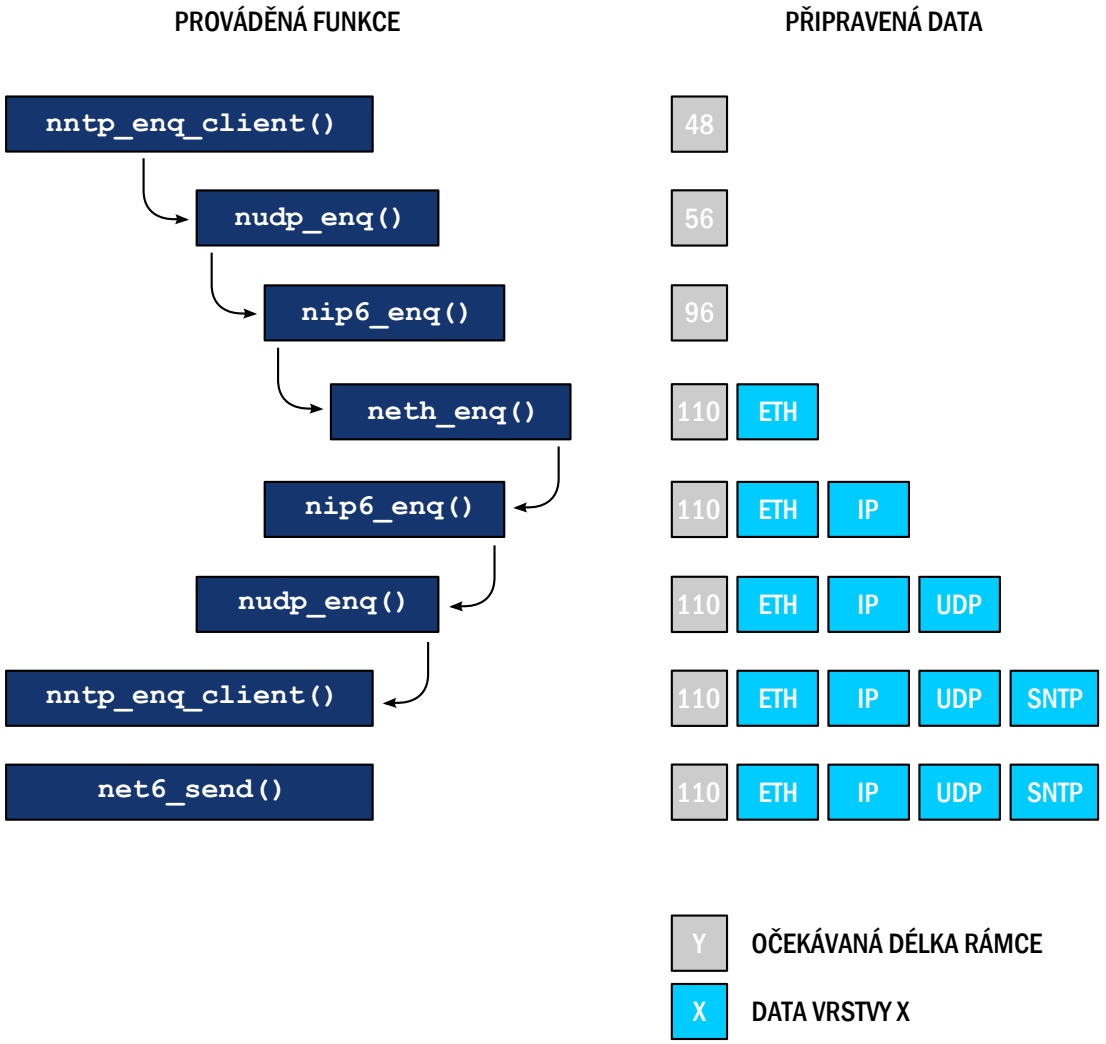
4.3.1 Základní principy

V této kapitole vysvětlím podstatu návrhu, který implementaci síťové části knihovny předcházel. Tím jsem se snažil co nejvíce přiblížit referenčnímu modelu ISO/OSI a zároveň efektivně využít omezené prostředky cílové architektury. Pro udržování základních informací o každém přijímaném i odesílaném paketu slouží zvláštní struktura, kterou v programové dokumentaci označuji jako *metadata*. Neobsahuje totiž samotná data, ale jen jejich vlastnosti – např. umístění v paměti, délku, hodnotu a pozici kontrolního součtu nebo některé příznaky. Tato struktura je využívána napříč celou knihovnou, neboť jednotlivé vrstvy jejím prostřednictvím získávají i předávají informace o změnách, které nad paketem provedly.

Při přípravě rámce k odeslání je v první fázi nutné připravit novou strukturu metadat a vyhradit místo pro data v paměti ENC. Následně se využije funkcí začínajících prefixem ve tvaru `<identifikátor protokolu>_enq`. Identifikátor protokolu je shodný se jménem zdrojového souboru a druhá část vychází ze slova *enqueue*, tedy zařazení do fronty. Tyto funkce nejdříve přičtou délku dat na příslušné vrstvě ke stávající hodnotě v metadatech a následně zavolají podobnou funkci o vrstvu níže. Tímto způsobem se postupuje, dokud se řízení nedostane na nejnižší použitou vrstvu (Ethernet u běžných rámců a IPv6 u paketů vysílaných pomocí lokální smyčky). Potom se program opět vrací do funkcí vyšších úrovní a tentokrát už zapisuje do paměti odpovídající obsah na příslušných vrstvách. Uvedenou

[1] Instrukční cyklus čipu MSP je 125 ns. [10]

metodu jsem zvolil s ohledem na potřebu sekvenčního zápisu, který tak může být prováděn přímo do paměti ENC.

