VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

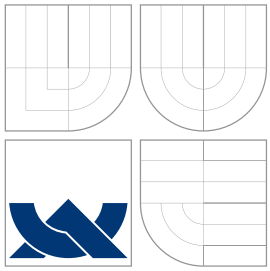
HARDWAROVÁ AKCELERACE ANALÝZY A EXTRAKCE POLOŽEK Z HLAVIČEK PAKETŮ

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

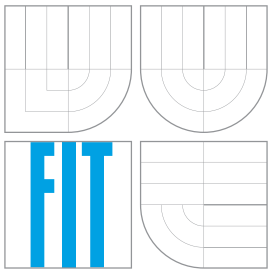
AUTOR PRÁCE
AUTHOR

LIBOR POLČÁK

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

HARDWAROVÁ AKCELERACE ANALÝZY A EXTRAKCE POLOŽEK Z HLAVIČEK PAKETŮ

HARDWARE ACCELERATION OF ANALYSIS AND HEADER FIELD EXTRACTION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — LIBOR POLČÁK
AUTHOR

VEDOUCÍ PRÁCE — Ing. JAN KOŘENEK
SUPERVISOR

BRNO 2008

Zadání bakalářské práce/5232/2007/xpolca03

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2007/2008

Zadání bakalářské práce

Řešitel: **Polčák Libor**
Obor: Informační technologie
Téma: **Hardwarová akcelerace analýzy a extrakce položek z hlaviček paketů**
Kategorie: Počítačová architektura

Pokyny:

1. Seznamte se s technologií FPGA a kartou COMBO6X.
2. Nastudujte síťové protokoly jednotlivých vrstev referenčního modelu ISO/OSI.
3. Analyzujte možnosti hardwarové realizace extrakce položek z hlaviček paketů.
4. Navrhněte vhodnou hardwarovou architekturu pro zpracování paketů pro 10 Gb/s a 40 Gbps. Snažte se o dosažení maximální propustnosti s využitím minimálního počtu hardwarových zdrojů.
5. Vytvořte implementaci navrženého řešení a ověřte na kartě COMBO6X v několika vybraných síťových aplikacích.
6. V závěru diskutujte parametry vytvořené implementace a naznačte možnosti rozšíření vytvořené hardwarové realizace.

Literatura:

- Protocol Wrappers for Layered Network Packet Processing in Reconfigurable Hardware, by Florian Braun, John Lockwood, and Marcel Waldvogel; *IEEE Micro*, Volume 22, Number 3, Feb 2002, pp. 66-74.
- Podle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1-3 zadání.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Kořenek Jan, Ing.**, UPSY FIT VUT
Datum zadání: 1. listopadu 2007
Datum odevzdání: 14. května 2008

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačových systémů a sítí
612 66 Brno, Božetěchova 2

L.S.

doc. Ing. Zdeněk Kotásek, CSc.
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. Pan

Jméno a příjmení: **Libor Polčák**
Id studenta: 78789
Bytem: Revoluční 12, 795 01 Rýmařov
Narozen: 14. 10. 1985, Rýmařov
(dále jen "autor")

a

2. Vysoké učení technické v Brně
Fakulta informačních technologií
se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....
(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

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

Název VŠKP: Hardwarová akcelerace analýzy a extrakce položek z hlaviček paketů
Vedoucí/školitel VŠKP: Kořenek Jan, Ing.
Ústav: Ústav počítačových systémů
Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v:
tištěné formě počet exemplářů: 1
elektronické formě počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti:
   - ☐ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

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

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

....

Nabyvatel Autor

Abstrakt

Tato práce se zabývá analýzou paketů a jejich zpracováním ve vysokorychlostních sítích za použití FPGA. Byl navržen model analýzy protokolů a vhodná hardwarové architektura. Popis protokolů je možno vytvořit pomocí XML, který je automatizovaně převeden do popisu ve VHDL. Díky tomu, že se zpracovává současně více bajtů, případně hlaviček protokolů, v jednom hodinovém cyklu, je navržená jednotka schopna zpracovávat pakety na rychlostech 10 Gb/s.

Klíčová slova

Síť, analýza paketů, extrakce položek, FPGA.

Abstract

This work deals with packet analysis and processing for high speed networks using FPGA. Model of the analysis and hardware architecture have been proposed. Protocols can be specified in XML. Automated tool is able to convert this specification to VHDL. As multiple bytes and protocol headers are processed within one clock cycle simultaneously, the proposed unit is able to handle packet processing on 10 Gbps speed and higher.

Keywords

Network, packet analysis, header field extraction, FPGA

Citace

Libor Polčák: Hardwarová akcelerace analýzy a extrakce položek z hlaviček paketů, bakalářská práce, Brno, FIT VUT v Brně, 2008

Hardwarová akcelerace analýzy a extrakce položek z hlaviček paketů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Jana Kořenka. Další informace mi poskytli kolegové z projektu Liberouter. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Libor Polčák
13. května 2008

Poděkování

Především bych rád poděkoval vedoucímu své diplomové práce panu Ing. Janu Kořenkovi za odborné vedení a čas věnovaný konzultacím této práce. Také bych chtěl poděkovat kolegům z projektu Liberouter za poskytnutí informací a zajištění technické podpory při návrhu a implementaci.

Obsah

Kapitola 1

Úvod

Síťové technologie jsou jednou z oblastí, které v poslední době zažívají velký rozvoj. Objem dat, který je přenášen mezi počítači se zvětšuje a proto se setkáváme se stále rychlejšími sítěmi, které umožňují přenést větší objem dat za časovou jednotku.

Data jsou po síti přenášena pomocí paketů. Paket obsahuje kromě přenášených dat ještě další části, které obsahují ve svých položkách důležité informace pro správné zpracování a doručení paketu. Většinou se jedná o hlavičky umístěné před vlastními přenášenými daty, v některých případech se přidávají i patičky za přenášená data. Při vytváření paketů se obvykle využívá principu zapouzdření. Hlavička, data a patička jednoho protokolu tvoří dohromady data jiného protokolu, který k nim přidá další hlavičku a patičku.

V síťovém provozu se můžeme setkat s různými druhy zařízení, které zajišťují její chod. Směrovače jsou nejdůležitějšími z nich. Jejich úkolem je zajistit, aby se přenášené pakety dostaly ze zdrojového uzlu na uzel cílový. Aby mohly být pakety zpracovány je potřeba zjistit informace uložené v hlavičkách paketů obsahující cílovou adresu, případně adresy uzlů, přes které má cesta vést.

Dalším často využívaným zařízeními jsou bezpečnostní prvky jako je firewall a bezpečnostní systémy určené k prevenci útoků [8]. Také tyto systémy využívají informace získané z hlaviček analyzovaných paketů. Jedná se především o zdrojové a cílové adresy a porty, použité protokoly apod.

Monitorovací systémy [6] slouží k analýze dat přenášených po síti. Ze získaných dat zjišťují statistické informace, které jsou určeny k dalšímu zpracování a analýze. Sledují použité protokoly a jejich stavové informace, zdrojové a cílové adresy a porty.

Ve větších sítích se již setkáváme s rychlostí 10 Gb/s nebo i vyšší. Od síťových zařízení je vyžadováno, aby si s těmito rychlostmi dokázala poradit. Dosažení tohoto cíle není úplně snadné, protože zpracování univerzálním procesorem není dostatečně rychlé. Existuje několik možností, jak výkonově kritické operace urychlit. Mohou být například přesunuty do aplikačně specifických obvodů nebo programovatelných hradlových polí FPGA. Cílem této práce je navrhnout architekturu, umožňující analýzu příchozích paketů a extrakci aplikačně specifických dat, pracující na rychlostech 10–40 Gb/s při využití technologie FPGA.

V současnosti existuje několik řešení pro hardwarovou akceleraci analýzy hlaviček paketů a extrakci dat z jejich položek, například [3, 1]. Žádné z nich však nebylo navrženo pro rychlosti přesahující 10 Gb/s.

Protože různá síťová zařízení potřebují ke své činnosti různé informace získané z příchozích paketů, je důležitým požadavkem na jednotku, aby byla snadno konfigurovatelná. Může jít jak o přidaní podpory pro nové protokoly, tak změnu extrahovaných dat z aktuálně podporovaných protokolů. Konfiguraci mají zvládnout i uživatelé, kteří s tvorbou

aplikací pro technologii FPGA nemají zkušenosti. Součástí práce je i vytvoření nástroje pro automatizovaný převod této konfigurace do cílového jazyka určeného k popisu hardwaru.

Navržená jednotka musí být použitelná v rámci projektu Liberouter [11], který se zabývá návrhem a tvorbou síťových aplikací určených pro rychlé sítě. Tento požadavek ovlivňuje především rozhraní jednotky, které musí být s projektem kompatibilní.

Teoretický úvod (kapitola 2) obsahuje vysvětlení referenčního modelu ISO/OSI, srovnání s architekturou protokolů TCP/IP a jsou zde uvedeny příklady protokolů, které má jednotka umět zpracovávat, a také přehled aplikací, které mohou využít služeb navrhované jednotky. Popis zadaného problému a přístupy k jeho řešení používané v současnosti jsou předloženy v kapitole 3. V kapitole 4 je popsána architektura navržené jednotky, její konfigurace pomocí XML [5] a převod konfigurace do jazyka VHDL. Výsledky získané po implementaci navržené architektury jsou prezentovány v kapitole 5. V závěru (kapitola 6) je provedeno zhodnocení dosažených cílů práce a její přínos a diskutovány možnosti dalšího pokračování.

Kapitola 2

Teoretický úvod

2.1 Referenční model ISO/OSI

Referenční model ISO/OSI byl vytvořen mezinárodní organizací ISO (*International Organization for Standardization*) jako abstraktní model pro znázornění síťového prostředí. V současné době je používán především pro studijní účely [16]. Model obsahuje sedm vrstev znázorněných na obrázku 2.1 a popsaných níže.

Obrázek 2.1: Referenční model ISO/OSI

Fyzická vrstva se zabývá přenosem jednotlivých bitů mezi síťovými uzly. Specifikuje přenosové médium a definuje všechny fyzikální a elektrické vlastnosti zařízení. Rozlišujeme fyzické spojení dvoubodové (sériová linka) a mnohobodové (Ethernet).

Linková vrstva přenáší větší bloky dat (*rámce*) mezi síťovými uzly. Umí detekovat poškozené rámce a zajistit jejich opětovné odeslání. Rámce obsahují fyzickou adresu zdroje a cíle.

Síťová vrstva má na starosti směrování paketu mezi různými sítěmi, zprostředkovává jeho doručení od zdroje k cíli. Také informuje o problémech, které se mohou vyskytnout při doručování dat.

Transportní vrstva je první vrstvou, která se vyskytuje pouze na koncových bodech spojení. Jejím úkolem je adresování procesů v rámci jednoho počítače a také může zajišťovat spolehlivé doručení všech odeslaných segmentů.

Relační vrstva udržuje stavové informace konkrétního spojení. Pokud je to požadováno, může zajišťovat atomičnost prováděných transakcí.

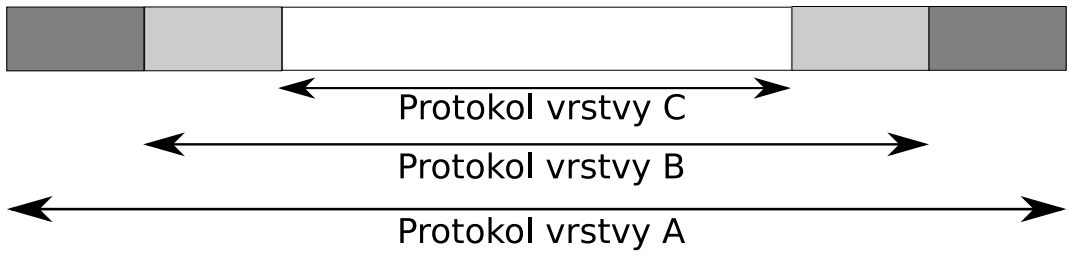
Prezentační vrstva převádí uspořádání dat používané na zdrojovém počítači na takové, které je vhodné pro přenos po síti a poté na uspořádání používané na cílovém počítači. Jde o změnu pořadí bajtů, převod kódů a abeced apod. Úkolem této vrstvy není zkoumat význam dat, ale pouze jejich strukturu.

Aplikační vrstva umožňuje podobně zaměřeným aplikacím komunikovat mezi sebou, spolupracovat. Mezi služby této vrstvy patří přenos souborů, elektronická pošta, diskuzní skupiny atd.

Každá z těchto vrstev má za úkol plnit jasně definované funkce, pro jejichž splnění využívá služeb sousední nižší vrstvy. Své služby pak nabízí sousední vyšší vrstvě.

Komunikaci zahajuje zdrojový uzel. Některý z procesů běžících na tomto uzlu vytvoří v aplikační vrstvě požadavek. Ten je zpracován prezentační vrstvou a předán relační vrstvě, která vytvoří spojení s cílovým uzlem. Takto se postupuje směrem k nižším vrstvám až k vrstvě fyzické, která má přístup k přenosovému médiu. Není dovoleno žádnou ze sedmi vrstev vynechat, ale je možné, že některá nebude aktivní. Takovou vrstvu pak nazýváme *nulovou* nebo *transparentní*.

Při průchodu aplikačních dat jednotlivými vrstvami se k nim přidávají přidávají hlavičky charakteristické pro danou vrstvu. Při použití některých protokolů se k datům přidávají také patičky, kontrolní data umístěná za přenášenými daty. Tím dochází k několikanásobnému zapouzdření původní informace. Příklad je zobrazen na obrázku 2.2. U příjemce se pak postupuje obráceně a postupně se zpracovávají řídící informace v hlavičce, případně patičce, zpracovávané vrstvy.



Obrázek 2.2: Zapouzdření dat v při tvorbě paketu

2.2 Protokolová architektura TCP/IP

Protokolová architektura TCP/IP [23] vznikla v době, kdy ještě referenční síťový model ISO/OSI neexistoval. Přesto jsou si oba přístupy velmi podobné. Je zde také využit vrst-

vový model, který pracuje obdobně jako model ISO/OSI. Architektura je definovaná sadou protokolů používaných pro komunikaci v počítačových sítích.

