
**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**
**BRNO UNIVERSITY OF TECHNOLOGY**

**FAKULTA INFORMAČNÍCH TECHNOLOGIÍ**
**FACULTY OF INFORMATION TECHNOLOGY**

**ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ**
**DEPARTMENT OF COMPUTER SYSTEMS**


# VYUŽITÍ JAZYKA P4 K POPISU AKCELEROVANÉHO ZAŘÍZENÍ NA OCHRANU PŘED DOS ÚTOKY

**P4 LANGUAGE-BASED DESCRIPTION OF ACCELERATED DEVICE AGAINST DOS ATTACKS**

**DIPLOMOVÁ PRÁCE**
**MASTER'S THESIS**


**AUTOR PRÁCE** — **Bc. MÁRIO KUKA**
**AUTHOR**

**VEDOUCÍ PRÁCE** — **Ing. JAN KUČERA**
**SUPERVISOR**



**BRNO 2019**

**Vysoké učení technické v Brně**
Fakulta informačních technologií

Ústav počítačových systémů (UPSY) Akademický rok 2018/2019

# Zadání diplomové práce


Student: **Kuka Mário, Bc.**
Program: Informační technologie Obor: Počítačové a vestavěné systémy
Název: **Využití jazyka P4 k popisu akcelerovaného zařízení na ochranu před DoS útoky**
**P4 Language-Based Description of Accelerated Device against DoS Attacks**
Kategorie: Počítačová architektura

Zadání:

1. Nastudujte vlastnosti jazyka P4 pro popis síťových aplikací, seznamte se s jeho kompilátorem a možnostmi převodu popisu síťového zařízení z jazyka P4 do jazyka VHDL, respektive technologie FPGA.
2. Seznamte se s problematikou útoků typu odepření služby, hardwarově akcelerovanými síťovými kartami rodiny COMBO a zařízením vyvíjeným v rámci sdružení CESNET pro ochranu před těmito útoky.
3. V jazyce P4 popište jednotlivé součásti tohoto hardwarově akcelerovaného zařízení.
4. V simulačním prostředí a s využitím dostupných nástrojů nejprve ověřte funkčnost takto vytvořeného popisu a následně proveďte jeho transformaci do hardwarové architektury.
5. Ověřte funkčnost a výkonové parametry výsledné hardwarové architektury, proveďte vyhodnocení výsledků.
6. V závěru diskutujte dosažené výsledky a možnosti dalšího pokračování práce.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1 až 3 zadání.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Kučera Jan, Ing.**
Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 22. května 2019
Datum schválení: 26. října 2018

# Abstrakt

Diplomová práca sa zaoberá vývojom sieťového zariadenia na ochranu pred (D)DoS útokmi prostredníctvom vysoko úrovňového jazyka P4. Hlavnou úlohou je vytvorenie flexibilného zriadenia s využitím jazyka P4. To nám umožní rýchlo reagovať na stále nové a zložitejšie útoky (D)DoS. Návrh zariadenia sa zaoberal prevodom jednotlivých častí firmwaru zariadenia do jazyka P4. Následne bol uskutočnený návrh celého firmwaru zariadenia. Návrh bol uskutočnený s ohľadom na limity aktuálne dostupných prekladových nástrojov jazyka P4. Navrhnuté zariadenie bolo implementované pre hardvérové akcelerátory s technológiu FPGA. Zariadenie bolo testované v laboratórnych podmienkach, kde bola overená jeho funkčnosť a výkonnosť. Zariadenie bude nasadené v sieťovej infraštruktúre CESNET.



# Abstract

This thesis describes the development of a networking device used to defend against (D)DoS attacks using P4 language. The main purpose was to design flexible device using P4 language based on already existing device, this would allow us to quickly react and respond to new more complex DDoS attacks. The design of the device dealt with the transfer of individual parts of the firmware into the P4 language. Subsequently, the entire device firmware was designed for hardware accelerators with FPGA technology. The firmware had been designed with respect to the limitations of current P4 language compilers. The device has been tested under laboratory conditions for functionality and performance. The device will be deployed in the network infrastructure of CESNET.