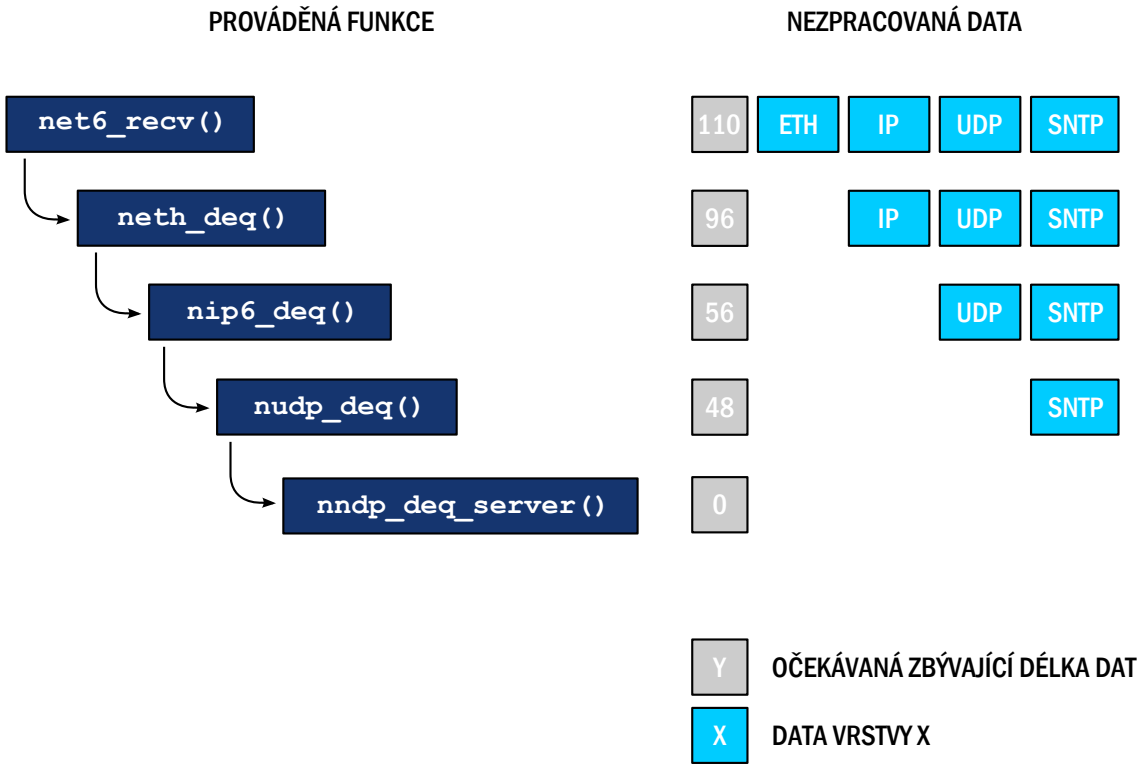


Obrázek 4.3: Příprava rámce k odeslání.

Jakmile je rámec připraven, lze jej předat funkci `net6_send()`, která jej zařadí do fronty paketů k odeslání (*transmitter queue*). Nebrání-li nic okamžitému odeslání, nastaví potřebné údaje v mikrokontroléru ENC. Dojde-li následně k chybě hardware ENC, je pokus o odeslání ještě jednou zopakován. V každém případě má aplikace možnost zjistit výsledek funkce `net6_send()` buď na základě návratové hodnoty, nebo pomocí globální proměnné `last_tx_err`. Druhý způsob byl navržen pro případy, kdy je funkce spouštěna v rámci makra. Doplňující informace lze nalézt mezi příznaky metadat. Před každým dalším pokusem o odeslání paketu jsou z pozice hlavičky fronty paketů odstraněny všechny, které byly buď úspěšně odeslány, nebo se u nich vyskytla neřešitelná chyba. Příklad reprezentující přípravu a odesílání rámce obsahujícího zprávu *Client* protokolu SNTP je uveden na obrázku 4.3.

Při příjmu paketů je postup do jisté míry podobný. Na základě přerušení od rozšiřují-

cího modulu se nejprve zavolá funkce `net6_recv`, která inicializuje strukturu metadat. Je-li rámec na základě příslušných bitů *status vectoru* rozpoznán jako nesprávně přijatý nebo broadcastový (tedy nesoucí IPv4), je již v tomto místě zahozen. V opačném případě se následně spustí posloupnost funkcí s prefixem ve tvaru `<identifikátor protokolu>_deq` (od slova *dequeue*, odebrání z fronty). Ty čtou a zpracovávají data na jednotlivých vrstvách, přičemž aktualizují příslušné části metadat. Příklad zpracování rámce při příjmu najdeme na obrázku 4.4.



Obrázek 4.4: Zpracování rámce při příjmu.

Specifická je obsluha paketů zasílaných přes lokální smyčku (loopback). Při jejich přípravě se nezařazují data linkové vrstvy a při odesílání se namísto žádosti čipu ENC nastaví globální příznak `loopb_int`. Ten je při nejbližším cyklu hlavní smyčky vyhodnocen podobně jako např. přerušení při příjmu skutečného paketu a vede k volání zvláštní funkce `net6_recv_loopb()`, která po aktualizaci metadat předá data ke zpracování IPv6 vrstvě stejně, jako by šlo o běžný paket.

4.3.2 Rozsah implementace

IPv6 a přidružené protokoly jsou velmi rozsáhlé a implementace všech nabízených funkcí by byla obtížná nejen z hlediska objemu práce, ale také značně omezených zdrojů. Pro snazší orientaci v odpovídajících dokumentech vydala IETF souhrnné *RFC 4294: IPv6 Node Requirements*, které přehledně s pomocí odkazů na jiná RFC definuje minimální požadavky kladené na uzly podporující IPv6. [13] Tento dokument jsem využil k rozvaze nad rozsahem

implementace, díky čemuž knihovna NET6 téměř všechny nároky splňuje. Přehled funkcí a jejich podporu shrnuje tabulka 4.1.

Požadavek	**Kardinalita**	**Plnění**
podpora IPv6 protokolu podle RFC 2460	povinný	✓
zpracování neznámých rozšiřujících hlaviček a voleb podle RFC 2460	povinný	✓
podpora rozšiřujících hlaviček *Hop-by-hop options*, *Destination options*, *Type 0 routing header*	povinný	✓
podpora rozšiřujících hlaviček *Fragmentation*, *Authentication header*, *Encapsulating security payload*	povinný	✗
implementace pravidel pro výběr adres podle RFC 3484	povinný	✓
podpora linkového standardu upraveného pro IPv6	povinný	✓
podpora *Neighbor discovery protocol (NDP)*	nepovinný	✓
podpora *Neighbor unreachability detection (NUD)*	povinný	✓
podpora bezestavové automatické konfigurace	povinný	✓
podpora *Duplicate address detection (DAD)*	povinný	✓
podpora hledání směrovačů	povinný	✓
podpora ICMPv6 zpráv *Redirect*	nepovinný	✗
podpora *Path MTU discovery*	nepovinný	✗

Tabulka 4.1: Přehled požadavků na IPv6 uzly a jejich implementace v knihovně *NET6*.