Vrstvy této architektury se částečně liší od vrstev modelu ISO/OSI (viz obrázek 2.3).

Obrázek 2.3: Vrstvy architektury TCP/IP

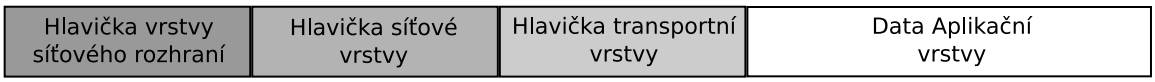
Vrstva síťového rozhraní zahrnuje služby fyzické a linkové vrstvy ISO/OSI modelu. Liší se v závislosti na použité síti. Má přímý přístup k fyzickému médiu. Data jsou z pohledu síťové vrstvy přenášena po blocích - rámcích.

Síťová vrstva odpovídá síťové vrstvě ISO/OSI modelu.

Transportní vrstva odpovídá transportní vrstvě ISO/OSI modelu. Navíc poskytuje některé služby dostupné v relační vrstvě ISO/OSI modelu, jako je například možnost udržování a využívání stavových informací konkrétních spojení.

Aplikační vrstva zajišťuje zbylé služby relační vrstvy, a všechny služby vrstev prezentační a aplikační. Mimo vlastní komunikaci mezi dvěma aplikacemi musí tedy zajišťovat převod uspořádání dat i případné požadavky na atomičnost transakcí apod.

Stejně jako v případě modelu ISO/OSI je i v protokolové architektuře využíván princip zapouzdření (viz obrázek 2.4). V rámci transportní, síťové i vrstvy síťového rozhraní dochází k přidání vlastní hlavičky k datům poskytnutým vyšší vrstvou a tato data jsou k dalšímu zpracování odeslaná jako jeden celek.



Obrázek 2.4: Zapouzdření dat v architektuře TCP/IP

2.3 Popis protokolů

Protokol slouží k výměně informací mezi dvěma síťovými uzly. Kromě struktury přenášených dat definuje protokol i způsob navazování a ukončení komunikace, adresování uzlů v síti, zda a jak se budou zpracovávat chyby, jak bude komunikace v čase vypadat (potvrzování přijatých dat, omezení odesílaných dat, aby se předešlo zahlcení sítě apod.) a také přenos dat.

Rozlišujeme protokoly standardizované institucemi jako je IEEE, ISO apod., nebo volně dostupné na internetu jako *Request for Comments* (žádost o komentáře) [20] a protokoly proprietární (soukromé), které jsou dostupné jen omezené skupině lidí, například v rámci jedné společnosti. V dalším textu se budeme zabývat protokoly standardizovanými a dostupnými jako RFC.

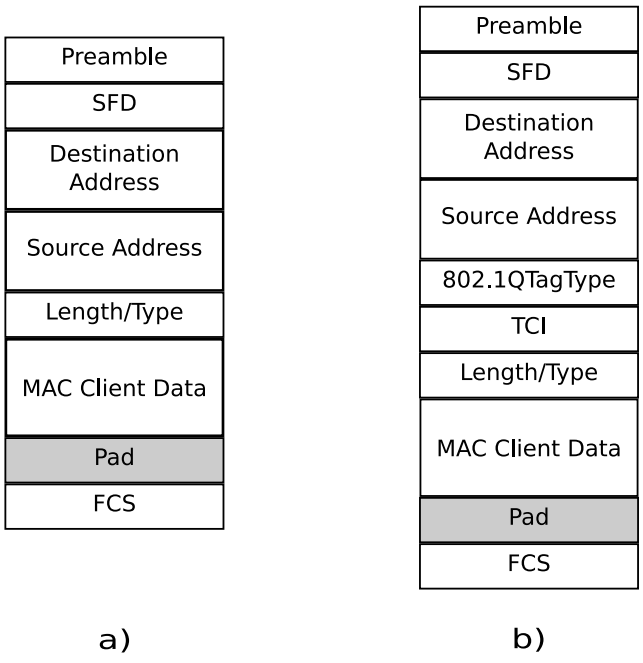
Data jsou daným protokolem přenášena v paketech který se dělí na hlavičku a přenášená data, za kterými ještě v některých případech následuje patička. Hlavička se skládá z několika polí, které nemusí být povinné, případně mohou mít variabilní délku. Pro formální popis protokolu se mohou použít regulární a bezkontextové gramatiky doplněné o případné sémantické vazby, případně grafové modely (například Petriho sítě).

Aplikace, které budou využívat navrhovanou jednotku pracují s daty přenášenými na vrstvě síťového rozhraní, síťové a transportní vrstvě protokolové architektury TCP/IP. Dále se budeme soustředit pouze na protokoly patřící do těchto vrstev.

Následuje přehled základních protokolů, které musí jednotka umět zpracovat. Popis je zaměřen především na formát přenášených dat, protože ten je pro návrh jednotky nejdůležitější.

2.3.1 Ethernet

Ethernet [7] je jedním z nejpoužívanějších protokolů vrstvy síťového rozhraní. Kromě hlavičky a patičky jsou k datům síťové vrstvy přidány kontrolní značky označující začátek a konec rámce. Na obrázku 2.5 je znázorněna struktura rámce.



Obrázek 2.5: Formát ethernetového rámce a) standardní formát b) rámec s VLAN tagem

Preamble (56 bitů) je tvořena posloupnost střídajících se bitů 1 a 0, která slouží k zajištění časové synchronizace.

Start Frame Delimiter (SFD) (8 bitů) slouží k oddělení preambule od začátku rámce. Skládá se také ze střídajících se bitů 1 a 0, jen poslední z nich je invertován na hodnotu 1. Má tedy hodnotu *10101011*.

Destination Address (48 bitů) identifikuje cíl rámce. Může se jednat o adresu individuální (1 uzel), multicastovou, nebo broadcastovou (skupina uzlů).

Source Address (48 bitů) identifikuje zdroj rámce.

Length/Type (16 bitů) pokud je menší nebo rovna hodnotě 1500 obsahuje délku rámce, jinak identifikuje použitý protokol na třetí vrstvě ISO/OSI modelu.

MAC Client Data + Pad (46 - 1500 oktetů) obsahuje data protokolů vyšších vrstev. Pokud jejich délka je menší než 46 oktetů, jsou doplněna položkou Pad tak, aby součet délek obou položek odpovídal minimální povolené velikosti. Dodržení minimální délky je důležité pro správné fungování mechanismu CSMA/CD [7] zabraňujícímu přijetí chybného rámce při kolizi, která nastává při současném vysílání více uzlů.

Frame Check Sequence (FCS) (32 bitů) se používá ke kontrole správnosti přijatých dat. Obsahuje kontrolní součet všech políček rámce kromě preambule, SFD a FCS.

S rozvojem rozlehlejších sítí vznikla potřeba vytvoření několika logických sítí v rámci jedné fyzické sítě. Tohoto cíle bylo dosaženo vytvořením *virtuálních lokálních sítí* (VLAN), ve kterých se využívá upravený ethernetový rámec. Tento typ rámce je v současné době standardizován [7] a je označován jako typ 802.1q. Od původního rámce se liší tím, že položka Length/Type obsahuje hexadecimální hodnotu 8100. Následuje položka *Tag Control Info* o velikosti 2 oktety, která obsahuje řídící informace, jako je priorita a VLAN identifikátor, který rozlišuje jednotlivé logické sítě. Teprve následující 2 oktety mají význam původního pole Length/Type.

2.3.2 Internet Protocol verze 4

Internet Protocol verze 4 (IPv4) [21] je v současné době nejrozšířenějším protokolem síťové vrstvy.

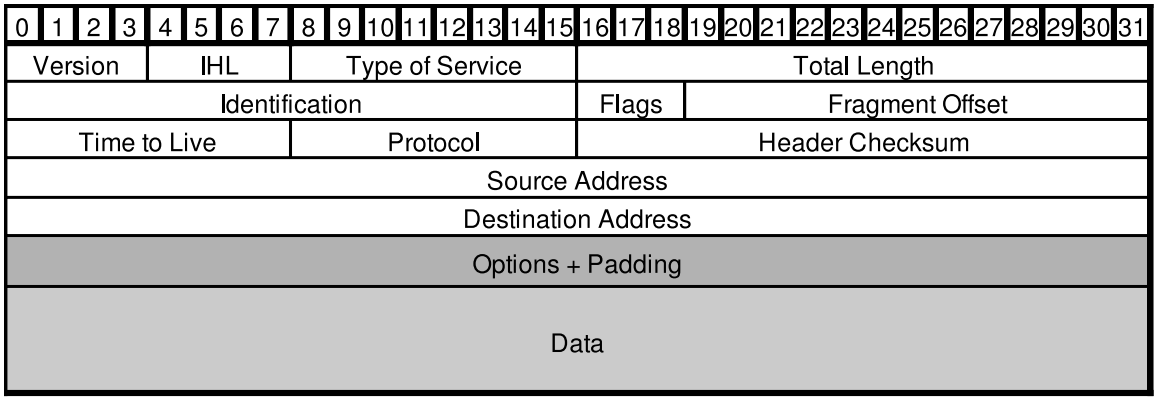
Na základě IP adres zajišťuje doručování předaných datagramů. Protože celková velikost dat předávaných nižší vrstvě může být vyšší než maximální povolená velikost dat, kterou je schopna zpracovat, je možné datagram v počátečním uzlu nebo po cestě rozdělit do několika fragmentů. Přenášený paket (obrázek 2.6) tak obsahuje hlavičku a data, která mohou být buď celý přenášený datagram, nebo jeho část (fragment). Pakety nemusí být doručeny v pořadí v jakém byly odeslány. Některé se mohou dokonce ztratit, nebo k cíli dorazit vícekrát. Poskytovaná služba tedy není spolehlivá.

Version (4 bity) obsahuje číslo verze internetového protokolu, je rovna 4.

Internet Header Length (IHL) (4 bity) specifikuje délku hlavičky ve 32-bitových slovech. Minimální platná hodnota je 5.

Type of Service (8 bitů) určuje typ požadované služby. Tento parametr je možno využít pro pro zajištění kvality přenosu vhodným nastavením síťových prvků.

Total Length (16 bitů) udává celkovou délku paketu včetně hlavičky v oktetech.



Obrázek 2.6: Formát IPv4 paketu

Identification (16 bitů) identifikuje fragmentovanou část datagramu.

Flags (3 bity) je využíváno při fragmentaci datagramu, obsahuje bity zakazující další fragmentaci, označující poslední fragment a oznamující, že další fragmenty budou následovat.

Fragment Offset (13 bitů) označuje pozici fragmentu v rámci datagramu.

Time To Live (8 bitů) se používá k zamezení nekonečného putování paketů v rámci sítě, pokud by došlo k porušení směrovacích tabulek a paket by se pohyboval v kruhu. Hodnota je nastavena v počátečním uzlu a při každém průchodu směrovačem je snížena o 1. Pokud dojde k dosažení nuly, je paket zahozen a vygenerováno chybové hlášení, které je odesláno zdrojovému uzlu.

Protocol (8 bitů) obsahuje identifikátor dalšího zabaleného protokolu. Obvykle se jedná o protokol vyšší vrstvy.

Header Checksum (16 bitů) obsahuje kontrolní součet IP hlavičky.

Source Address (32 bitů) obsahuje IP adresu zdrojového uzlu.

Destination Address (32 bitů) obsahuje IP adresu cílového uzlu.

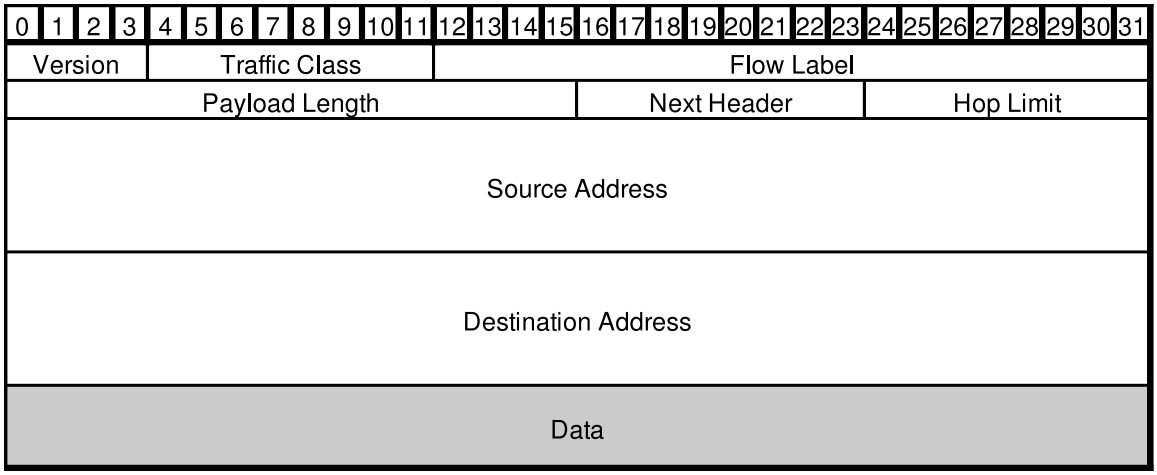
Options + Padding (proměnná délka, která je násobkem 32 bitů) obsahuje volitelné položky IP hlavičky jako je požadovaná cesta, zabezpečení atd. Kvůli složitosti zpracování se příliš nevyužívá. Pokud celková délka volitelných položek není násobkem 32 bitů, je využita položka Padding, která toto zarovnání zajistí.

Data obsahuje data vyšší vrstvy, jejich význam je určen obsahem položky Protocol.

2.3.3 Internet Protocol verze 6

Především kvůli nedostatku adres internetového protokolu verze 4 a jejich blízkému vyčerpání byla zavedena nová verze [4] označená číslem 6 (IPv6). Kromě zvětšení délky adresy identifikující síťové uzly ze 32 na 128 bitů, bylo zavedeno několik dalších změn, jako je například bezstavová konfigurace, podpora multicastové komunikace přímo ve specifikaci

(multicast pro IPv4 byl představen až později), podpora datagramů větších než 64 kilobajtů (tzv. *jumbogramů*), větší důraz na bezpečnost a v neposlední řadě je požit jiný formát (obrázek 2.7) hlavičky paketu.