# Kľúčové slová

DCPro, (D)DoS Protector, COMBO, CESNET, Hardvérová Akcelerácia, FPGA, Vysoko rýchlostné siete, 100 Gbps, 200 Gbps, Spracovanie sieťových dát, P4, DoS, DDoS, NFB

# Keywords

DCPro, (D)DoS Protector, COMBO, CESNET, Hardware Acceleration, FPGA, High-speed Networks, 100 Gbps, 200 Gbps, Network Traffic Processing, P4, DoS, DDoS, NFB

# Citácia

KUKA, Mário. *Využití jazyka P4 k popisu akcelerovaného zařízení na ochranu před DoS útoky.* Brno, 2019. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jan Kučera

# Využití jazyka P4 k popisu akcelerovaného zařízení na ochranu před DoS útoky

## Prehlásenie

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne pod vedením pána Ing. Jana Kučery. Ďalšie informácie mi poskytli pracovníci vedecko výskumnej skupiny Liberouter pod združeným CESNET. Uviedol som všetky literárne zdroje a publikácie, z ktorých som čerpal.

.....
Mário Kuka
20. mája 2019

## Poďakovanie

V prvom rade by som chcel venovať veľké poďakovanie vedúcemu práce Ing. Janu Kučerovi za odbornú pomoc, ochotu a trpezlivosť nielen pri tvorbe tejto práce ale taktiež za všetkú ostanú pomoc pri tvorbe ďalších materiálov prezentujúcich výsledky tejto práce. Ďalej by som chcel poďakovať pracovníkom z oddelenia nástrojov pre monitorovanie a konfiguráciu so združením CESNET za všetku poskytnutú odbornú pomoc a podporu.

# Obsah

# Kapitola 1

# Úvod

So súčasným rozvojom moderných technológií a ich snahou čo najefektívnejšie prepojiť celú spoločnosť neustále narastajú požiadavky na zefektívňovanie a skvalitňovanie komunikačných prostriedkov. Jedným z hlavných prostriedkov, ktoré nám dneska umožňujú komunikovať, je rozvoj osobných počítačov a s tým spojený rozvoj počítačových sietí. Prostredníctvom týchto sietí si tak môžu užívatelia vymieňať správy, realizovať hovory, zdieľať súbory, prípadne využívať sieťové prostriedky ako sú sieťové tlačiarne, súborové servery a ďalšie množstvo služieb, ktoré sú poskytované prostredníctvom počítačových sietí. Neustále zvyšujúce sa nároky na množstvo prenášaných informácií má za následok neustáli rozvoj počítačových sietí. V súčasnosti je tak trendom neustále zvyšovanie rýchlostí počítačových sietí a využívanie nových dostupných technológií ako je 100 GbE na hlavných sieťových linkách a serverových farmách. Tento neustály rozvoj sa však premieta aj do oblastí bezpečnosti a monitorovania počítačových sietí. Ich úlohou je udržovať tieto siete vo funkčnom stave a chrániť ich pred neustále hroziacim nebezpečím. Jedným typom z týchto hrozieb sú útoky zamerané na odoprenie služby DoS (Denial of Service), ktorých úlohou je zamedzenie prístupu legitímnych užívateľov k poskytovaným službám prostredníctvom počítačových sietí. Cieľom niektorých DoS útokov (napr. SYN flood) býva samotný server s cieľom vyčerpať jeho zdroje (pamäť, výpočtové prostriedky), čím dôjde k jeho nedostupnosti pre legitímnych užívateľov. Útoky typu DoS častokrát bývajú vedené z viacerých zariadení. Tieto útoky označujeme ako distribuované útoky typu DoS, skrátene (D)DoS. Jedným z najrozšírenejších útokov tohoto typu (D)DoS sú v dnešnej dobe takzvané volumetrické útoky. Cieľom týchto útokov typicky býva samotná sieťová infraštruktúra (počítačová sieť organizácie) a nie iba jedno koncové zariadenie.

S neustálim zvyšovaním rýchlosti počítačových sietí sa zvyšuje sila útokov (D)DoS. Týmto útokom sa tak musia neustále prispôsobovať aj samotné zariadenia na ochranu pred takýmito útokmi. Z dôvodu vysokých požiadavkov na dátovú priepustnosť a výkonnosť obecne, sa tak v týchto zariadenia často krát používa hardvérová akcelerácia.