Vynechání podpory fragmentace, autentizace a šifrování paketů bylo dohodnuto již při upřesňování zadání s ohledem na značně omezené prostředky cílové architektury a rozsah práce. I bez ní by však neměla být použitelnost zařízení ve většině sítí výrazně omezena. K fragmentaci se v [27] uvádí, že lze očekávat její vzácné použití vzhledem k tomu, že IPv6 požaduje infrastrukturu s minimální hodnotou *Maximal transmission unit (MTU)* 1280 bytů. Povinná podpora fragmentace se přitom podle *RFC 2460* [17] týká pouze paketů o původní velikosti do 1500 bytů. O zabezpečení IPSec se zase v [27] píše, že v mnoha implementacích chybí. To potvrzuje i skutečnost, že není zahrnuto ve stacku *uIPv6* společností Cisco, Atmel a institutu SICS určeného pro vestavné systémy. Autoři se odvolávají na to, že povinnost implementace IPSec má být do budoucna zrušena. [12]

4.3.3 Ethernet

Soubory `neth.*` určené linkové vrstvě patří k nejkratším. Podíl na tom mají jak hardwarové schopnosti ENC, tak jednoduchá a pevná struktura Ethernetového rámce. Při sestavování rámců je zařazena adresa cíle nebo šest nulových bytů v případě její nedostupnosti, vlastní adresa a konstanta `86DD` hexadecimálně informující o IPv6 jako vyšší vrstvě. Při rozboru je naopak ověřováno, že cílová MAC adresa je buď individuální (porovnání s vlastní adresou zařízení provádí *Unicast filter* na ENC) nebo skupinová vyhovující IPv6. V opačném případě není rámec tomuto zařízení určen. Je-li v poli *Type* nalezena zmíněná konstanta reprezentující IPv6, předá se řízení vyšší vrstvě, jinak není rámec dále zpracováván.

4.3.4 IPv6

V souborech `nip6.*` najdeme funkce obsluhující samotné jádro IPv6. Na této úrovni jsou také definovány struktury reprezentující *Neighbor cache* a *Prefix list*, které jsou sice plněny a aktualizovány především pomocí NDP, ale právě IPv6 vrstva jich nejvíce využívá.

Neighbor cache je v kódu začleněna jako statická struktura `neigc`. Obsahuje stálý počet položek daný konstantou `NEIGC_SIZE`, přičemž první z nich je vyhrazena bráně. Ostatní se plní a případně přepisují jednoduše podle indexu v rámci struktury. Kromě dvou základních údajů o sousedovi – linkové a síťové adresy – se uchovává také stav záznamu (pole `status`), počet pokusů o získání linkové adresy (`retry_cnt`) a ukazatel na záznam v kalendáři, který se spustí po vypršení časového limitu (`tim`). Je-li totiž informace o sousedovi aktualizována, může být akce v kalendáři odložena či zrušena. Znázornění záznamu v *Neighbor cache* je uvedeno na obrázku 4.5.

Obrázek 4.5: Struktura záznamu v *Neighbor cache*.

Podoba *Prefix listu* je o něco jednodušší. Struktura `prefl` je i v tomto případě statická. Její položky však obsahují pouze samotný prefix a ukazatel na záznam v kalendáři, který ho po uplynutí určité doby zneplatní. I tuto akci lze odložit, dostane-li *NET6* aktualizaci v podobě zprávy *Router advertisement*.

Sestavování paketů na úrovni IP vrstvy zahrnuje hledání vhodné linkové adresy, což je hlavním úkolem funkce `nip6_get_eth_addr()`. Tato funkce dále vyhodnocuje příbuzné informace, jako je dosažitelnost cíle v rámci linky, zdrojová IPv6 adresa (mezi povinnými adresami uzlu jsou hned 3 kandidáti) nebo rozhraní, které se má použít (kromě rozšiřujícího modulu se nabízí lokální smyčka). Výsledkem může být i zjištění, že linková adresa k dispozici není. Jde-li o souseda, bude na základě příznaku `ETH_DST_ADDR_NA` v metadatech paketu proveden funkcí `net6_send()` pokus o její získání. Zbývajícím případem je možnost, že uzel se na lince nenachází a zároveň není známa žádná brána – pak je datagram prohlášen za nedoručitelný. Nedojde-li k chybě, pokračuje program vhodnou rutinou pro zařazení dat nižší vrstvy. Po jejím dokončení zařadí základní IPv6 hlavičku, přičemž pole související s toky a QoS jsou vyplněny nulami.

Rozbor paketů začíná načtením údajů ze základní hlavičky. Následuje kontrola, zda cílová IPv6 adresa paketu patří do souboru těch, na nichž uzel naslouchá. Dalším krokem je již zpracování rozšiřujících hlaviček. Obecnější hlavičky *Destination options* a *Hop-by-hop options* jsou zpracovávány jednotně, přičemž v některých případech může být paket na základě obsažených voleb okamžitě zahozen. *RFC 2460* dále ukládá zvláštní postup při zpracování hlavičky *Routing* s neznámou hodnotou v poli *Typ*. To jsou v případě knihovny *NET6* prakticky všechny, protože s jediným původně povinným typem `0` nařizuje *RFC 5095* po jeho zamítnutí zacházet stejně. [11] Postup závisí na hodnotě pole *Segments left*: je-li nulová, lze pokračovat následující hlavičkou, ve všech ostatních případech je však datagram třeba zahodit. [17] Stejně tak je zpracování ukončeno, nezná-li knihovna kód uvedený v poli *Další hlavička* základní či libovolné rozšiřující hlavičky. Ať už je datagram zahozen z jakéhokoliv důvodu, je třeba zaslat odesílateli informační ICMPv6 zprávu *Parameter problem*.

V ideálním případě je však řízení předáno vyšší vrstvě, jakmile je zpracována poslední IPv6 hlavička.

Některé vrstvy pracující nad IPv6 využívají mechanismus kontrolního součtu, do něhož se počítá kromě samotné zprávy vyšší vrstvy také tzv. *IPv6 pseudo-header* (kapitola 2.3.7). Tento set údajů proto IPv6 vrstva zahrnuje do položky `chksum` v metadatech při odesílání i příjmu datagramů a vyšší vrstvy se touto úlohou nemusí zabývat.

4.3.5 ICMPv6

Obsluha ICMPv6 umístěná v souborech `nicmp6.*` se od předcházejících částí liší tím, že tento protokol není nosný, ale obsah jeho zpráv již představuje samotnou přenášenou informaci. Nenajdeme zde proto pouze jedinou dvojici funkcí, která by zajišťovala celé rozhraní pro nižší i vyšší vrstvy. Namísto toho je k dispozici sada rutin se jmény ve tvaru `nicmp6_enq_<typ zprávy>`, které sestavují příslušný typ ICMPv6 zpráv. Zvláštní funkce `nicmp6_enq_hdr()` pak slouží k zařazení jednotné 4bytové hlavičky. Ta mimo jiné obsahuje pole pro kontrolní součet, které je nejdříve vyplněno nulami a po sestavení těla zprávy doplněno skutečným CRC.

Na straně příjmu má sice IP vrstva k dispozici univerzální funkci `nicmp6_deq()`, ale ta jen zpracuje společnou hlavičku, ověří kontrolní součet a na základě kódu zprávy předá řízení odpovídající funkci se jménem ve tvaru `nicmp6_deq_<typ zprávy>`. NET6 pak volí další postup na základě pravidel uvedených v tabulce 4.2.