Obrázek 2.7: Formát IPv6 paketu

Version (4 bity) obsahuje hodnotu 6 značící verzi internetového protokolu.

Traffic Class (8 bitů) používá se pokud je potřeba nastavit některým paketům jinou prioritu než ostatním a ovlivňuje jak zpracování paketu směrovači na cestě, tak přímo v cílovém uzlu.

Flow Label (20 bitů) specifikuje speciální zacházení routerů pro směrování paketů mezi zdrojem a cílem. Dá se využít například pro zajištění kvality služeb.

Payload Length (16 bitů) délka přenášených dat v oktetech. Do této délky není započítaná samotná délka základní IPv6 hlavičky. Pokud je pole nulové, jedná se o jumbogram.

Next Header (8 bitů) určuje typ další hlavičky. Odpovídá poli Protocol v IPv4 hlavičce. Je použita stejná množina hodnot jako u IPv4 protokolu rozšířená o identifikátory tzv. rozšiřujících hlaviček.

Hop Limit (8 bitů) má stejný význam jako Time to Live u IPv4 protokolu. Každý směrovač hodnotu snižuje o 1 a pokud se hodnota dostane na 0, je paket zahozen a vygenerován paket oznamující chybu, který je odeslán zdrojovému uzlu.

Source Address (128 bitů) obsahuje IP adresu zdrojového uzlu.

Destination Address (128 bitů) obsahuje IP adresu cílového uzlu.

Data obsahuje data vyšší vrstvy, případně rozšiřující hlavičky. Význam dat je určen obsahem položky Next Header.

Za povinnou základní hlavičkou mohou následovat přímo data protokolu vyšší vrstvy, nebo některá z rozšiřujících hlaviček, které nahrazují volitelné parametry dostupné v IPv4

hlavičce. Každá z těchto hlaviček znovu obsahuje pole Next Header, které může opět obsahovat identifikátor další rozšiřující hlavičky, nebo identifikátor protokolu vyšší vrstvy.

IPv6 definuje tyto rozšiřující hlavičky:

Hop-by-Hop Options header pokud je přítomna, musí být umístěna jako první ze všech rozšiřujících hlaviček. Obsahuje speciální požadavky, které musí zpracovat všechny směrovače na cestě.

Routing header obsahuje seznam adres směrovačů, přes které má být paket doručen k cíli.

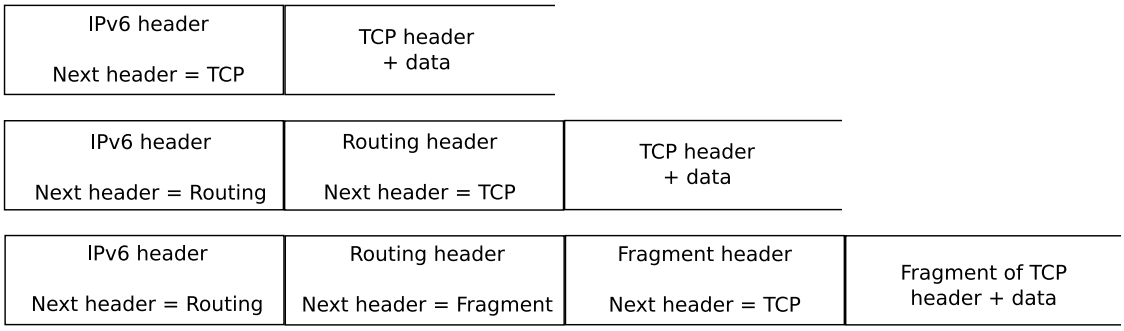
Fragment header slouží k přenesení informací nutných pro správné sestavení fragmentovaného datagramu. IPv6 připouští jen fragmentaci přímo ve zdrojovém uzlu. Zdrojový uzel si musí předem zjistit maximální možnou velikost paketu, kterou je možno k cíli přenést [13].

Destination Options header podle pozice na které se vyskytuje je zpracovávána jen na cílovém uzlu, nebo na cílovém uzlu a všech uzlech hlavičkou specifikovaných.

Authentication header slouží k zajištění autenticity a integrity [9].

Encapsulating Security Payload header slouží k ochraně přenášených dat [10].

Hlavičky jsou za sebou řazeny tak, jak je naznačeno na obrázku 2.8. Pořadí hlaviček není normou striktně určeno, je pouze doporučeno.

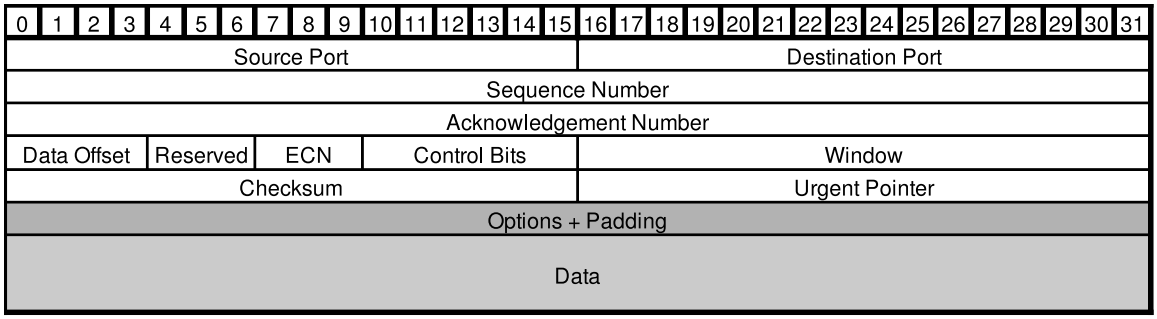


Obrázek 2.8: Příklad zapouzdření rozšiřujících hlaviček v IPv6 paketu

2.3.4 Transmission Control Protocol

Transmission Control Protocol (TCP) [22] je protokolem transportní vrstvy TCP/IP modelu, který zajišťuje spolehlivý přenos dat. Data jsou doručena přesně v takovém pořadí, v jakém byla odeslána. Ze souvislého bloku dat předaného vyšší vrstvou vytváří segmenty (obrázek 2.9), které posílá ke zpracování síťové vrstvě.

Jedná se o stavový protokol, to znamená, že si uchovává stav každého spojení. Využívá principu *klouzavého okna* a *pozitivního potvrzování* doručených segmentů. To mu umožňuje znovu požádat o segmenty ztracené na cestě, uspořádat segmenty, které došly mimo pořadí a identifikovat segmenty, které došly více než jednou. Také obsahuje mechanismus bránící zahlcení sítě.



Obrázek 2.9: Formát TCP segmentu

Source Port (16 bitů) obsahuje číslo portu, který identifikuje zdrojovou aplikaci.

Destination Port (16 bitů) obsahuje číslo portu, který identifikuje cílovou aplikaci.

Sequence Number (32 bitů) pokud je nastaven příznak SYN, potom obsahuje počáteční sekvenční číslo a první datový oktet má sekvenční číslo o 1 vyšší. Pokud příznak SYN nastaven není, obsahuje sekvenční číslo prvního datového oktetu v segmentu.

Acknowledgement Number (32 bitů) je platné pouze pokud je nastaven příznak ACK a obsahuje hodnotu, kterou odesilatel očekává v poli Sequence Number v příštím doručeném segmentu. Příjemce tuto informaci může využít ke znovu odeslání ztracených dat.

Data Offset (4 bity) udává počet 32-bitových slov v TCP hlavičce. Velikost hlavičky může být 20-60 oktetů, vždy jde o násobek 32 bitů.

Reserved (3 bity) Rezervováno pro budoucí potřeby. Mělo by být nastaveno na 0.

Explicit Congestion Notification (ECN) (3 bity) pole je využíváno k signalizaci zahlcení sítě přímo koncovými uzly. Aby bylo možno tento mechanismus využívat, musí se na dohodnout oba koncové body. Tato položka byla přidaná v RFC 3168 [19].

Control Bits (6 bitů) kontrolní bity

URG - Udává, zda je platná položka Urgent Pointer

ACK - Udává, zda je platná položka Acknowledgement Number

PSH - Udává, zda se má využít tzv. *push* funkce, cílový uzel předá data aplikaci jakmile dorazí. Používá se například u interaktivních aplikací, jako je telnet apod.

RST - Reset spojení

SYN - Odesílatel žádá o synchronizaci sekvenčních čísel.

FIN - Ukončuje spojení ve směru od odesilatele.

Window (16 bitů) udává počet oktetů, které je odesilatel schopen přijmout. Prvním oktetem se rozumí ten, na který odkazuje položka Sequence Number.

Checksum (16 bitů) obsahuje kontrolní součet počítaný z celého segmentu a části IP hlavičky (tzv. pseudohlavičky).

Urgent Pointer (16 bitů) pokud je nastaven příznak URG obsahuje sekvenční číslo posledního oktetu urgentních dat.

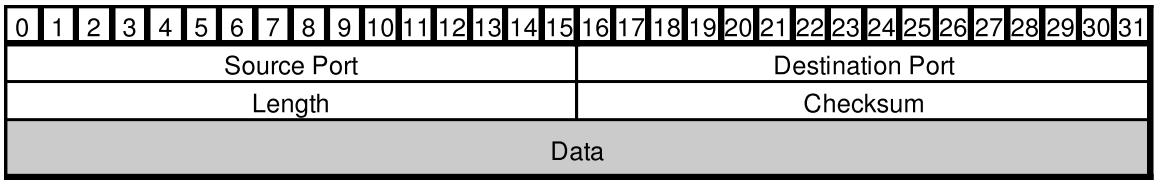
Options + Padding (proměnná délka, která je násobkem 32 bitů) jsou volitelné položky, které mohou následovat za povinnou hlavičkou. Pokud jejich délka není násobkem 32 bitů, jsou doplněny položkou Padding tak, aby byla tato podmínka splněna.

Data obsahuje data vyšší vrstvy.

2.3.5 User Datagram Protocol

User Datagram Protocol (UDP) [17] poskytuje nespolehlivé služby na transportní vrstvě. Využívá se v aplikacích, které spolehlivost doručení dat nevyžadují, protože si ji zajistí samy (kritické aplikace, kterým nestačí záruky, které poskytuje protokol TCP), nebo takové, u kterých je větší problém zpoždění dat, než jejich ztráta (přenos audia, videa apod.).

Protokol je bezstavový a nemá žádnou ochranu proti zahlcení sítě. Nezaručuje, že datagramy vytvořené z dat předaných vyšší vrstvou (obrázek 2.10 doručí právě jedenkrát. Datagram může k cíli dorazit více krát, nebo vůbec. Nezaručuje doručení v pořadí, v jakém byly datagramy odeslány.



Obrázek 2.10: Formát datagramu UDP

Source Port (16 bitů) obsahuje číslo portu, který identifikuje zdrojovou aplikaci, pokud je to potřeba. Pokud by tato informace neměla význam, obsahuje nuly.

Destination Port (16 bitů) obsahuje číslo portu, který identifikuje cílovou aplikaci.

Length (16 bitů) délka celého datagramu v bajtech. Minimální hodnota je 8.

Checksum (16 bitů) obsahuje výsledek kontrolního součtu pro IP pseudohlavičku, UDP hlavičku a data. Pokud se kontrolní součet nemá provádět, pole je nastaveno na hodnotu nula. Pokud by byl kontrolní součet nulový, je invertován na hodnotu 0xFFFF.

2.4 Aplikace

Existuje několik skupin síťových prvků, ve kterých se z důvodu rychlosti používá hardwarová akcelerace. Do této skupiny patří směrovače, monitorovací sondy, systémy pro detekci a prevenci nežádoucího provozu, firewally a generátory paketů pro testování výkonnosti sítě a síťových prvků. Při použití pouze softwarového řešení, bez využití hardwarové akcelerace, je možné, v závislosti na aplikaci, zpracovávat data v síti o rychlosti několika stovek magabitů za sekundu.

Směrovač, nebo-li router, je jedním ze základních prvků, na němž závisí fungování celé sítě. Pracuje na třetí, síťové, vrstvě ISO/OSI modelu. Podle údajů uložených ve směrovacích

tabulkách směrují tato zařízení pakety přijaté na jednom portu na port jiný. Z pohledu extrakce položek je pro správnou činnost směrovače nutné extrahovat položky týkající se směrování umístěné v protokolu síťové vrstvy. Jedná se o cílovou adresu, adresu uzlu, na který se má přeposlat v dalším kroku a položku, která udává, že se má paket zahodit (Time to Live, Hop Limit).

Monitorovací sondy se používají k získávání statistik o využití sítě, mohou přispět k odhalení uživatelů, kteří nerespektují pravidla sítě, mohou najít úzká hrdla sítě apod. Sledují všechny pakety (je možné provádět i deterministické vzorkování) procházející daným místem. Jako příklad je možné uvést sondy exportující data ve formátu *NetFlow* [2], či *IPFIX* [18]. Sondy sbírají ze sítě data, pakety agregují do toků a získaná data odesílají na kolektor, kde jsou uložena do databáze, kde je možné je dále analyzovat. Z pohledu extrakce položek z hlaviček paketů je především nutné získat zdrojové a cílové adresy a porty, informace o použitých protokolech a informace týkající se stavového spojení protokolu TCP apod.

Firewall slouží k základní ochraně sítě před útokem zvenčí. Jeho základem jsou pravidla, která určují, které pakety se mají zahodit a které je možné nechat projít do nebo ze sítě. Pakety je možné filtrovat na základě použitých protokolů, čísel portů a adres. Firewally také umožňují sledovat TCP komunikaci a uchovávat si stav spojení ve stavových tabulkách. Pro jejich správnou činnost je potřeba extrahovat položky z linkové, síťové i transportní vrstvy a to především zdrojové a cílové adresy a porty, TCP příznaky, směrovací informace.

Existují však útoky, kterým firewally zabránit nemohou. Jedná se například o útoky na standardní služby běžící na standardních portech, které není možné filtrovat. V takových situacích je možné využít IDS/IPS (Intrusion Detection/Prevention System). Jedná se o systémy, které slouží k detekci útoků, případně k jejich prevenci. Hledají vzory v jednotlivých paketech i celých tocích. Pro svou činnost využívá informace získané z aplikační vrstvy i vrstev nižších, ze kterých potřebuje podobné položky jako firewall. Pokud dojde k průniku do sítě, je jejich úkolem informovat administrátora, nebo vytvořit firewallové pravidla zabraňující pokračování útoku.