Hardvérová akcelerácia poskytuje vysokú rýchlosť spracovania a preto je typicky používaná pre rýchle výpočty konkrétnej úlohy tam, kde nedostačujú obecné procesory. Vývoj aplikácií pre tieto zariadenia je ale náročný a zdĺhavý, čo znemožňuje rýchlo reagovať na nové typy sieťových útokov. Preto je dnešným cieľom tento proces čo najviac uľahčiť a urýchliť. Jednou z možností ako toho dosiahnuť je použitie vysokoúrovňových deklaratívnych jazykov, ako je napríklad jazyk P4, určený pre popis spracovania sieťových prenosov. Jazyk P4 je protokolovo nezávislý a umožňuje nám tak vytvárať sieťové zariadenia, ktoré sú schopné spracovávať aj neštandardné, užívateľom definované protokoly. Medzi veľkú výhodu jazyka P4 patrí jednoduchosť popisu, flexibilita celého zariadenia a nezávislosť na

cieľovej platforme. Skombinovaním hardvérovej akcelerácie a vysoko úrovňového jazyka P4, je tak možné vytvoriť vysoko výkonné zariadenie, ktoré bude možné rýchlo a efektívne prispôsobovať novým typom sieťových útokov.

Táto práca nadväzuje na bakalársku prácu [18] a jej cieľom je vytvoriť akcelerované zariadenie využívajúce jazyk P4, ktoré nahradí súčasne používané zariadenie. Toto zaradenie už nevyhovuje súčasným potrebám a nárokom, ktoré sú naň kladené. Pre možnosť nasadenia zariadenia na neustále sa rozvíjajúce počítačove siete je potrebné, aby nové zariadenie poskytovalo vysokú flexibilitu pre možnosť jednoduchého rozširovania filtračných jednotiek o nové položky, pridávania nových filtračných jednotiek a podobne. K dosiahnutiu tohoto cieľa je práve využívaný jazyk P4 a je kladený dôraz na čo najväčšiu možnú automatizáciu v procese pridávania nových funkcionalít do zariadenia.

Práca je logicky rozdelená do niekoľkých kapitol, ktoré postupne opisujú jednotlivé časti riešenia daného problému. Kapitola 2 sa zaoberá bezpečnosťou počítačových sietí zo zameraním na útoky typu odoprenie služby. Kapitola 3 sa venuje rekonfigurovatelným zariadeniam a ich možným využitím pri spracovaní dát v počítačových sieťach. Kapitola sa taktiež zaoberá možnosťou využitia jazyka P4 v týchto zariadeniach. Kapitola 4 podrobne rozoberá vlastnosti jazyka P4 a venuje sa možnostiam, ako tento jazyk prekladať na technológiu programovateľných hradiel FPGA. Kapitola 5 sa zaoberá návrhom architektúry zariadenia s využitím jazyka P4, ktoré bude slúžiť na ochranu voči (D)DoS útokom a ktoré bude fungovať na platforme hardvérových akcelerátorov NFB. Kapitola 6 sa venuje postupu akým boli jednotlivé časti navrhnutého zariadenia realizované a implementované. Kapitola 7 popisuje výsledky dosiahnuté pri realizácii navrhnutého zariadenia, v jednotlivý častiach vývoja, až po dosiahnuté výsledky získané laboratórnym testovaním na reálnom hardvéry. Posledná kapitola 8 zhrnuje celkovo dosiahnuté výsledky práce.

# Kapitola 2

# Bezpečnosť počítačových sietí

Nástup technológie počítačových sietí nám priniesol efektívny spôsob ako medzi sebou komunikovať a zdieľať informácie. Spolu s touto technológiu sa však začalo rozvíjať aj odvetvie nazývané počítačová kriminalita. Jej cieľom je zneužiť informácie, ktoré sú zdieľané prostredníctvom počítačových sietí, alebo zámerne poškodiť niektorého z účastníkov komunikácie. Počítačová bezpečnosť sa tak stala neoddeliteľnou súčasťou sietí a zaoberá sa ochranou počítačových sietí pred útokmi kriminálnikov. Jej cieľom je ochrana zdieľaných informácií pred zneužitím, ochrana komunikujúcich účastníkov a ochrana samotných prvkov sieťovej infraštruktúry (prepínače, smerovače, atď.).