Příchozí zpráva	**Akce**
Echo request	odpověď v podobě zprávy *Echo reply*
Echo reply	předání údajů demonstrační aplikaci
Neighbor solicitation	odpověď v podobě zprávy *Neighbor advertisement*
Neighbor advertisement	aktualizace údajů v *Neighbor cache*
Router advertisement	aktualizace struktur automatické konfigurace

Tabulka 4.2: Volba akce na základě typu přijaté ICMPv6 zprávy.

Stojí za zmínku, že právě ICMPv6 vrstva využívá funkci *DMA copy* na čipu ENC, když kopíruje část příchozí zprávy do nově vytvářené odpovědi. Tato technika se využívá u pole *Optional data* v případě sestavování *Echo reply* a u pole *Original IPv6 Datagram Portion* při tvorbě zprávy *Parameter problem*.

4.3.6 NDP

Některé kroky související s NDP jsou pro jednoduchost vykonávány již v rámci ICMPv6 vrstvy současně se zpracováváním příslušné zprávy. Obsah souborů `nndp.*` je proto vázán na složitější mechanismy, které jsou obvykle řízeny pomocí kalendáře (kapitola 4.1.3). Funkce `nndp_*()` tak obvykle slouží jako *callback* u odpovídajících kalendářových záznamů.

Hlavní pole působnosti této sekce leží v *bezestavové automatické konfiguraci* (kapitola 2.3.3). Struktura `aconf` obsahuje množství souvisejících údajů, nejdůležitější je však položka `status` určující aktuální *fázi automatické konfigurace*. Diagram celého procesu včetně návaznosti na kód prezentuji na obrázku 4.6.

Nejprve si uzel sestaví linkovou adresu a pomocí DAD ověří, že je v rámci linky unikátní. V praxi to znamená zaslání tří zpráv *Neighbor solicitation* a následné vyhodnocení případných odpovědí. Dalším krokem je odeslání zprávy *Router solicitation*, aby uzel nemusel čekat, až se směrovač ohlásí sám. Nepodaří-li se informace o směrovači získat, naplánuje *NET6* do kalendáře s určitým odstupem nový pokus a zatím umožní jen komunikaci v rámci linky. Na základě příjmu vhodné zprávy *Router advertisement* – tedy takové, která bude obsahovat alespoň jeden prefix s příznakem *A* (více v kapitole 2.5.3) – si pak uzel vytvoří také globální adresu a bude tedy schopen navazovat spojení do Internetu. Globální adresa se po určité době bez aktualizace změní na *odloženou (deferred)* a později na *neplatnou* – v té chvíli se *NET6* vrátí do stavu, v němž disponuje pouze linkovou adresou. Podobné lhůty se sledují u lokálních prefixů (příznak *L*), kde se však rozlišují pouze stavy „platný“ a „neplatný“.

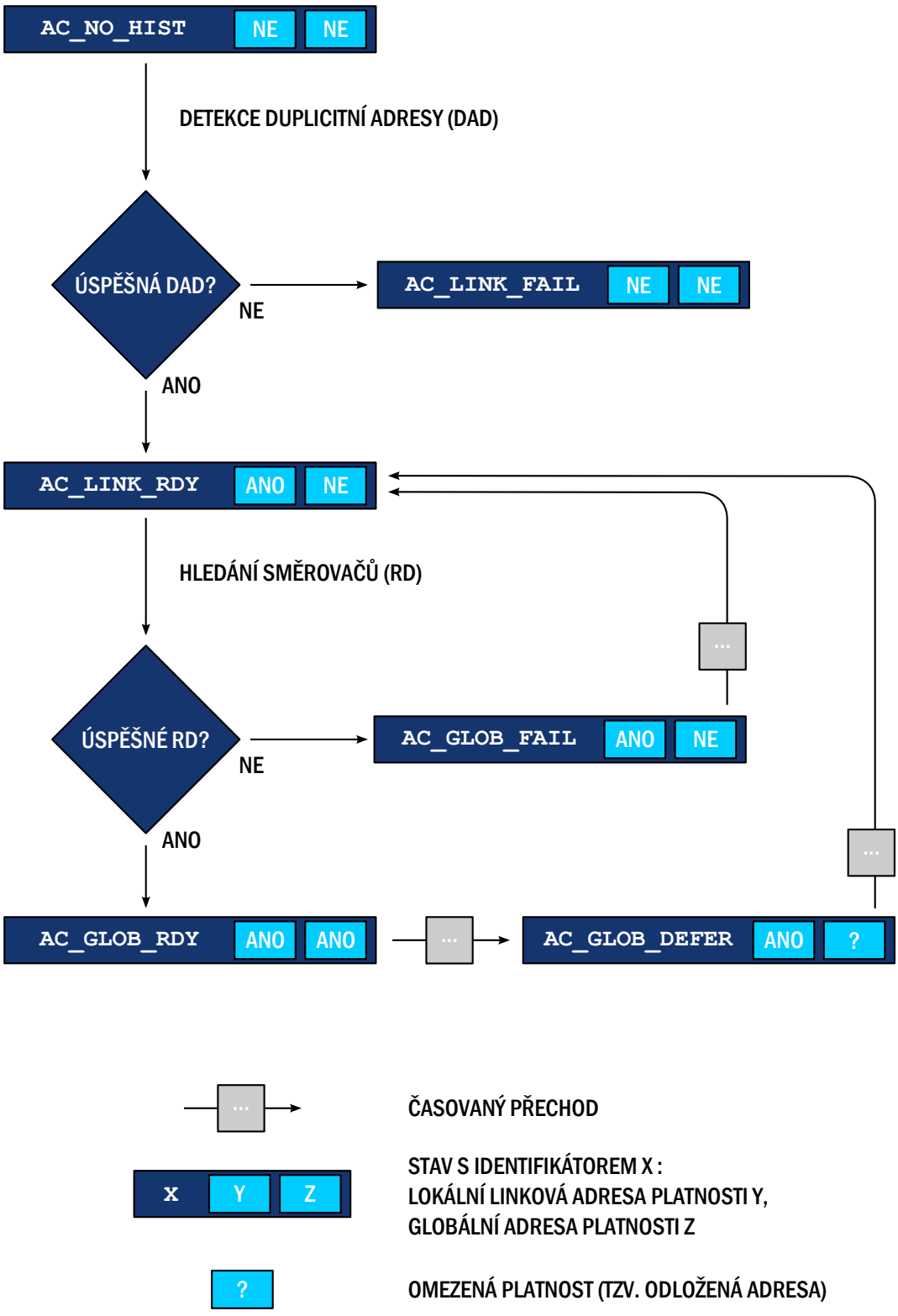
Kromě automatické konfigurace patří do této části knihovny také hledání sousedů včetně NUD, které spočívá v aktualizaci příslušných záznamů ve struktuře `neigc` (kapitola 4.3.4). Podobně jako u DAD provádí *NET6* nejdříve tři pokusy o zjištění linkové adresy souseda, aby následně dále vyhodnocovalo jeho dosažitelnost podle pravidel uvedených v teoretické části (kapitola 2.5.3).