Kapitola 3

Přístup k řešení problému

Pro řešení problému analýzy hlaviček paketů a extrakce jejich položek existuje mnoho přístupů. Který přístup si zvolíme záleží na charakteru provozované síťové aplikace a její požadované propustnosti. Můžeme však říct, že potřebujeme zařízení, které dokáže vyřešit problém obecně formulovaný takto: je potřeba analyzovat skupinu n bajtů, odpovídající přijatému paketu, a na základě provedené analýzy identifikovat hlavičky jednotlivých protokolů. Podle druhu nalezených hlaviček je nutné extrahovat informace důležité pro správnou činnost aplikace.

3.1 Řešení pomocí obecného procesoru

K řešení je možné využít obecného procesoru dostupného v obvyklém PC, které je vybaveno nejméně jedním síťovým rozhraním, kterých může být v závislosti na provozované síťové aplikaci více (směrovač apod.). Přijaté pakety jsou přeneseny pomocí I/O sběrnice do hlavní paměti počítače.

PC musí být vybaveno vhodným programovým vybavením, které dokáže přijaté pakety z paměti načíst a zpracovat. V této fázi zpracování se musí vykonat všechny akce, které provozovaná síťové aplikace vyžaduje. V případě směrovače musí být, kromě analýzy protokolů v přijatém paketu a extrakce potřebných položek, realizována i logika, která za pomoci extrahovaných dat a uložených směrovacích tabulek rozhodne, co se má s paketem dále vykonat. Obvykle je rozhodnuto, že má být paket odeslán pomocí jednoho z dostupných síťových rozhraní. Paket je na něj přenesen pomocí I/O sběrnice.

Použití programového vybavení zajišťuje podporu snadného přidání nových protokolů. Navíc je již toto vybavení často dostupné v odladěné formě. Propustnost tohoto řešení je závislá na rychlosti použitého procesoru a propustnosti I/O sběrnic.

3.2 Využití specializovaného technického vybavení

Kvůli odstranění problému s omezenou propustností je ve výkonných síťových zařízeních využíváno specializovaného technického vybavení. Zpravidla bývá využita hardwarová akcelerace na bázi aplikačně specifických integrovaných obvodů (ASIC), nebo programovatelných hradlových polí (FPGA). Tyto technologie se od sebe liší tím, že u zařízení založených na technologii ASIC je jejich činnost pevně daná při výrobě, naopak při použití technologie FPGA je možno konfiguraci měnit, některé aplikace využívají i dynamickou rekonfiguraci za běhu. Technologie ASIC bývá obvykle energeticky méně náročná.

Komerční firmy používají častěji zařízení vyrobené na bázi technologie ASIC, protože se při výrobě velkého počtu kusů rozdělí cena výroby masky mezi všechna tato zařízení. Zařízení specializovaných na vysokorychlostní sítě se však obvyklé vyrábějí v malém počtu kusů a náklady na výrobu by se tak znatelně promítly v jejich ceně. Proto se v poslední době začíná využívat technologie FPGA, která umožňuje provádět dodatečné změny v návrhu, které nejsou spojeny s vysokými náklady na změnu výroby.

3.3 Současné architektury využívající technologii FPGA

Cílem teto práce je navržení obvodu schopného pracovat s propustností dat 10 Gb/s a více za využití technologie FPGA. V současné době existuje několik řešení.

Jednou z možností je implementace procesoru přímo v FPGA. Například je možno využít připraveného obecného procesoru MicroBlaze [25]. Pro využití v síťových aplikacích byl navržen a implementován také RISC procesor s instrukční sadou doplněnou o instrukce specializované na zpracování paketů [14]. Další z možností je využít jazyk Handel-C, který poskytuje vysokou úroveň abstrakce, a jednotku implementovat pomocí něj.

Z výsledků srovnání zmíněných řešení [3] je patrné, že ani jedno z těchto řešení nedosahuje propustnosti 1 Gb/s. Propustnost lze zvýšit použitím více jednotek s využitím paralelizace. Toto řešení je nevýhodné v tom, že je potřeba velké množství zdrojů na čipu. To je dáno jak více násobným použitím dané jednotky, tak nutnosti použití pomocných jednotek schopných rozdělovat vstupní tok na více části a spojovat více takových toků do jednoho.

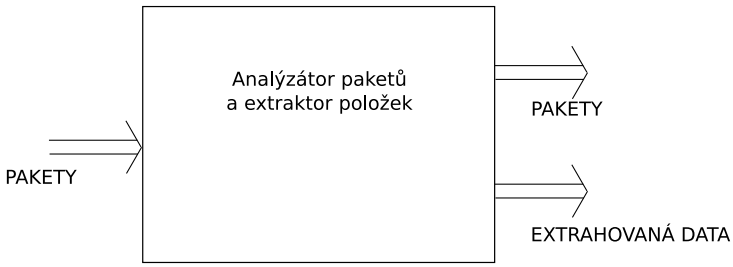
Dalším možným přístupem je využití vrstvové architektury [1]. Zpracování paketů je rozděleno do několika modulů, kde se každý modul zabývá zpracováním jedné vrstvy TCP/IP modelu. Uvedené výsledky ukazují, že toto řešení je vhodné pro sítě s rychlostí do 3 Gb/s.

Kapitola 4

Architektura a konfigurace

Než se dostaneme přímo k návrhu architektury jednotky a popisu její implementace, je potřeba zmínit specifikaci požadavků funkce jednotky a její konfigurovatelnosti.

Při návrhu bylo nutné uvažovat příjem vstupních a vysílání výstupních dat pomocí protokolu FrameLink [24], který se používá v rámci projektu Liberouter [11]. Protože je tento protokol na vyšších datových šířkách neefektivní, je potřeba již při návrhu jednotky počítat s možným přechodem na protokol LocalLink [12]. Podle požadavků má být na výstupu jednotky kromě extrahovaných položek označených identifikátorem také kopie vstupních dat (viz obrázek 4.1). Jednotka by měla být snadno uživatelský konfigurovatelná tak, aby bylo možno jednoduše přidat nový protokol nebo změnit, které položky se mají extrahovat. Jednotka by měla umožňovat extrakci dat na úrovní jednotlivých bajtů.



Obrázek 4.1: Vstupy a výstupy navrhované jednotky

Návrh jednotky by měl počítat s možností výskytu paketů, ve kterých je použito několik protokolů stejné vrstvy TCP/IP modelu, které jsou do sebe zabaleny. Jde například o situace, kdy je přenášen IPv4 paket uvnitř jiného IPv4 paketu, IPv6 paket uvnitř IPv4 paketu nebo dokonce zabalení Ethernetového rámce uvnitř IP paketu.

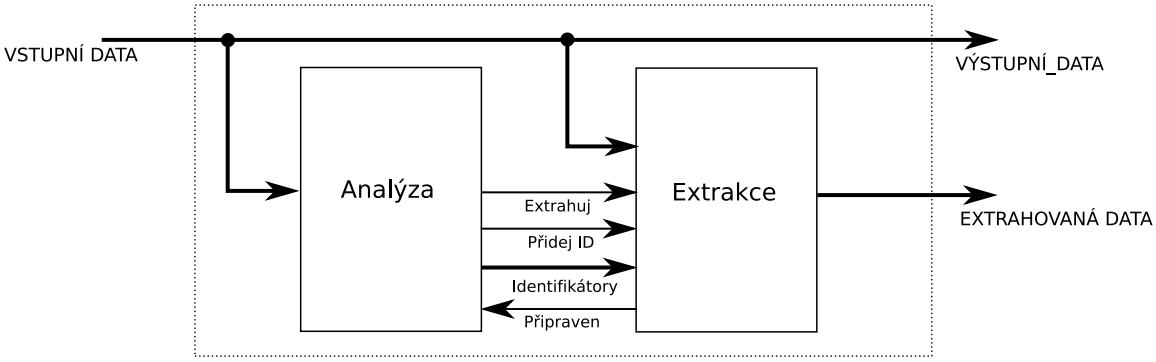
4.1 Popis architektury

Při návrhu architektury jednotky jsem vycházel z rozdělení jednotky na dvě části – moduly.

Úkolem prvního modulu (analyzátoru) je analýza vstupních dat a identifikace obsažených protokolů ve zpracovávaném paketu. Na základě této analýzy modul signalizuje dalšímu modulu, která data se mají extrahovat a jaký identifikátor je jednotlivým bajtům přiřazen.

Druhý modul (extraktor) vybírá ze vstupních dat ty bajty, které byly prvním modulem označené k extrakci. O každém zpracovávaném bajtu je potřeba vědět, zda má být extrahován, nebo zahozen, zda jde o nově extrahovanou položku a má se před ní přidat identifikátor, nebo zda jde o pokračování položky z předchozího bajtu a jaký identifikátor je dané položce přiřazen. Identifikátor je reprezentován číslem uloženým na osmi bitech. Rozhraní je ještě potřeba doplnit o signál, který analyzátoru oznamuje, že je extraktor připraven přijímat požadavky na extrakci.

Rozdělení jednotky na dva moduly, včetně jejich rozhraní je znázorněno na obrázku 4.2.



Obrázek 4.2: Dekompozice návrhu jednotky do modulů

Oddělení analýzy protokolů od extrakce dat není důležité jen z pohledu dekompozice problému na jeho jednotlivé části, ale také kvůli tomu, že jde o problémy vyžadující jiný způsob řešení pomocí hardwarové architektury. Zatímco při návrhu extrakční části je možné využívat principu zřetězeného zpracování, analyzátor musí umět zpracovávat vstupní slovo v jediném kroku.

Navržené rozdělení jednotky na moduly také přináší nezávislost modulů na sobě. Získáme tím možnost výměny jednoho z modulů za jiný, který vykonává stejnou činnost odlišným způsobem, bez toho, abychom museli upravovat druhý modul v jednotce. Například je možné nahradit modul provádějící extrakci, který má výstup formátovaný určitým způsobem, za modul s odlišným formátováním extrahovaných dat.

4.2 Propustnost jednotky

Propustnost jednotky t závisí přímo úměrně na počtu bajtů n, které zpracujeme v jednom hodinovém cyklu, a na frekvenci f, s jakou tuto akci provádíme. Platí tedy vzorec 4.1.

$$t = n \cdot f \tag{4.1}$$

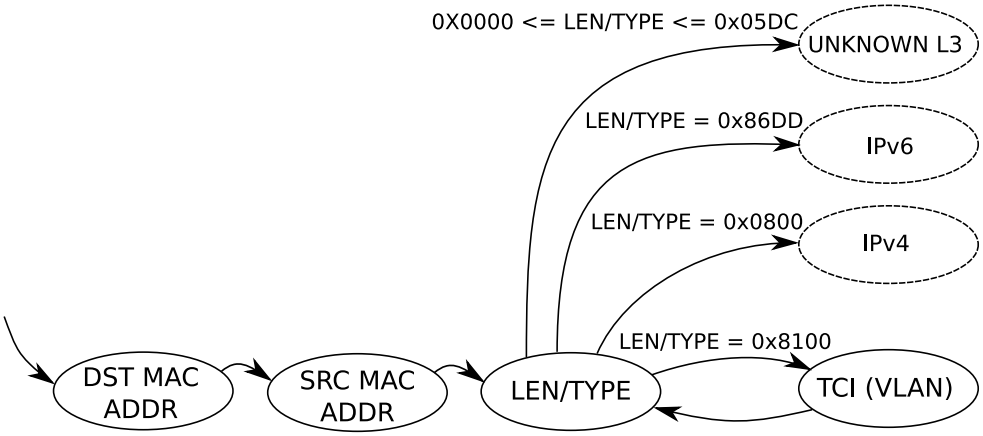
Řešení popisované v [1] i [3] pracují s datovými šířkami slova 2–4 bajty. To znamená, že v jednom hodinovém cyklu zpracují maximálně 2, případně 4 bajty. Protože potřebujeme pracovat s frekvencí 100, 125 nebo 156,25 MHz, lze snadno spočítat, že tato řešení nevyhovují. V závislosti na požadované propustnosti a zvolené frekvenci musí jednotka zpracovávat v jednom hodinovém cyklu datové slovo o šířce 8 bajtů a více.

Navrhovaná jednotka musí podporovat i protokoly s hlavičkami kratšími než je minimální délka slova. Například pro protokol MPLS [15] je minimální délka hlavičky 32 bitů. Také se může stát, že přechod mezi hlavičkami protokolů se nachází v rámci slova. Proto

je potřeba najednou zpracovávat hlavičky více protokolů. Při návrhu jednotky je potřeba s tímto problémem počítat. Pokud by jednotka nebyla schopna zpracovávat v jednom hodinovém cyklu všechny protokoly obsažené v aktuálním slově, snižovala by se propustnost jednotky.

4.3 Model analýzy vstupních dat

Abychom mohli navrhnout modul zajišťující analýzu přijatého rámce, musíme si nejdříve zvolit vhodný model, pomoci kterého tento analyzátor popíšeme. Jak již bylo uvedeno v sekci 2.3, můžeme protokoly formálně popsat pomocí bezkontextových gramatik doplněných o sémantické vazby nebo grafových modelů, jako jsou např. Petriho sítě. Příklad abstraktního modelu popisujícího hlavičku protokolu Ethernet je na obrázku 4.3.



Obrázek 4.3: Abstraktní model popisující hlavičku protokolu Ethernet s naznačenými přechody pro popis protokolů vyšší vrstvy.

Protože je v jazycích využívaných pro popis hardwaru jako je VHDL nebo Handel-C jednoduché popsat stavový automat, zvolíme pro popis protokolů automat, který je definován v definici 4.3.1.

Automat vychází z Mealyho automatu, který má navíc k dispozici sadu proměnných, do kterých je možno uložit sémantické informace jako je například identifikátor protokolu vyšší vrstvy, délka volitelných položek nebo splnění určité podmínky (například hodnota pole *Version* u IPv4 a IPv6). S proměnnými lze také během přechodu provádět jednoduché operace (sčítání, porovnání apod.).