Podla správy [3] spoločnosti Symantec, ktorá popisuje bezpečnostné hrozby na Internete za uplynulí rok, došlo k zvýšeniu počtu škodlivých programov o viac ako 80%. V minulom roku bola veľká pozornosť verejnosti venovaná virtuálnym menám, kryptomenám. Na získavanie týchto mien tak začalo vznikať obrovské množstvo škodlivých programov, nazývaných minery, ktoré využívajú dostupné prostriedky napadnutého zariadenia k získavaniu rôznych kryptomien. Detekcia týchto škodlivých programov dosiahla nárast až o 8500% oproti minulému roku. Ďalší veľký nárast počtu útokov, až o 600%, bol zaznamenaný na chytré zariadenia domácností ako sú rôzne snímače tepla, vlhkosti, požiaru, bezpečnostné kamery a mnohé ďalšie, ktoré patria do oblasti internetu vecí, anglicky Internet of Things (IoT). Takto napadnuté zariadenia IoT sú následne používané na vytváranie masívnych distribuovaných útokov na odoprenie služby (Distributed Denial of Service, DDoS). Ako príklad môžeme uviesť doteraz najväčší DDoS útok zo dňa 5 marca 2018, ktorý dosahoval veľkosti až 1,7 Tbps [4]. Na generovaní tohoto útoku sa prevažne podieľali napadnuté zariadenia IoT.

Spôsoby obrany a detekcie proti takýmto útokom sú čoraz lepšie a dômyselnejšie, avšak aj metódy útočníkov sa neustále vyvíjajú a vytvárajú tak čoraz zložitejšie, prepracovanejšie útoky a nástroje na ich generovanie. Postupne tak vniklo veľké množstvo rôznych škodlivých programov a útokov, ktoré môžeme rozdeliť do niekoľkých nasledujúcich kategórií [16]:

**Zisťovanie otvorených portov,** známe taktiež ako „port scan“, je spôsob akým sa zisťujú otvorené porty, aké služby na nich pracujú a ako sú zabezpečené. Nejedná sa priamo o útok ale skôr o metódu na získanie informácií o danom systéme a často krát samotnému útoku predchádza. Zaznamenávanie tejto činnosti sa deje pomocou monitorovania sieťových tokov a sledovania odchýliek od normálneho sieťového prenosu. Jednou z týchto odchýliek môže byť zvýšený počet otváraní a zavieraní sieťových spojený na uzatvorených portoch.

**Malware** sú škodlivé programy ktoré zneužívajú známe bezpečnostné chyby a dôveryhodnosť užívateľov. Sú používané k získaniu osobných informácií o užívateľoch, k získaniu neoprávneného prístupu k danému zariadeniu alebo môžu toto zariadenie využiť k svoje replikácii a šíreniu na ďalšie zariadenia. Najčastejším spôsobom šírenia je pomocou emailovej komunikácie, napadnutím počítača prostredníctvom bezpečnostnej chyby alebo priamym, nevedomým nainštalovaním samotným užívateľom. Detekcia najčastejšie prebieha pomocou antivírových programov a systémami, ktoré vyhľadávajú malware pomocou signatúr popísaných reťazcami alebo regulárnymi výrazmi.

**Penetračné útoky** sú určené k prevzatiu kontroly nad systémom a k získaniu určitej úrovne oprávnenia, najčastejšie tých administrátorských. To umožní útočníkovi získať citlivé dáta zo systému alebo môže do systému nainštalovať malware. Najčastejšie je tento typ útoku používaný na informačné systémy, kde sa zneužívajú chyby, danej aplikácie a operačných systémov. Najlepšie protiopatrenie voči týmto útokom je dôkladné a pravidelné testovanie systému pomocou takzvaných penetračných testov.

**Miner** je špeciálnym typom malware, ktorý v minulom roku zaznamenal obrovské množstvo napadnutých počítačov a vyčlenil sa tak do samostatnej kategórie. Jedná sa o škodlivý program, ktorého cieľom je využiť prostriedky napadnutého počítača k získavaniu (ťaženiu) virtuálnych mien na účet útočníka. Miner je možné typicky detektovať antivírusovými programami alebo pozorovaním neobvyklého využívania zdrojov počítača. K detekcií sa využíva aj analýza sieťových prenosov, kde sa sleduje či počítače náhodou nekomunikujú s verejne známymi servermi, slúžiacich na distribúciu výpočtu a zhromažďovanie výsledkov z týchto škodlivých programov (minerov).

**(D)DoS útoky** slúžia k preťaženiu zariadenia, prípadne celej počítačovej siete, obrovským množstvom dát tak, aby zariadenie a služby ktoré na nom pracujú prestali fungovať. Týmto útokom sa podrobne venuje nasledujúca kapitola.