4.3.7 UDP a SNTP

Implementace transportního protokolu *User Datagram Protocol (UDP)* a protokolu *Simple Network Time Protocol (SNTP)* pro zjištění aktuálního času byla do práce zařazena pouze za účelem demonstrace schopností knihovny. Pro zachování systému uspořádání zdrojových souborů však byly příslušné funkce rozmístěny do souborů `nudp.*` a `nntp.*`.

Z pohledu zápisu a čtení dat je obsluha UDP kompletní. Všechna pole včetně kontrolního součtu, který je u UDP v kombinaci s IPv6 povinný, jsou korektně zpracovávána. Vrstva si ale neudržuje žádné informace o používaných portech, při sestavování paketů je tedy nutné jí tyto údaje předat a pro účely rozboru paketů je třeba konkrétním dvojicím portů přiřadit konkrétní funkce vyšších vrstev. Pro jednoduché aplikace, jako je právě SNTP, je to však zcela postačující. Používá se u ní výchozí číslo zdrojového i cílového portu `123`. [5]

Obdobně minimalistické jsou funkce související s SNTP. Funkce `nntp_enq_client()` umožňuje sestavit zprávu typu *Client*, kterou se může uzel SNTP serveru dotázat na aktuální čas. Většina použitých polí je prázdná nebo obsahuje vhodně zvolené konstanty. Funkce `nntp_deq_server()` dokáže zpracovat zprávu typu *Server*, kterou SNTP server zasílá několik údajů, na jejichž základě lze stanovit přesný čas. [24] Demonstrace však využívá pouze údaje *Transmit timestamp*, vyjadřujícího aktuální UTC čas serveru ve chvíli odesílání zprávy. Čas získaný pomocí této funkce se tak může lišit od skutečného o dobu, kterou paket putoval sítí.



Obrázek 4.6: Posloupnost a význam jednotlivých fází automatické konfigurace.

Kapitola 5

Nasazení v praxi

Tato kapitola obsahuje informace vztahující se k praktickému využití celého řešení. Popisuje vhodné síťové prostředí pro jeho nasazení, dostupné demonstrační aplikace i návod k využití funkcí knihovny *NET6* pro vlastní účely.

5.1 Testovací prostředí

Mají-li být dostupné funkce smysluplně využity, je třeba připojit FITkit s rozšiřujícím modulem do sítě typu Ethernet, v níž se se nachází alespoň jeden další uzel schopný komunikace pomocí IPv6. Pro plnohodnotné nasazení je však vhodné mít v cílové síti také směrovač s připojením do Internetu na bázi IPv6.

Přímou IPv6 konektivitu nabízí ke 14. 4. 2010 v České republice přibližně každý třetí poskytovatel připojení k Internetu (ISP). Její skutečná dostupnost je však nízká, neboť největší ISP obvykle patří do skupiny, která nový protokol nepodporuje a navíc o jeho nasazení v nejbližší době ani neuvažuje. [8, 6] V mnohých zemích je však situace podobná a ke slovu tak přicházejí společnosti s celosvětovou působností, které nabízejí IPv6 konektivitu ve formě tunelů, které se někdy neoficiálně označují jako *6in4* nebo *IPv6-in-IPv4*. [27, 9]

Tunely spojují koncovou síť s jednou z tzv. bran, přičemž IPv6 datagram je na této části trasy „zabalen“ do běžného IPv4 datagramu a teprve od brány dál pokračuje dál do světa IPv6 Internetu běžným způsobem. Kromě tunelu získá směrovač koncové sítě k dispozici také prefix, který může nabídnout ostatním uzlům pro konfiguraci jejich vlastních globálních adres. Daní za použití tunelu je především určitá časová režie pro přenos datagramu tunelem. Mezi poskytovatele *6in4* tunelů patří například společnosti Hurricane Electric a Sixxs. [21]

Počítače vybavené novějšími verzemi rozšířených operačních systémů (OS) lze zapojit do sítě v roli IPv6 uzlu, aniž by bylo třeba zvláštní konfigurace. Do této skupiny OS patří například Microsoft Windows verze 7 a Vista, Microsoft Server počínaje verzí 2003, distribuce Linuxu založené na jádře 2.6.12 a vyšším či FreeBSD od verze 6.1 výše. V systému Microsoft Windows XP lze podporu IPv6 poměrně snadno aktivovat. [27] OS na bázi UNIXu jsou v souvislosti s IPv6 obvykle schopny hrát navíc roli směrovače nebo serveru, přičemž ani v tomto případě není postup při konfiguraci nikterak složitý. [21, 9]

V síti použité při vývoji a testování knihovny *NET6* byly trvale zapojeny dva uzly s podporou IPv6 (OS Microsoft Windows 7 a Microsoft Windows XP) a jeden směrovač (OS Ubuntu 9.10) s IPv6 konektivitou, získanou prostřednictvím brány společnosti Hurricane Electric umístěné ve Frankfurtu. Provoz v rámci Ethernetové linky zároveň vytvářely další

tři uzly pracující nad IPv4, přičemž jeden z nich generoval skupinové vysílání (multicast), a jeden směrovač s přímou IPv4 konektivitou. Tato konfigurace vystavovala knihovnu NET6 rozmanitým druhům Ethernetových rámců nesoucích skutečnou komunikaci nad IPv4 i IPv6.

Nastavení směrovače potřebné pro zprovoznění tunelu, samotného směrování a rozesílání zpráv *Router advertisement* na použitém OS je popsáno v dodatku B.

5.2 Demonstrační aplikace

Program umístěný v souboru `main.c` vytváří na základě knihoven *libfitkit* a *NET6* funkční celek. Po inicializaci hardware a jednotlivých částí knihoven vstupuje řízení do nekonečné smyčky, v níž se vyhodnocují příznaky hardwarových přerušení, příznak lokální smyčky `loopb_int` a aktivita terminálu. Zdroje přerušení jsou z pohledu hostitelského mikrokontroléru MSP tři: přetečení čítače, uplynutí nastaveného počtu tiků na čítači a přerušení na univerzálním portu přicházející od ENC. V případě posledně jmenovaného se pomocí rozhraní ENC dále zjišťuje, zda byl podnětem příchod rámce (pak je zahájen jeho rozbor) nebo změna stavu fyzické linky (pak dojde k resetu většiny struktur plněných pomocí NDP, aby mohly být přizpůsobeny nové síti).