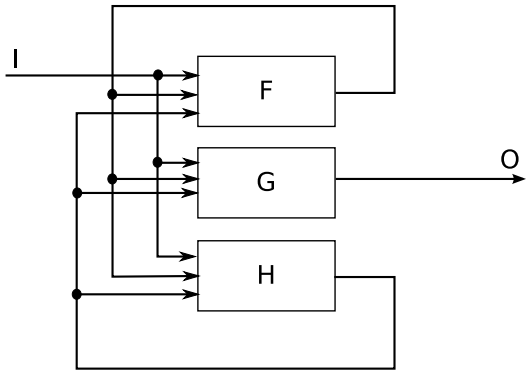
Přechod je možné provést na základě aktuálního stavu, hodnoty vstupu a hodnot uschovaných ve vnitřních proměnných. Na základě prováděného přechodu se budou vykonávat 2 typy akcí. První z nich je změna obsahu proměnných. Druhou je přiřazení výstupního symbolu.

Definice 4.3.1 *Navržený automat* je 10-tice $(S, S_0, I, O, U, U_0, V, F, G, H)$, kde:

- S je konečná neprázdná množina vnitřních stavů automatu
- S_0 je počáteční stav automatu, platí: $S_0 \in S$
- I je konečná neprázdná množina vstupních symbolů

- O je konečná neprázdná množina výstupních symbolů
- U je konečná neprázdná množina možných hodnot proměnné
- U_0 je počáteční hodnota všech proměnných, platí $U_0 \in U$
- V je konečná množina proměnných, kde každá proměnná může nabývat právě jedné hodnoty z množiny U
- $F(i, s, v)$ je přechodová funkce z aktuálního stavu s do následujícího stavu na základě vstupního symbolu i a hodnot proměnných v ($F\colon I \times S \times U^{|V|} \to S$)
- $G(i, s, v)$ je výstupní funkce, která přiřazuje výstupní symbol na základě aktuálního stavu s, vstupního symbolu i a hodnot proměnných v ($G\colon I \times S \times U^{|V|} \to O$)
- $H(i, s, v)$ je funkce, která přiřazuje proměnným novou hodnotu na základě aktuálních hodnot v, aktuálního stavu s a vstupního symbolu i ($H\colon I \times S \times U^{|V|} \to U^{|V|}$)

Blokové schéma tohoto automatu je uvedeno na obrázku 4.4.



Obrázek 4.4: Blokové schéma navrženého automatu

Z důvodu optimalizací doplníme model o data flow diagram, pomocí kterého je možné zjistit, které operace je možné předřadit před automat a tím zvýšit výslednou frekvenci, na které je možné analyzátor provozovat.

Nyní si popíšeme jakým způsobem je možné navržený model uplatnit pro analýzu protokolů.

V hlavičkách protokolů, které se budou pomocí automatu použitém v analyzátoru zpracovávat, se vyskytují mimo jiné pole o velikosti několika bitů, některá mohou mít dokonce velikost pouhého bitu. Proto bude tento automat zpracovávat jeden vstupní bit v každém kroku.

Vnitřní stavy automatu reprezentují aktuálně zpracovávaný bit. To znamená, že jsou charakterizovány aktuálně zpracovávaným protokolem, polem a pořadím zpracovávaného bitu v něm.

Množina vstupních symbolů odpovídá množině možných kombinací vstupních dat přicházejících do analyzátoru v jednom kroku. Množina výstupních symbolů odpovídá možné kombinaci příkazů na výstupním rozhraní analyzátoru v jednom kroku.

Množina proměnných obsahuje všechny proměnné, které jsou potřeba pro zpracování všech podporovaných protokolů. Do množiny hodnot, které mohou proměnné nabývat zařadíme pravdivostní a číselné hodnoty a také speciální hodnotu *nedefinovaná hodnota*, který

je také počáteční hodnotou všech proměnných. Pokud provedeme s proměnnou ve stavu *nedefinovaná hodnota* jakoukoliv operaci, výsledkem bude opět *nedefinovaná hodnota*.

Přechodová, výstupní funkce a funkce přiřazující hodnoty proměnným jsou definované na základě zpracovávané kombinace protokolu a extrahovaných dat.

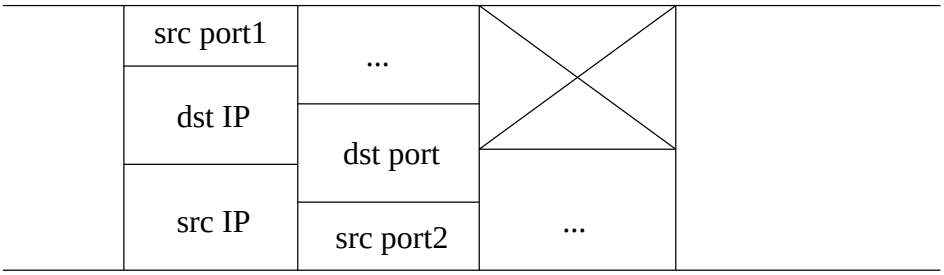
Protože platí dříve uvedená rovnice 4.1, musí být automat v analyzátoru schopen vykonat více kroků v rámci jednoho hodinového taktu.

4.4 Extrakce dat

Bylo navrženo několik možných protokolů definujících formát výstupních dat. Každý z těchto protokolů má oproti ostatním výhody a nevýhody. Následuje jejich přehled.

Nezarovnaná data bez identifikátorů

Extrahovaná data není nutné označovat identifikátory, pokud jsou pro každý rámec extrahovány stejné položky, nebo jsou extrahovány položky stejné velikosti a přijímající strana je na základě ostatních extrahovaných schopna tyto položky odlišit (např. zdrojový a cílový protokol TCP a UDP).



Obrázek 4.5: Výstupní protokol jako nezarovnaná data bez identifikátorů

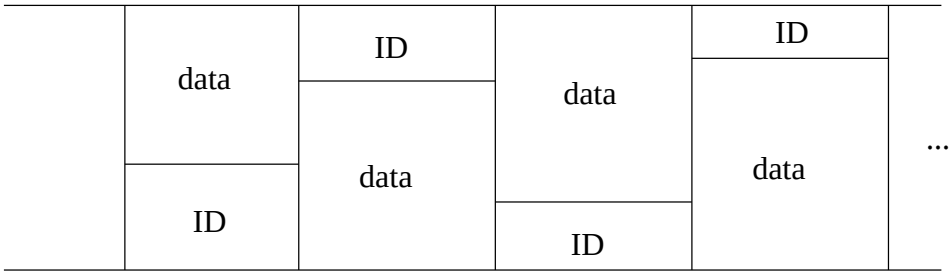
Na obrázku 4.5 je znázorněn formát rámce při použití tohoto přístupu. Rámec obsahuje zdrojovou a cílovou IP adresu, zdrojový port, který je rozdělen a přenášen ve dvou slovech, port cílový a další položky. Část posledního slova není využita.

Nevýhodou je, že přijímající strana musí znát sémantiku dat a ve své režii realizuje obdobu analýzy a extrakce realizované touto jednotkou. Další nevýhodou je, že toto řešení nedokáže snadno zajistit extrakci z rozšiřujících hlaviček protokolu IPv6, jejichž pořadí není ve specifikaci pevně určeno.

Výhodou řešení je nízká režie při přenosu rámce, kterou tvoří pouze nevyužitá část posledního slova rámce.

Nezarovnané identifikátory a nezarovnaná data

V případě, že není splněna podmínka pro možnost posílat neoznačená data nebo není možné zajistit, že přijímající jednotka bude neoznačená data schopna zpracovat, je nutné každou extrahovanou položku označit. Toto označení budeme nazývat identifikátorem. Existuje více možností, jak identifikátory do paketu umístit. První z nich je bez zarovnání identifikátoru i dat.



Obrázek 4.6: Výstupní protokol jako nezarovnané identifikátory následované nezarovnanými daty

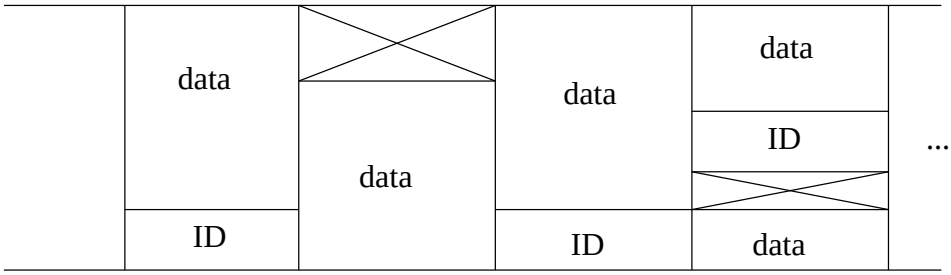
Obrázek 4.6 ukazuje příklad rámce formátovaného popisovaným způsobem. Identifikátory druhé i třetí položky jsou přenášeny ve dvou slovech, protože leží na jejich rozhraní.

Přijímající strana si sémantiku dat odvozuje podle hodnoty identifikátoru. Nevýhodou tohoto řešení je, že řídící logika v přijímající jednotce musí být složitá, protože je nutné identifikátory očekávat na všech pozicích.

Výhodou je nízká režie přenosu rámce, která je tvořena nevyužitou částí posledního slova a počtem bajtů, které v rámci zabírají identifikátory.

Zarovnané identifikátory a nezarovnaná data

Abychom zjednodušili analýzu rámce s extrahovanými položkami, kterou musí přijímající strana vykonávat, je možné identifikátory umístit jen na určité, předem dané pozice. Například je možné určit, že se identifikátor může nacházet jen v prvním bajtu (v prvních bajtech v případě použití identifikátorů delších než jeden bajt) v každém slově, nebo je možné povolit umístění začátku identifikátorů pouze na každý čtvrtý bajt slova apod.



Obrázek 4.7: Výstupní protokol jako zarovnané identifikátory následované nezarovnanými daty

Na obrázku 4.7 je příklad rámce s tímto uspořádáním. Aby byla dodržena podmínka, že se identifikátory mohou vyskytovat jen na určitých pozicích, jsou některé přenášené bajty nevyužity a slouží pouze jako výplň.

Nevýhodou popisovaného řešení je větší režie, oproti předcházejícím způsobům ukládání dat do výstupního rámce. Režii v tomto případě tvoří bajty obsazené identifikátory a bajty, které nejsou využité kvůli zarovnání identifikátorů, případně nevyužitá část posledního slova rámce.

Naopak výhodou je jednodušší zpracování rámce přijímající jednotkou, která očekává identifikátory a tím i datové položky jen na určitých pozicích. Takže je redukován počet nutných multiplexorů a přerovnávacích obvodů. Sníží se také počet multiplexorů a přerovnávacích obvodů v extrakční části navrhované jednotky, ale pro některé kombinace extrahovaných položek a zarovnání dat může být nutné použít vyrovnávací paměť, aby nedocházelo k pozastavení zpracování příchozího paketu a tím ke snižování propustnosti celé jednotky.

Řešení není vhodné pro aplikace, které potřebují extrahovat velké množství položek z hlaviček paketů, protože by mohla nastat situace, kdy by po přidání výplně byla nutné odeslat více výstupních dat, než bylo přijato vstupních dat a tím by docházelo ke snižování propustnosti celé jednotky.

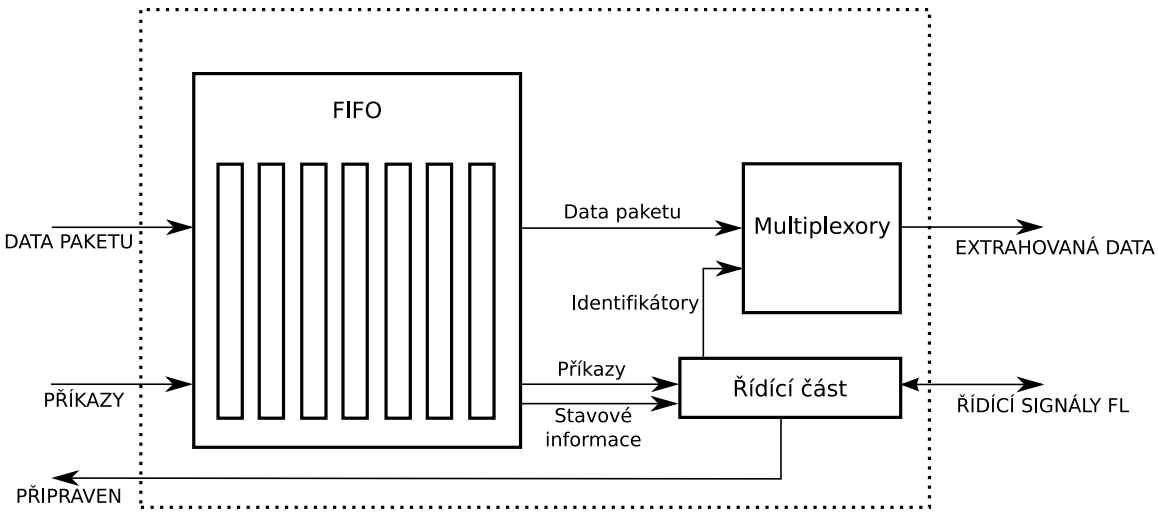
4.4.1 Architektura extrakčního modulu

Blokové schéma extrakčního modulu je zobrazeno na obrázku 4.8. Do modulu vstupují data zpracovávaného paketu a příkazy modulu provádějícího analýzu dat (extrahuj, přidej identifikátor, identifikátory) a z modulu vystupuje informace, zda je modul připraven přijmout další data. Modul má také standardní FrameLinkové [24] rozhraní obsahující extrahovaná data ve zvoleném formátu.

Modul se skládá ze tří částí. První část tvoří FIFO, do kterého se zapisují vstupní data a příkazy analyzátoru. Tyto informace jsou však zapsány jen pro ty data, ze kterých má být extrahován alespoň jeden bajt. FIFO umožňuje zlepšení propustnosti především pokud použitý výstupní protokol obsahuje velké množství identifikátorů položek, případně jsou identifikátory zarovnané. FIFO také odděluje extrakci od analýzy a tím brání vytvoření kritické cesty.

Druhá část modulu obsahuje sadu multiplexorů, na jejichž vstupy jsou přivedeny identifikátory a vstupní data. Multiplexory slouží ke tvorbě výstupního protokolu. Multiplexory jsou řízeny pomocí výstupu řídící jednotky, která tvoří poslední část modulu.

Řídící jednotka generuje na základě informací, které bajty se mají extrahovat a před které se má přidat identifikátor řídící signály pro multiplexory ve druhé části modulu. Jsou zde také generovány řídící signály protokolu FrameLink pro výstupní rozhraní. Protože



Obrázek 4.8: Blokové schéma extrakčního modulu

může dojít k nutnosti pozastavení vstupu v důsledku zaplnění FIFA, je generován také signál pro pozastavení vstupu, který je také součástí rozhraní analyzátoru.

V jednotce, která byla implementována v rámci této práce, extrakční modul pouze předává příkazy obdržené na vstupním rozhraní na svůj výstup. Tato konfigurace byla zvolena, protože je vhodná pro provedení základních testů analyzátoru a také proto, že nebylo doposud rozhodnuto, který z navržených výstupních protokolů bude v rámci projektu Liberouter [11] použit.

4.5 Konfigurace pomocí XML

Jedním ze základních požadavků na jednotku je její snadná konfigurovatelnost. Protože by ji měl být schopen vytvořit i obyčejný správce sítě, který nemá zkušenosti s aplikacemi pro platformu FPGA, bylo zvoleno, že bude popsána v jazyce XML [5]. Tento jazyk je běžně používán ve velké množině různých počítačových aplikací, takže je vysoká pravděpodobnost, že jej bude znát i běžný uživatel.

Popis pomocí XML byl vytvořen tak, aby byl intuitivní. Uživatel popisuje jednotlivé protokoly, které jsou navzájem odděleny. Pro každý protokol je možné definovat položky v něm obsažené a zvolit, zda se mají extrahovat. Uživatel má možnost si definovat proměnné jak pro celý protokol, tak pro jednotlivá pole. V rámci polí je možné definovat operace, které se mají při jeho zpracování provést. Typicky se jedná o uložení hodnoty pole do proměnné, případně porovnání hodnoty pole s jinou proměnnou, nebo konstantou. Pro přechod mezi protokoly byl navržen systém tzv. skoků, jejichž provedení je řízeno na základě hodnoty uložené v zadané proměnné.

Atributy elementů popisu se řídí DTD, které je uvedeno v příloze A. Tento dokument definuje, které elementy mají být pro popis jednotky použity a jaké jsou jejich atributy.

Kořenový element `hfe` obsahuje seznam protokolů v elementu `protocols`, kde každý protokol je popsán v příslušném elementu `protocol`. Každý protokol má jedinečný identifikátor definovaný v atributu `id`.

Pro popis protokolu může být nutné využít proměnné, jejichž seznam se nachází v elementu `variables`, který obsahuje jednotlivé proměnné. Ty jsou popsány pomocí elementu `variable`, který specifikuje typ a identifikátor dané proměnné.

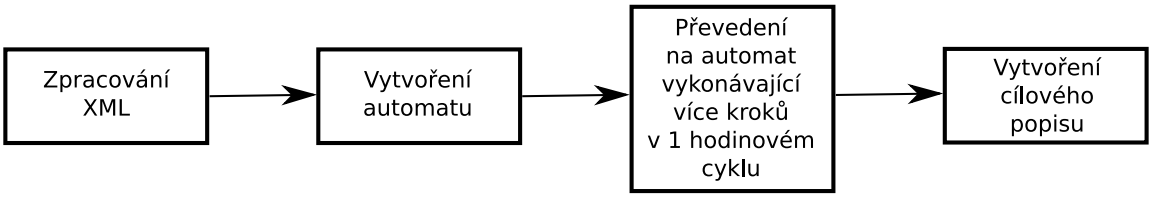
Pole v hlavičce protokolu jsou popsána pomocí elementů `field`. Každé pole může obsahovat vlastní sadu proměnných definovaných opět pomocí elementu `variables` a libovolný počet operací popsaných pomocí elementu `fieldop`. Počet bitů, které pole zabírá je dáno atributem `chunk` a `size`. Atribut `chunk` určuje o jakou délku dat může délka pole minimálně růst, pokud je vynechán použije se 1 bit. Atribut `size` potom udává, kolik takových skupin bitu pole zabírá. Toto rozdělení bylo zvoleno především proto, aby bylo možno popsat délku polí typu volitelných parametrů protokolu IPv4 a TCP. U IPv4 je `chunk` nastaven na 32 bitů a `size` je nastaven podle hodnoty v poli *Internet Header Length*. Pomocí atributu `extract` je možné poli přiřadit identifikátor, pod kterým má být extrahováno.

Za každým popisem pole je možné provést skok ve zpracování pomocí elementu `jump`. Skok se provádí na základě hodnoty proměnné specifikované v atributu `selectby`. Cíle skoku je možné definovat na základě porovnání na rovnost, menší, nebo rovno a větší, nebo rovno. Tento element lze využít především pro změnu zpracovávaného protokolu.

4.6 Generování jednotky

Pro převod popisu jednotky vytvořený pomocí XML do jazyka VHDL byl vytvořen program napsaný v jazyce Python. Tento jazyk byl vybrán především kvůli snadné práci s XML a podpoře objektového programování.

Jednotlivé kroky převodu jsou zachyceny na obrázku 4.9. Po načtení a zpracování souboru obsahující popis v XML je vytvořen model analýzy, reprezentovaný pomocí orientovaného grafu. Tento graf odpovídá automatu, který zpracovává pouze jeden vstupní bit v každém hodinovém cyklu. Aby bylo možné dosáhnout požadované propustnosti, je nutné převést reprezentaci na výsledný model, který je opět tvořen orientovaným grafem. Z tohoto grafu je již možné vytvořit automat vykonávající více kroků během jednoho hodinového cyklu. Výsledný automat je převeden do popisu v cílovém jazyce a uložen do souborů.



Obrázek 4.9: Kroky programu sloužícího k vytvoření modelu analýzy a cílového kódu automatu

Skript nejdříve vyhodnotí argumenty předané z příkazové řádky. Následuje zpracování zadaného souboru XML a vytvoření objektu třídy *HFEModel*, který představuje model zpracovávaných protokolů (obrázek 4.10). Obsahuje protokoly popsané objekty třídy *Protocol*. Protokol se skládá z objektů třídy *ProtocolItem*, které obsahují jedno pole protokolu (objekt třídy *Field*) a skoky, které je možné po zpracování tohoto pole provést (objekt třídy *Jump*). Každý objekt třídy *Field* obsahuje všechny operace, které se mají při zpracování tohoto pole provést. Tyto operace jsou reprezentovány hierarchii tříd *Operation* (obrázek 4.11).

Bázová třída *Operation* má tyto potomky:

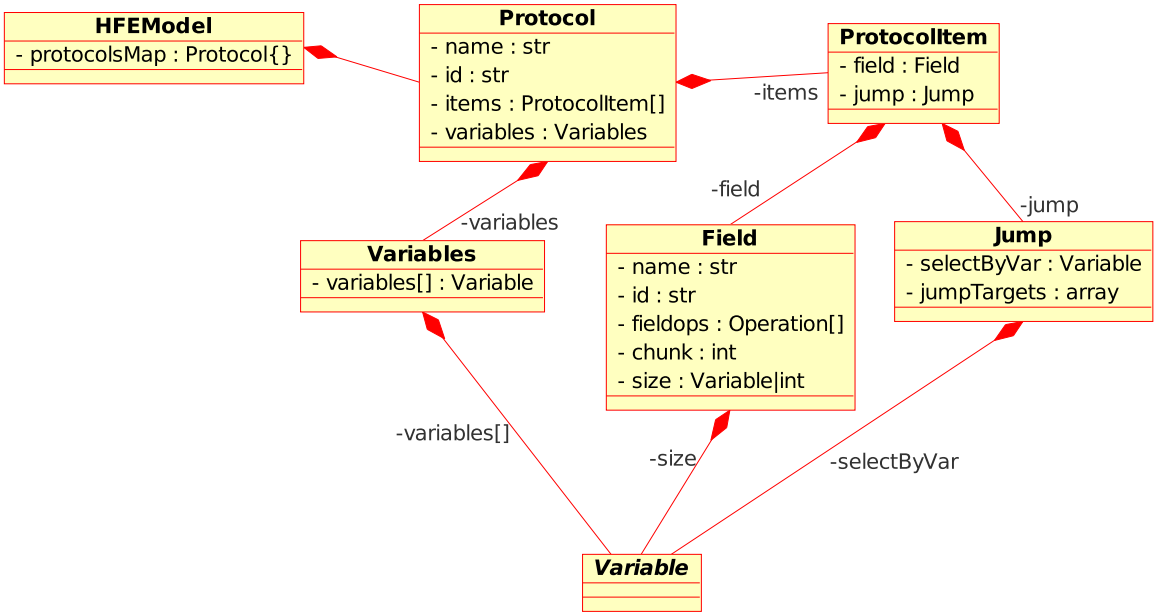
OperationNOP reprezentuje operaci, při které se nic neprovádí. Je implementovaná jako jedináček a objekt této třídy je využíván v situacích, kdy s bitem, pro který je použita, není potřeba vykonávat žádnou akci.

ConditionOperation je abstraktní třída reprezentující operace vyjadřující určitou podmínkou. Může se jednat o operace porovnání (rovnost; nerovnost; větší, nebo rovno; menší, nebo rovno), nebo booleovské operace spojující dvě jiné operace (v současné době je implementováno pouze booleovský součin – *ConditionBoolAnd*).

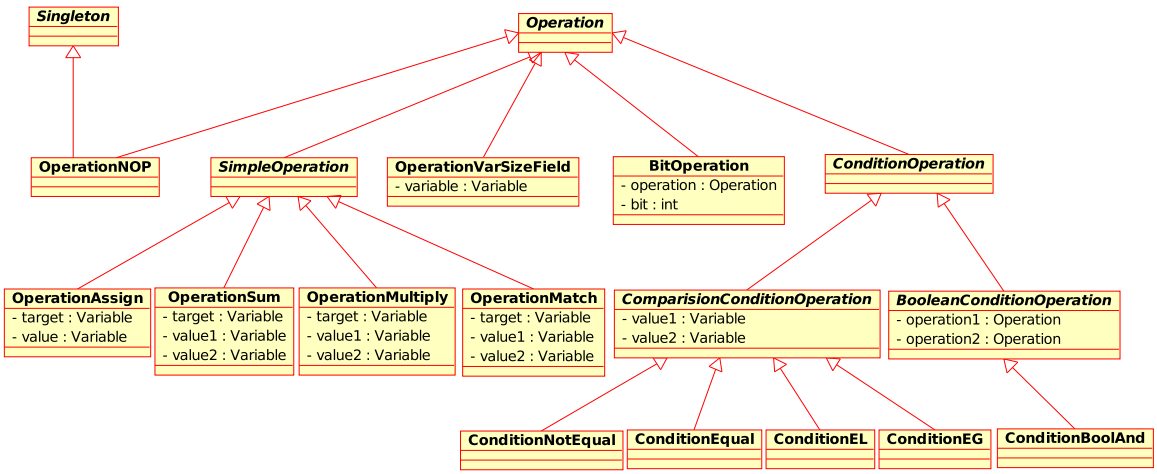
SimpleOperation je bázovou třídou pro základní (jednoduché) operace, které je možné s proměnnými vykonávat. Jde o třídy reprezentující sčítání, násobení, přiřazeni a test shody.

OperationVarSizedField se využívá při práci s položkami proměnné délky, jako jsou například *Options* v protokolu IPv4 a TCP.

BitOperation je třída využívaná pro operace týkající se vstupu analyzátoru. Tato třída obsahuje číslo vstupního bitu a operaci, která se s ním má provést.



Obrázek 4.10: Třídy využívané k popisu zpracovávaných protokolů



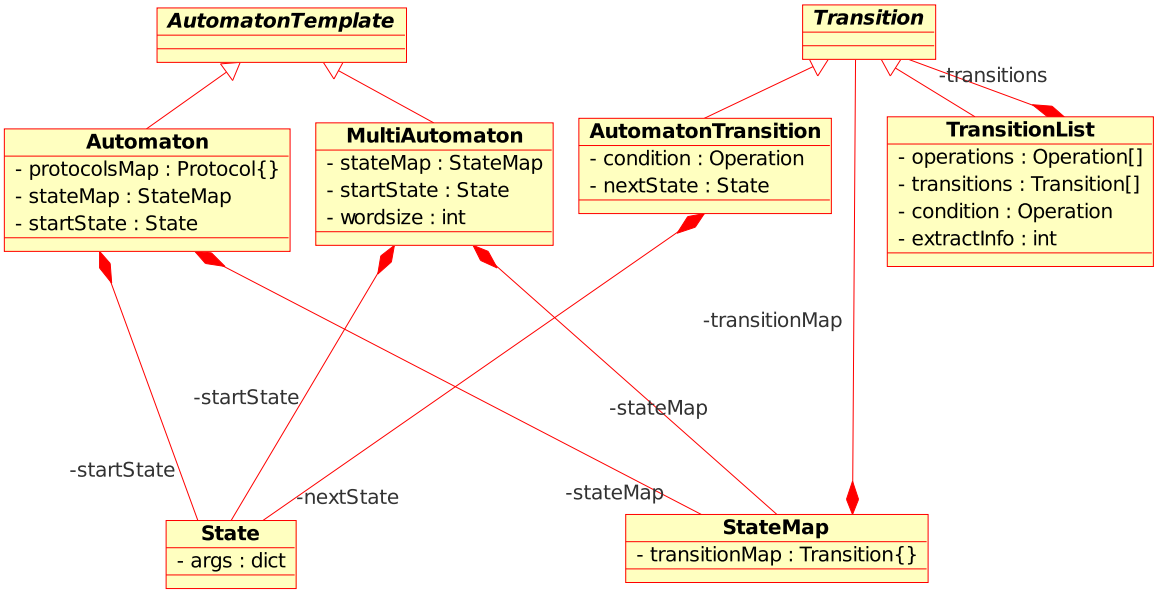
Obrázek 4.11: Hierarchie tříd představujících operace

Na základě vytvořeného modelu je zkonstruován automat (objekt třídy *Automaton*), který je schopen zpracovat jeden bit v každém kroku. Jednotlivé stavy jsou reprezentovány třídou *State*, přechody třídou *Transition* a operace řídící přechody i operace s proměnnými hierarchií tříd *Operation*. Tyto objekty jsou spolu propojeny pomocí třídy *StateMap*, která reprezentuje orientovaný graf, kde stavy reprezentují uzly a přechody reprezentují orientované hrany. Tento orientovaný graf reprezentuje nejdůležitější část objektu třídy *Automaton*.