V rámci demonstrační aplikace nabízím několik způsobů, jak ověřit funkčnost knihovny. Některé lze aktivovat nastavením vhodných přepínačů umístěných v souboru `base.h` při překladu aplikace, většina je však přístupná za pomoci příkazů zasílaných prostřednictvím terminálu.

Nejrozsáhlejším dokladem o způsobu zpracování příchozích dat je možnost přehledných výpisů přijatých datagramů na terminál. Předpokladem pro jejich aktivaci je nastavení direktivy preprocesoru `NET6_DBG` v souboru `base.h`. Výpisy z jednotlivých vrstev pak lze vypnout či zapnout pomocí příslušné konstanty s názvem ve tvaru `<zkratka protokolu>_DBG_PRINT`, přičemž neobvyklá zkratka `NHW` se vztahuje k výpisu údajů poskytovaných hardwarem ENC, jako je pozice rámce v paměti nebo *status vector*.

Seznam funkcí dostupných prostřednictvím terminálu se zobrazí po zadání příkazu `HELP`. Jejich stručný seznam najdeme v tabulce 5.1.

Příkaz	**Význam**
`HELP`	zobrazí nápovědu k příkazům terminálu
`STAT`	zobrazí základní informace o stavu zařízení
`TXQ`	zobrazí frontu rámců k odeslání
`NC`	zobrazí obsah *Neighbor cache*
`RES`	provede softwarový reset čipu ENC
`TG`	zobrazí adresu cíle
`MTG X Y`	nastaví adresu cíle na základě ručního zadání X je hexadecimální číslice udávající offset v rámci adresy cíle Y je sekvence hexadecimálních číslic k zápisu
`PTG X`	nastaví adresu cíle na základě seznamu předdefinovaných cílů X je index záznamu, který má být použit

Tabulka 5.1: Přehled příkazů terminálu.

První skupinu tvoří příkazy pro zjištění stavu zařízení. Příkaz `STAT` poskytuje základní informace, jako je dostupnost fyzické linky nebo fáze automatické konfigurace. Pomocí příkazu `TXQ` můžeme nahlédnout do fronty rámců k odeslání. Výsledek práce nejvýznamnější části knihovny zpřístupňuje příkaz `NC`, který vypíše *Neighbor cache*, přičemž první položka označená zkratkou `GW` na začátku řádku nereprezentuje běžného souseda, ale výchozí směrovač (bránu).

Osamocený příkaz `RES` se týká hardware ENC – čip je jeho prostřednictvím požádán o softwarový reset a vzápětí je znovu inicializován. Všechna konfigurace samotné knihovny zůstává však při provádění tohoto příkazu zachována.

Poslední skupina slouží konkrétním demonstračním aplikacím. Vzhledem k délce IPv6 adres, jejichž zápis přesahuje maximální délku příkazu na terminálu FITkitu, jsem vytvořil strukturu `targ` obsahující adresu aktuálního cíle využívaného pro terminálové aplikace. Obsah této struktury je možné vypsat pomocí příkazu `TG`. Sekvence `MTG` (*manual target*) pak slouží k manuálnímu zadání adresy, což je vzhledem k uvedenému omezení nutné provést obvykle přinejmenším ve dvou krocích. Pokud bychom tedy chtěli zadat například IPv6 adresu `2001:0718:0802:0809::93E5:0917` (získanou překladem doménového jména `www.fit.vutbr.cz`), postupovali bychom následovně:

```
// zapisujeme 8 bytů s offsetem 0, tedy první polovinu adresy cíle
mtg 0 2001071808020809
// zapisujeme 8 bytů s offsetem 8, tedy druhou polovinu adresy cíle
mtg 8 0000000093E50917
```

Server FIT VUT v Brně se však řadí mezi tzv. *předdefinované cíle*, kterými lze strukturu `targ` naplnit podstatně pohodlnějším způsobem – pomocí příkazu `PTG` (*predefined target*). V tomto případě by šlo o tuto krátkou sekvenci:

```
// nastavujeme adresu s indexem 1 ze seznamu předdefinovaných cílů
ptg 1
```

Fakulta je totiž v seznamu předdefinovaných cílů zařazena na pozici `1`. Výchozí nastavení seznamu předdefinovaých cílů najdeme v tabulce 5.2.

Jakmile si vybereme vhodný cíl, můžeme vyzkoušet příkaz `PING`. Ten představuje jednoduchý nástroj pro zjištění dosažitelnosti cíle, jemuž je zaslána zpráva *Echo request*. Na základě přijetí odpovědi ve formě zprávy *Echo reply* nebo vypršení přiděleného časového intervalu je vypsána odpovídající zpráva do terminálu. Související možností je také opačná situace, kdy knihovna vhodně odpovídá na příchozí zprávy *Echo request*. Tento test je však

Pozice	Cíl	SNTP server
0	lokální smyčka	✗
1	FIT VUT v Brně	✓
2	ICM Varšavské univerzity	✓

Tabulka 5.2: Výchozí nastavení seznamu předdefinovaných cílů.

třeba ovládat ze vzdáleného stroje například pomocí příkazu `ping -6` (systémy Microsoft Windows) nebo `ping6` (UNIXové systémy).

Zbývající příkaz `TIME` se chová podobně s tím rozdílem, že cílem musí být NTP server a na základě odpovědi není na terminálu zobrazena pouze informace o příjmu, ale také časová známka, kterou NTP server do odpovědi umístil při jejím odesílání a odpovídá tedy aktuálnímu času v rámci možností implementace (podrobnosti v kapitole 4.3.7). Oproti údaji získanému výpisem rozboru SNTP zprávy je tato časová známka převedena do člověku srozumitelné podoby ve formátu obvyklém pro Českou republiku.

5.3 Vlastní využití

Ačkoliv s sebou projekt přináší možnosti samostatného využití, měla by knihovna být připravena sloužit především jako nezbytný základ pro konkrétní aplikace vyžadující síťové připojení na bázi IPv6. Potřebný návod poskytnu právě v této kapitole.

5.3.1 Podpůrné mechanismy

Mnohým síťovým aplikacím mohou přijít vhod podpůrné struktury, které využívá i knihovna samotná. Je to například stav automatické konfigurace udržovaný ve struktuře `aconf` nebo informace o stavu fyzické linky dostupná pomocí funkce `enc_on_link()`. Vzhledem k časté vazbě na časové intervaly se mohou hodit také funkce kalendáře `cal_<>()`, dostupnost sousedů lze ověřit náhledem do *Neighbor cache* uložené ve struktuře `neigc`. Podrobnosti lze najít v programové dokumentaci projektu.