Aby bylo možné provádět více kroků v jednom hodinovém taktu, musí být vytvořený model převeden na automat zpracovávající více bajtů v rámci jednoho taktu. K tomu slouží objekt třídy *MultiAutomaton*, který na základě automatu vytvořeném v předešlém kroku a

požadované vstupní šířky dat vytvoří nový orientovaný graf, také reprezentovaný objektem třídy *StateMap*, kde každý přechod může obsahovat operace pro každý vstupní bit. Tyto operace mohou být podmíněné splněním určité podmínky, aby bylo možné zpracovávat hlavičky různých protokolů v jednom hodinovém cyklu.

Třídy využívané při tvorbě základního automatu i automatu umožňujícího provádění více operací během jednoho hodinového cyklu jsou zobrazeny na obrázku 4.12.



Obrázek 4.12: Třídy využívané pro reprezentaci automatů, které představují navržený model analýzy hlaviček

Výstup programu v podobě kódu VHDL je realizován pomocí funkce *VHDLPrinter*. Ta vytvoří adresář, který byl specifikován pomocí argumentu z příkazové řádky, pro výstupní soubory. VHDL soubory jsou vytvořeny pomocí objektů tříd *HFETop*, zodpovědných za vytvoření souboru obsahující entitu jednotky a top level architekturu, a *Mfsm*, který vytváří popis modulu zajišťujícího analýzu paketu, který je realizován pomocí vytvořeného automatu převedeného do VHDL.

Přestože byl implementován pouze převod do VHDL, zvolený přístup k řešení dovoluje poměrně snadným způsobem přidat podporu i pro další jazyky určené pro popis hardwaru, například Handel-C. Pokud by bylo požadováno podporovat další výstupní jazyk, bylo by potřeba pouze nadefinovat novou funkci, která by dokázala převést objekt třídy *StateMap* do nového cílového jazyka. Dále by bylo potřeba přidat speciální metody objektům tříd *State*, *Variable*, *Constant* a hierarchii tříd *Operation* pro reprezentaci objektů v přidávaném výstupním jazyce.

VHDL soubor obsahující analyzátor popisuje výsledný automat behaviorálním popisem jako jeden proces. V tomto procesu jsou použity proměnné, které reprezentují proměnné automatu. Při syntéze pak syntetizátor podle potřeby pro tyto proměnné vytvoří registry, které jsou potřeba pokud je obsah proměnné využíván i v jiném hodinovém cyklu, než se objevuje hodnota ukládaná do proměnné na vstupu.

Kapitola 5

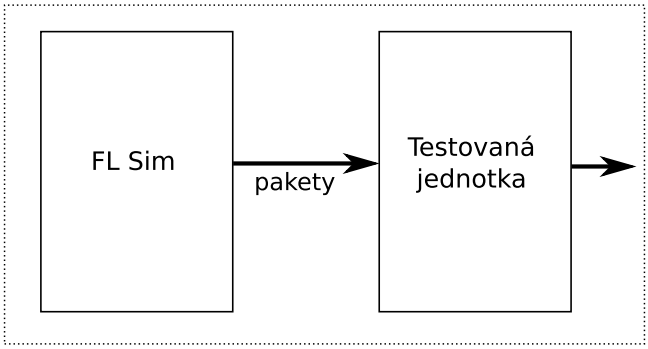
Výsledky

Než bude možné vygenerovanou jednotku využít, je potřeba ověřit, že pracuje správně. K tomuto účelu slouží funkční simulace. Pomocí funkčních simulací je možné sledovat hodnoty jednotlivých signálů uvnitř jednotky i na jejím vstupu a výstupu. Proto si můžeme ověřit, že jednotka pracuje tak, jak je požadováno, případně je možné nalézt chyby.

Je také nutné zjistit kolik zdrojů jednotka potřebuje pro svou činnost a jaká je maximální frekvence hodinového signálu, při které jednotka pracuje správně. Tyto údaje je možné zjistit pomocí syntézy.

5.1 Funkční simulace

Provedením funkčních simulací ověříme správnou funkčnost jednotky. Aby mohl být tento úkol splněn, bylo vytvořeno prostředí pro ověření navržené jednotky (obrázek 5.1), ve kterém byla využita komponenta *FL Sim* vytvořená v rámci projektu Liberouter [11]. Tato komponenta slouží k odeslání FrameLinkových rámců popsaných v textovém souboru.



Obrázek 5.1: Testovací design jednotky vytvořený pro provedení funkčních simulací

Pro provedení funkčních simulací byl využit program ModelSim SE 6.2g firmy Mentor Graphics. Pomocí komponenty FL Sim byly na vstup testované jednotky odesílány síťové pakety obsahující různé síťové protokoly a byla zkoumána správná funkčnost jednotky.

5.2 Syntéza

Syntéza je proces, při kterém je vstupní VHDL kód převeden na hardwarovou reprezentaci. Protože je tato reprezentace závislá na použité architektuře FPGA a jednotka bude využívané především na hradlovém poli Virtex-5, byly výsledky zjišťovány pro FPGA XC5VLX110.

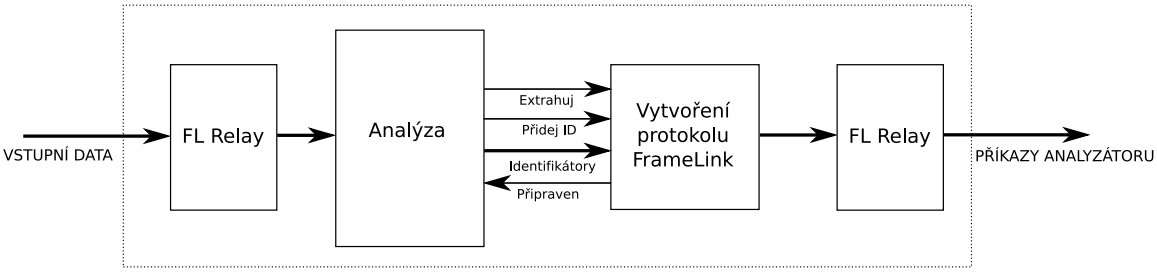
Syntéza jednotky byla prováděna pomocí nástrojů XST verze 9.1.03i firmy Xilinx a Precision RTL Synthesis verze 2007a.11 firmy Mentor Graphics.

5.3 Architektura testované jednotky

Pro testování vytvořené jednotky byla na vstup a výstup jednotky přidaná komponenta *FL Relay* vytvořena v rámci projektu Liberouter [11]. Tato komponenta slouží k rozdělení cesty mezi dvěma jednotkami na dvě části tím, že mezi komponenty vloží registr. Tím se vyhneme možné kritické cestě, která by mohla vzniknout mezi sousedními jednotkami, pokud by se použily jako součást většího celku, jako je například monitorovací sonda FlowMon [6]. Vstupní i výstupní rozhraní komponenty FL Relay využívá protokol FrameLink, takže jejím použitím neporušíme část zadání týkající se požadovaného rozhraní jednotky.

Využití komponent FL Relay nám také pomůže při zkoumání výsledků po syntéze. Pokud bychom modul analýzy připojili přímo na vstupní porty, syntetizátor by nemusel najít možnou kritickou cestu začínající na vstupu jednotky nebo končící na jejím výstupu.

Blokové schéma konečné podoby architektury testované jednotky je uvedeno na obrázku 5.2. Jak bylo zmíněno dříve, místo extrakčního modulu se pouze vytváří protokol FrameLink z výstupního rozhraní analyzátoru.



Obrázek 5.2: Konečná architektura jednotky

Pro fukční simulace i syntézu byla zvolena konfigurace jednotky, která podporuje protokoly Ethernet, IPv4, IPv6, TCP a UDP. Konfigurace jednotky umožňovala zpracovávat pakety, ve kterých se vyskytuje více protokolů síťové vrstvy (IPv4, IPv6).

5.4 Výsledky

Funkční simulace prokázaly, že jednotka analyzuje vstupní data správně. Příklady zpracování paketů je možné nalézt v příloze B.

Výsledky syntézy pro hradlové pole Virtex-5 XC5VLX110 jsou shrnuty v tabulce 5.1. Pro vstupní datové šířky 64, 128 a 256 bitů jsou pro oba syntetizátory uvedeny obsazené zdroje, vyjádřené v počtu obsazených buněk CLB, odhad maximální frekvence hodinového

signálu, na které je možné jednotku používat, a teoretická maximální propustnost jednotky, která byla vypočítaná pro zjištěnou frekvenci hodinového signálu podle vzorce 4.1.

Syntetizátor	Precision			XST		
Vstupní šířka [b]	64	128	256	64	128	256
Počet buněk CLB	122	293	1672	182	367	1287
Maximální frekvence [MHz]	190	107	54	208	122	76
Maximální propustnost [Gb/s]	12,2	13,7	13,8	13,3	15,6	19,5

Tabulka 5.1: Tabulka s výsledky syntézy pro vstupní šířku 64, 128 a 256 bitů

Tabulka 5.1 ukazuje, že se podařilo splnit cíl a jednotka je schopna analyzovat pakety na síti o rychlosti 10 Gb/s. Také je vidět, že s rostoucí datovou šířkou vstupu klesá frekvence a pro jednotu se vstupem o šířce 256 bitů je menší než 100 MHz.

Kapitola 6

Závěr

Hlavním cílem této práce bylo vytvoření návrhu hardwarové jednotky, která dokáže analyzovat obsah paketů přijatých ze sítě a extrahovat položky z jejich hlaviček tak, aby byla použitelná v projektu Liberouter [11]. Tento cíl byl splněn.

Na počátku řešení práce jsem se seznámil s technologii FPGA a kartou Combo6X, což mi umožnilo vytvořit počáteční návrh požadované jednotky. Současně jsem se zabýval studiem síťových protokolů jednotlivých vrstev referenčního modelu ISO/OSI, které bylo potřebné především pro návrh automatu zajišťujícího analýzu dat přenášených v paketech. Znalost používaných protokolů a obsahů jejich hlaviček mě umožnila identifikovat potenciální problémy, které mohou při analýze paketů nastat, a pomohla mi při návrhu struktury XML dokumentu pro specifikaci protokolů a způsobu extrakce položek z hlaviček paketů.

V rámci analýzy možností hardwarové akcelerace extrakce položek v hlavičkách paketů bylo navrženo několik možných formátu pro extrahovaná data, které byly diskutovány v kapitole 4.4. Výběr použitého formátu závisí na aplikaci, ve které bude jednotka použita, především na počtu extrahovaných položek a jejich velikosti.

Základem návrhu vhodné hardwarové architektury pro zpracování paketu v sítích s propustností 10–40 Gb/s je navržený model popisu dat obsažených v paketech. Jeho uplatnění umožnilo zpracování několika vstupních bajtů v jednom hodinovém cyklu, ve kterých se může nacházet i více hlaviček protokolů. Využití navrženého modelu vedlo také ke snížení počtu hardwarových zdrojů v porovnání se stávajícími řešeními.

Po dohodě s vedoucím se práce zaměřila na návrh a implementaci modulu pro analýzu paketu, který má rozhodující vliv na maximální frekvenci výsledného řešení. Extrakční část byla implementována pouze pro účely ověření správné funkce modulu, který provádí analýzu paketů. Proto nedošlo k ověření funkce jednotky ve vybraných síťových aplikacích, ale pouze pomocí funkčních simulací.

Implementované řešení je vhodné pro analýzu paketů v sítích o rychlosti 10 Gb/s, což potvrdily výsledky syntézy jednotky, které ukazují, že je možné jednotku použít pro zpracování datového toku s propustností 15–20 Gb/s při využití FPGA s technologií Virtex-5.

Pro provoz jednotky na FPGA Virtex-2 s frekvencí alespoň 100 MHz je potřeba jednotku dále optimalizovat. Například je možné přesunout operace pracující pouze se vstupem před automat a tím zvýšit frekvenci, na jaké lze automat používat. Další zvýšení propustnosti je možné dosáhnout využitím paralelního zpracování zapojením několika jednotek. Tímto řešením je možné dosáhnout i propustnosti 100 Gb/s.

Literatura

[1] Braun, F.; Lockwood, J.; Waldvogel, M.: Protocol Wrappers for Layered Network Packet Processing in Reconfigurable Hardware. *IEEE Micro*, 1 2002: s. 66–74, ISSN 0272-1732.

[2] Claise, B.: RFC 3954 Cisco Systems NetFlow Services Export Version 9. [online], Vytvořeno v říjnu 2004 [cit. 2008-04-12].
URL <http://tools.ietf.org/html/rfc3954>

[3] Dedek, T.; Martínek, T.; Marek, T.: High Level Abstraction Language as an Alternative to Embedded Processors for Internet Packet Processing in FPGA. In *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on*, 2007, ISBN 978-1-4244-1060-6, s. 648–651.