5.3.2 Odesílání paketů

Jak jsem uvedl v příslušných kapitolách, pro odeslání datagramu pomocí knihovny *NET6* je třeba určité inicializace, zápis odpovídajících dat do paměti ENC a samotný požadavek na odeslání. Pro zjednodušení tohoto standardního postupu nabízí knihovna makro `net6_disp()`, jehož čtyři parametry jsou uvedeny v tabulce 5.3.

Výsledek funkce `net6_send()` je po dokončení makra k dispozici v globální proměnné `last_tx_res`. Je-li třeba, může aplikace samozřejmě realizovat jednotlivé kroky ve vlastní režii. V takovém případě může zmíněný výsledek číst obvyklým způsobem jako návratovou hodnotu.

5.3.3 Příjem paketů

Doporučeným postupem pro příjem paketů je volání příslušné funkce aplikace z nejvyšší vrstvy, která je ještě zajišťována knihovnou *NET6*. Má-li tedy aplikace pracovat např. nad

Parametr	Význam
`meta`	struktura metadat pro odesílaný rámec
`size`	největší možná délka rámce v bytech
`addr`	IPv6 adresa cíle
`fn`	funkce, která má zahájit sestavování rámce na nejvyšší vrstvě

Tabulka 5.3: Přehled parametrů makra `net6_disp()` určeného k odesílání datagramů.

protokolem UDP, bude volající funkcí `nudp_deq()`. Po vstupu do obslužné funkce aplikace je ukazatel pro čtení z paměti ENC (`ERDPT`) nastaven na místo, kde začínají data náležející aplikaci. Jejich zbývající délku v přijatém rámci pod názvem `payld_len` najde aplikace stejně jako další důležité informace v položkách metadat. Tuto strukturu je proto vhodné obslužné funkci předat.

5.3.4 Případová studie – ping

Postup zmíněný v předcházejících podkapitolách doprovodím názorným příkladem nástroje *ping*, který je dostupný v rámci demonstrační aplikace. Na základě příkazu terminálu se volá funkce, jejíž krátký kód zde uvádím:

```
void app_ping_send() {
  // resetujeme příznak dokončení operace
  app_ping_fin = F;
  // sestavíme a odešleme paket
  net6_disp(tx_meta, NICMP6_ECHO_REQ_LEN, targ,
            nicmp6_enq_echo_req(&tx_meta, 0, 0));
  // nastavíme akci po vypršení časového intervalu
  cal_add(0, 0x8000, T, &app_ping_timeout);
}
```

Ta resetuje příznak dokončení operace, odešle paket pomocí makra `net6_disp()` a do kalendáře naplánuje prostřednictvím funkce `cal_add()` akci, která bude spuštěna po vypršení časového intervalu. Do funkce `nicmp6_deq_echo_reply()` bylo doplněno volání obslužné funkce `app_ping_recv()`, které jsou kromě metadat předány také další důležité údaje z ICMPv6 vrstvy. Tam poté dojde k vyhodnocení a případnému nastavení příznaku dokončení operace.

Třetí funkcí využívanou v této aplikaci je `app_ping_timeout()`, která se projeví pouze tehdy, pokud před jejím voláním nebyla operace úspěšně dokončena. V takovém případě se totiž cíl považuje za nedosažitelný a `app_ping_timeout()` zašle negativní hlášení do terminálu.

Kapitola 6

Závěr

Předmětem mé práce bylo vytvořit knihovnu umožňující síťovou komunikaci pomocí IPv6 a přidružených protokolů. Cílovou platformou byla vývojová deska FITkit doplněná o rozšiřující modul pro připojení do sítě typu Ethernet. Vzhledem k použitému jazyku ANSI C a struktuře kódu by však mělo být možné ji poměrně snadno upravit i pro jiný hardware, který by využíval mikrokontroléru ENC28J60.

Knihovna podporuje protokoly IPv6, ICMPv6, NDP, UDP, SNTP a částečně vykonává také práci nad standardem Ethernet. V kombinaci s hardwarovým zázemím čipu ENC28J60 je připravena pro provoz v IPv6 sítích nejrůznější konfigurace[1]. Vhodným pokračováním práce na knihovně by mohla být podpora stavové konfigurace, která by umožnila zapojení zařízení do sítí spravovaných výhradně pomocí protokolu DHCPv6. Pro mnohé aplikace by jistě byla užitečná nadstavba zpřístupňující transportní protokol TCP.

Výstupem této práce je nejen samotná komunikační knihovna, ale také demonstrační aplikace použitelná pro sledování příchozí síťové komunikace, zjištění konfigurace místní sítě, získání aktuálního času z NTP serveru či test dosažitelnosti pomocí aplikace *ping*.

[1] Omezení daná rozsahem implementace a zdroji hardware jsou vysvětlena v kapitolách 2.5.3 a 4.3.2.

Literatura

[1] *PIC24FJ64GA002 product page* [online]. http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en026374, 12.2.2010 [cit. 2010-05-10].

[2] *ENC28J60 product page* [online]. http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en022889, 13.2.2008 [cit. 2009-12-21].

[3] *Internet Control Message Protocol version 6 (ICMPv6) Parameters* [online]. http://www.iana.org/assignments/icmpv6-parameters, 18.12.2009 [cit. 2010-05-07].

[4] *Assigned Internet Protocol numbers* [online]. http://www.iana.org/assignments/protocol-numbers, 28.4.2010 [cit. 2010-04-30].

[5] *Port numbers* [online]. http://www.iana.org/assignments/port-numbers, 3.5.2010 [cit. 2010-05-04].

[6] *Adresy IPv6* [online]. http://www.nix.cz/cz/ipv6_addr, c2009 [cit. 2010-04-14].

[7] *MSP430F2617 product page* [online]. http://focus.ti.com/docs/prod/folders/print/msp430f2617.html, c2010 [cit. 2010-03-22].

[8] *Databáze ISP* [online]. http://www.ipv6portal.cz/databaze/isp, c2010 [cit. 2010-04-14].

[9] *Building an IPv6 router with GNU/Linux* [online]. http://tomicki.net/ipv6.router.php, Naposledy aktualizováno 21.4.2010 [cit. 2010-04-30].

[10] *FITkit* [online]. http://merlin.fit.vutbr.cz/FITkit, Naposledy aktualizováno 22.3.2010 [cit. 2010-05-03].

[11] Abbley, J.; Savola, P.; Neville-Neil, G.: *RFC 5095: Deprecation of Type 0 Routing Headers in IPv6* [online]. http://tools.ietf.org/html/rfc5095, 2007.

[12] Abeille, J.: *Contiki 6lowpan/uIPv6 FAQ* [online]. http://www.sics.se/contiki/uipv6-faq.html, 29. 9. 2008, [cit. 2010-05-07].

[13] Arkko, J.; Blanchet, M.; Chakrabarti, S.; aj.: *RFC 4294: IPv6 Node Requirements* [online]. http://tools.ietf.org/html/rfc4294, 2006.

[14] Bigelow, S. J.: *Mistrovství v počítačových sítích.* Computer press, 2004, iSBN 80-251-0178-9.

[15] Conta, A.; Deering, S.: *RFC 2463: Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6)* [online]. `http://tools.ietf.org/html/rfc2463`, 1998.

[16] Crawford, M.: *RFC 2464: Transmission of IPv6 Packets over Ethernet Networks* [online]. `http://tools.ietf.org/html/rfc2464`, 1998.

[17] Deering, S.; Hinden, R.: *RFC 2460: Internet Protocol, Version 6 (IPv6)* [online]. `http://tools.ietf.org/html/rfc2460`, 1998.

[18] Hagen, S.: *IPv6 Essentials.* O'Reilly & Associates, Inc., 2002, iSBN 0-596-00125-8.

[19] Hinden, R.; Deering, S.: *RFC 4291: IP Version 6 Addressing Architecture* [online]. `http://tools.ietf.org/html/rfc4291`, 2006.

[20] Kozierok, C. M.: *The TCP/IP Guide* [online]. `http://www.tcpipguide.com/free/index.htm`, Version 3.0, 20.10.2005 [cit. 2010-05-18].

[21] Krčmář, P.: *Návod: Jak jednoduše a rychle na IPv6* [online]. `http://www.root.cz/clanky/navod-jak-jednoduse-a-rychle-na-ipv6/`, 12.11.2008 [cit. 2010-02-19].

[22] Loshin, P.: *IPv6: Theory, protocol and practice.* Elsevier, Inc., 2004, iSBN 1-55860-810-9.

[23] Mayer, K.; Fritsche, W.: IP-enabled wireless sensor networks and their integration into the internet. In *InterSense '06: Proceedings of the first international conference on Integrated internet ad hoc and sensor networks*, ACM, 2006, ISBN 1-59593-427-8.

[24] Mills, D.: *RFC 2030: Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI* [online]. `http://tools.ietf.org/html/rfc2030`, 1996.

[25] Narten, T.; Nordmark, E.; Simpson, W.: *RFC 2461: Neighbor Discovery for IP Version 6 (IPv6)* [online]. `http://tools.ietf.org/html/rfc2461`, 1998.

[26] Peterka, J.: *Protocol stack* [online]. `http://www.earchiv.cz/a95/a518k130.php3`, 30.8.1995 [cit. 2010-04-29].

[27] Satrapa, P.: *Internetový protokol verze 6.* CZ.NIC, z. s. p. o., 2008, iSBN 978-80-904248-0-7.

[28] Spurgeon, C. E.: *Ethernet: The definitive guide.* O'Reilly Media, Inc., 2000, iSBN 1-56592-660-9.

[29] Thomson, S.; Narten, T.: *RFC 2462: IPv6 Stateless Address Autoconfiguration* [online]. `http://tools.ietf.org/html/rfc2462`, 1998.

Dodatek A

Obsah CD

dox/	programová dokumentace ve formátu HTML
net6/	zdroj řešení pro prostředí QDevKit
tex/	zdroj technické zprávy pro prostředí LaTeX
bt-xpecha02.pdf	technická zpráva ve formátu PDF
bt-xpecha02-print.pdf	technická zpráva ve formátu PDF pro tisk
readme	nápověda k obsahu CD

Dodatek B

Návod ke konfiguraci IPv6 routeru

V této příloze popíši konkrétní postup, který jsem použil při konfiguraci IPv6 routeru na operačním systému Ubuntu 9.10. Mnohé části návodu však mají obecnou platnost, chceme-li použít některou z linuxových distribucí. Vycházel jsem z návodů [21, 9].

Založil jsem si účet u společnosti *Hurricane Electric*, přičemž jsem pro potřeby své lokální sítě obdržel prefix `2001:470:1f0b:108f::/64`. Vytvořil jsem si skript pro shell *bash* s názvem `ipv6-router` a umístil jsem jej tak, aby byl spuštěn při aktivaci libovolného síťového rozhraní (adresář `/etc/network/if-up.d`). V něm jsem vytvořil tunel reprezentovaný rozhraními `sit1` (místní zakončení tunelu) a `sit0` (vzdálené zakončení tunelu s adresou brány ve formátu tzv. *IPv4 kompatibilní adresy*), opravil jsem automaticky generovanou směrovací tabulku a nakonec jsem přidal *výchozí cestu (default route)*, která směruje téměř všechen IPv6 provoz na rozhraní `sit1`. [21] Výsledný obsah souboru `ipv6-router` byl následující:

```
#!/bin/sh
PATH=/sbin:/bin:/usr/bin
ifconfig sit0 up
ifconfig sit0 inet6 tunnel ::216.66.80.30
ifconfig sit1 up
ifconfig sit1 inet6 add 2001:470:1f0b:108f::2/64
ip -6 route del 2001:470:1f0b:108f::/64 dev sit1
ip -6 route add 2001:470:1f0b:108f::/64 dev eth0
route -A inet6 add ::/0 dev sit1
```

Úspěšné přeposílání IPv6 provozu od ostatních uzlů v síti do tunelu a zpět však bylo třeba ještě povolit přidáním následujícího řádku do souboru `/etc/sysctl.conf`:

```
net.ipv6.conf.all.forwarding = 1
```

Aby mohly ostatní uzly v síti služeb směrovače využívat, bylo třeba ještě spustit službu `radvd`. Její název skrývá slova *Router advertisement daemon* a napovídá tedy, že hlavní funkcí je rozesílání zpráv *Router advertisement* protokolu ICMPv6. Mnohé údaje v konfiguračním souboru `/etc/radvd.conf` není třeba měnit, v každém případě je však třeba nastavit rozhraní, na němž se mají ohlášení směrovače rozesílat, a upravit prefix sítě tak, aby odpovídal konfiguraci tunelu. [9] Konkrétní podoba mého nastavení byla následující:

```
// zvolíme rozhraní, jehož se bude nastavení týkat
interface eth0
{
  // povolíme periodické rozesílání zpráv Router advertisement
  AdvSendAdvert on;
  // nastavíme rozmezí náhodného intervalu pro rozesílání zpráv
  MinRtrAdvInterval 5;
  MaxRtrAdvInterval 15;

  // zadáme prefix, který se bude prostřednictvím zpráv rozesílat
  prefix 2001:470:1f0b:108f::/64
  {
    // povolíme využití prefixu pro rozlišování uzlů na lince
    AdvOnLink on;
    // povolíme využití prefixu pro automatickou konfiguraci
    AdvAutonomous on;
  };
};
```