[4] Deering, S.; Hinden, R.: RFC 2460 Internet Protocol, Version 6 (IPv6) Specification. [online], Vytvořeno v prosinci 1998 [cit. 2008-04-12].
URL <http://tools.ietf.org/html/rfc2460>

[5] Extensible Markup Language (XML). [online], Poslední modifikace 2008-01-30 [cit. 2008-04-09].
URL <http://www.w3.org/XML/>

[6] WWW stránka FlowMon sondy. [online], [cit. 2008-04-09].
URL <http://www.flowmon.org/>

[7] IEEE: *Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications*. Ieee std 802.3 edition vydání, 2002.
URL <http://standards.ieee.org/getieee802/download/802.3-2002.pdf>

[8] Intrusion detection system. [online], Poslední modifikace 2008-04-08 [cit. 2008-04-09].
URL <http://en.wikipedia.org/wiki/Intrusion-detection_system>

[9] Kent, S.: RFC 4302 IP Authentication Header. [online], Vytvořeno v prosinci 2005 [cit. 2008-04-12].
URL <http://tools.ietf.org/html/rfc4302>

[10] Kent, S.: RFC 4303 IP Encapsulating Security Payload (ESP). [online], Vytvořeno v prosinci 2005 [cit. 2008-04-12].
URL <http://tools.ietf.org/html/rfc4303>

[11] WWW stránka projektu Liberouter. [online], [cit. 2008-02-19].
URL <http://www.liberouter.org/>

[12] LocalLink Interface Specification. [online], [cit. 2008-02-21].
URL <http://www.xilinx.com/products/design_resources/conn_central/locallink_member/sp006.pdf>

[13] McCann, J.; Deering, S.; Mogul, J.: RFC 1981 Path MTU Discovery for IP version 6. [online], Vytvořeno v srpnu 1996 [cit. 2008-04-12].
URL <http://tools.ietf.org/html/rfc1981>

[14] Mikušek, P.: *Návrh a implementace procesní jednotky pro analýzu vstupních paketů*. Bakalářská práce, FIT VUT v Brně, 2005.

[15] Multiprotocol Label Switching. [online], Poslední modifikace 2008-02-26 [cit. 2008-03-01].
URL <http://en.wikipedia.org/wiki/MPLS>

[16] OSI model. [online], Poslední modifikace 2008-02-07 [cit. 2008-02-07].
URL <http://en.wikipedia.org/wiki/OSI_model>

[17] Postel, J.: RFC 768 User Datagram Protocol. [online], Vytvořeno v srpnu 1980 [cit. 2008-04-12].
URL <http://tools.ietf.org/html/rfc768>

[18] Quittek, J.; Zseby, T.; Claise, B.; aj.: RFC 3917 Requirements for IP Flow Information Export (IPFIX). [online], Vytvořeno v říjnu 2004 [cit. 2008-04-12].
URL <http://tools.ietf.org/html/rfc3917>

[19] Ramakrishnan, K.; Floyd, S.; Black, D.: RFC 3168 The Addition of Explicit Congestion Notification (ECN) to IP. [online], Vytvořeno v září 2001 [cit. 2008-04-12].
URL <http://tools.ietf.org/html/rfc3168>

[20] RFC. [online], Poslední modifikace 2008-02-08 [cit. 2008-04-09].
URL <http://cs.wikipedia.org/wiki/RFC>

[21] RFC 791 Internet Protocol. [online], Vytvořeno v září 1981 [cit. 2008-04-11].
URL <http://tools.ietf.org/html/rfc791>

[22] RFC 793 Transmission Control Protocol. [online], Vytvořeno v září 1981 [cit. 2008-04-12].
URL <http://tools.ietf.org/html/rfc793>

[23] Sada protokolů Internetu. [online], Poslední modifikace 2008-01-12 [cit. 2008-02-07].
URL <http://cs.wikipedia.org/wiki/Sada_protokol%C5%AF_Internetu>

[24] Tobola, J.: *Platforma pro rychlý vývoj síťových zařízení*. Diplomová práce, FIT VUT v Brně, 2007.

[25] Xilinx: *MicroBlaze Processor Reference Guide*. 4 2005.

Příloha A

DTD pro popis analýzy a extrakce v XML

```
<!-- Kořenový element -->
<!ELEMENT hfe (protocols)>

<!-- Parsování protokolů -->
<!ELEMENT protocols (protocol*)>
<!ATTLIST protocols
 <!-- string: id počátečního protokolu -->
 start IDREF #REQUIRED
>

<!-- Parsovatelný protokol -->
<!ELEMENT protocol (EMPTY | (variables?, (field, jump?)+))>
<!ATTLIST protocol
 <!-- string: jméno protokolu -->
 name CDATA #IMPLIED
 <!-- string: jednoznačný identifikátor protokolu -->
 id ID #REQUIRED
>

<!-- Proměnné pro protokol -->
<!ELEMENT variables (variable+)>

<!-- Proměnná, do které je možno dočasně uchovat hodnotu -->
<!ELEMENT variable EMPTY>
<!ATTLIST variable
 <!-- string: jméno proměnné - slovní popis -->
 name CDATA #IMPLIED
 <!-- string: jednoznačný identifikátor proměnné -->
 id ID #REQUIRED
 <!-- string: textový řetězec identifikující typ proměnné -->
 type (int | boolean) #REQUIRED
>
```

```
<!-- Pole v hlavičce protokolu -->
<!ELEMENT field (varibles?, fieldop*)>
<!ATTLIST field
 <!-- string: Jméno pole - slovní popis -->
 name CDATA #IMPLIED
 <!-- string: jednoznačný identifikátor pole -->
 id ID #REQUIRED
 <!-- integer: velikost chunku dat v bitech, pokud je atribut vynechán,
               předpokládá se 1 -->
 chunk NMTOKEN "1"
 <!-- integer/variable_id: počet chunků v poli -->
 size NMTOKEN #REQUIRED
 <!-- integer: 0-255, idenetifikátor extrahovaného políčka, pokud není
               uvedeno, pole se neextrahuje -->
 extract NMTOKEN #IMPLIED
>

<!-- Operace, která se má s polem provést -->
<!ELEMENT fieldop (fieldopparam*)>
<!ATTLIST fieldop
 <!-- string: Název operace, která se má provést -->
 ref (op_sum | op_assign | op_match) #REQUIRED
<!--
Podporované operace:
op_multiply:
param target (jméno proměnné typu int)
param value1 (celé číslo/jméno proměnné typu int)
param value2 (celé číslo/jméno proměnné typu int)
Nastaví proměnnou target na výsledek výrazu "value1 * value2"

op_sum:
param target (jméno proměnné typu int)
param value1 (celé číslo/jméno proměnné typu int)
param value2 (celé číslo/jméno proměnné typu int)
Nastaví proměnnou target na výsledek výrazu "value1 + value2"

op_assign:
param target (jméno proměnné typu int)
param value (celé číslo/jméno proměnné typu int)
Nastaví proměnnou target na hodnotu value
```

```
op_match:
param target (jméno proměnné typu boolean)
param value1 (celé číslo/jméno proměnné typu int)
param value2 (celé číslo/jméno proměnné typu int)
Nastaví proměnnou target na výsledek porovnání "value1 == value2"
-->
>

<!-- Parametr operace -->
<!ELEMENT fieldopparam EMPTY>
<!ATTLIST fieldopparam
 <!-- string: Jméno parametru -->
 name NMTOKENS #REQUIRED
 <!-- string: Identifikátor proměnné/pole, který se má předat jako
              parametr -->
 ref IDREF #IMPLIED
 <!-- string/integer: Hodnota, která se má předat jako parametr -->
 value NMTOKEN #IMPLIED
 <!-- Poznámka: Je povolen právě jeden z atributů ref a value -->
>

<!-- Kam se má skočit -->
<!ELEMENT jump (case+, others?)>
<!ATTLIST jump
 <!-- string: Identifikátor proměnné, na jejímž základě se rozhodne
              o cíli skoku -->
 selectby IDREF #REQUIRED
>

<!-- Za jakých podmínek se má skok vykonat -->
<!ELEMENT case EMPTY>
<!ATTLIST case
 <!-- string/integer: Vykonej skok, pokud je proměnná rovna této
                      hodnotě -->
 equal NMTOKEN #IMPLIED
 <!-- string/integer: Vykonej skok, pokud je hodnota proměnné rovna
                      této hodnotě, nebo je větší -->
 min NMTOKEN #IMPLIED
 <!-- string/integer: Vykonej skok, pokud je hodnota proměnné rovna této
                      hodnotě, nebo je menší -->
 max NMTOKEN #IMPLIED
 <!-- string: Identifikátor cíle skoku (protokol, v případě, že se má
              začít zpracovávat nový protokol, identifikátor pole,
              kterým se má pokračovat v případě, že jsme již začali
              zpracovávat nový protokol a zpracovali jsme některá jeho
              pole) -->
 ref IDREF #REQUIRED
>
```

```
<!-- Vykoná skok, pokud není splněna žádná z předchozích podmínek -->
<!ELEMENT others EMPTY>
<!ATTLIST others
 <!-- string: Identifikátor cíle skoku (protokol, v případě, že se má
              začít zpracovávat nový protokol, identifikátor pole,
              v případě, že jsme již začali zpracovávat nový protokol
              a toto pole a předcházející jsou již zpracovány) -->
 ref IDREF #REQUIRED
>
```

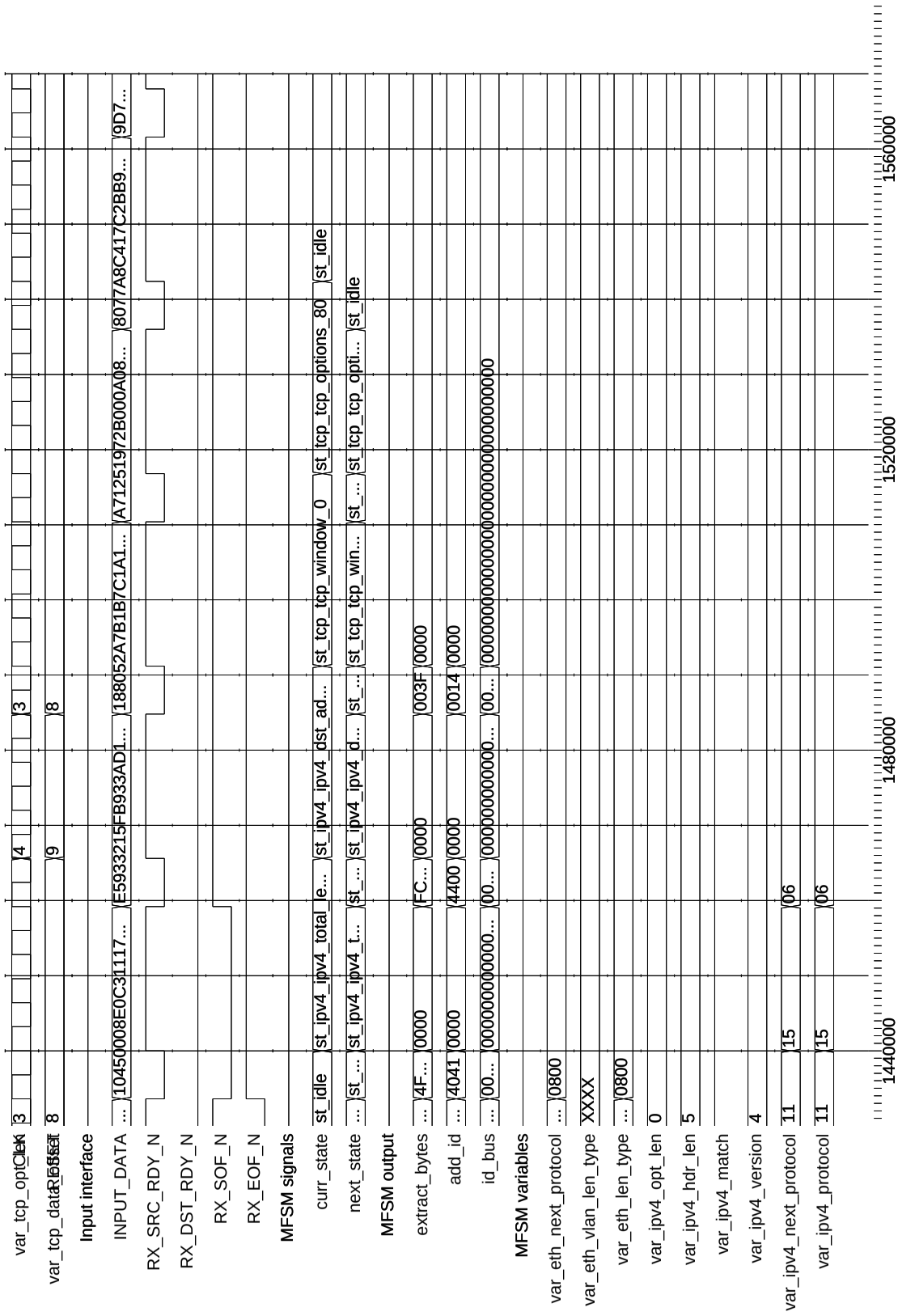
Příloha B

Simulace

Časové diagramy zachycují průběh funkčních simulací. Správnou funkci jednotky potvrzují správné posloupnosti stavů a přechodů automatu a také správně nastavené signály určující, které bajty se mají extrahovat, před které má být přidán identifikátor a hodnota přidávaných identifikátorů.

- Na obrázku B.1 je zobrazeno zpracování paketu obsahujícího protokol Ethernet, IP verze 4 a UDP. V cyklu, ve kterém přišlo první slovo, je zpracována celá hlavička protokolu Ethernet a část hlavičky protokolu IPv4. Hlavička protokolu IPv4 je obsažena ještě ve druhém a částečně ve třetím slově. Celá hlavička protokolu UDP se nachází ve třetím slově. Obsažená data protokolu UDP nejsou zpracovávaná. Automat přechází do výchozího stavu, ve kterém čeká na začátek dalšího paketu.
- Časový diagram na obrázku B.2 ukazuje zpracování paketu obsahujícího protokol Ethernet, IPv4 a TCP. Podobně jako v předchozím případě jsou hlavičky protokolů Ethernet a IPv4 obsaženy v prvních třech slovech paketu. Následuje analýza hlavičky protokolu TCP, data tohoto protokolu nejsou analyzována a automat přechází do výchozího stavu.
- Zpracování paketu obsahujícího protokoly Ethernet, IPv6 a TCP je zobrazeno na obrázku B.3. Hlavička protokolu Ethernet je zpracována opět v prvním slově paketu. Hlavička protokolu IPv6 je delší než hlavička protokolu IPv4, proto je zpracovávaná déle. Její analýza je dokončena teprve s příchodem čtvrtého slova paketu. Po dokončení analýzy hlavičky protokolu TCP automat přechází do výchozího stavu.

Obrázek B.1: Zpracování paketu obsahujícího protokol Ethernet, IPv4 a UDP



Obrázek B.2: Zpracování paketu obsahujícího protokol Ethernet, IPv4 a TCP

Obrázek B.3: Zpracování paketu obsahujícího protokol Ethernet, IPv6 a TCP