## 2.1 Útoky typu odoprenie služby

Účelom útokov typu odoprenie služby, anglicky Denial of Service (DoS), je vyvolať nedostupnosť sieťovej služby, prípadne zapríčiniť úplný výpadok celej služby. Tieto útoky sa nezameriavajú na trvalé poškodzovanie, získavanie a kompromitovanie dát nachádzajúcich sa na sieťových službách, ale ich účelom je zamedzenie prístupu k týmto dátam ostatným užívateľom. DoS útoky môžeme rozdeliť, podla princípu akým útočia na sieťové služby, do niekoľkých kategórií [12]:

1. **Úroveň sieťového zariadenia** – Sem patria útoky DoS, ktoré zneužívajú softvérové nedostatky a chyby sieťových zariadení, alebo sa snažia vyčerpať hardvérové prostriedky sieťového zariadenia.

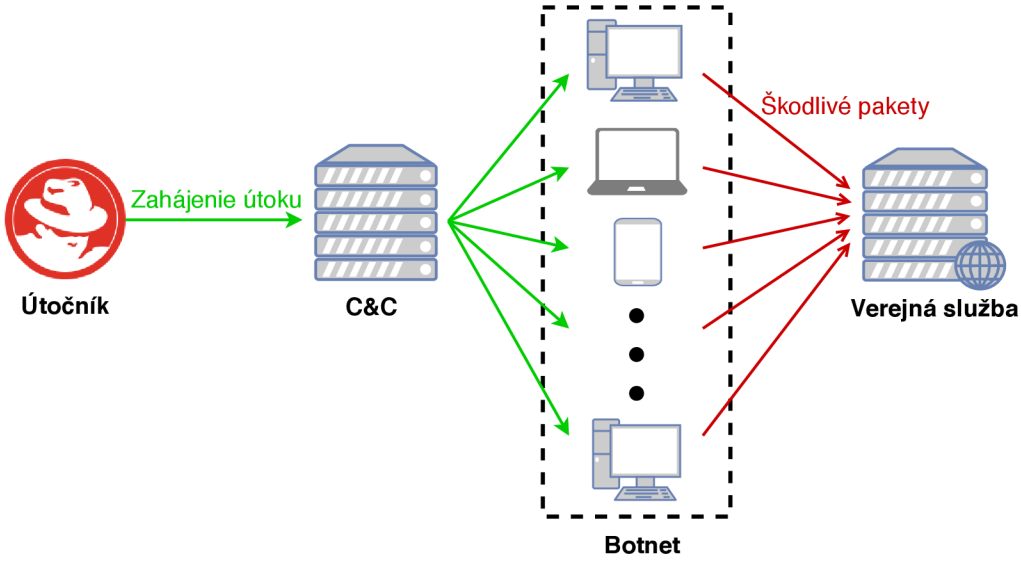
2. **Úroveň operačného systému (OS)** – Pri tomto DoS útoku sa využívajú princípy toho, ako jednotlivé OS implementujú komunikačné protokoly (TCP, UDP, ...) a aké systémové prostriedky tieto implementácie vyčerpávajú.

3. **Aplikačná úroveň** – Pri tomto type sa zneužívajú chyby sieťových aplikácií, pomocou ktorých je možné zapríčiniť pád samotnej aplikácie, alebo je prostredníctvom nej možné vyčerpať zdroje celého systému.

4. **Záplava dátami** – Útočník sa snaží využiť celú šírku pásma siete ktorou je sieťová služba pripojená tak, že posiela obrovské množstvo požiadavkov, ktoré nieje schopné táto služba spracovať.

5. **Úroveň štandartných protokolov** – Snahou je využiť vlastností a funkcii štandartných protokolov. Napríklad niektoré DoS útoky využívajú toho, že protokol IP umožňuje podvrhnutie zdrojovej adresy a tým je možné sieťový tok presmerovať na iné miesto v sieti.

DoS útoky môžeme ďalej odlišovať podla počtu zariadený, ktoré sa podieľajú na vytváraní a generovaní daného útoku. Ďalšou významnou kategóriou útokov DoS sú tak distribuované útoky typu odoprenie služby, anglicky Distributed Denial of Service (DDoS) [20]. Hlavným znakom útokov typu DDoS je že útočník na generovanie útoku využíva väčšie množstvo počítačových staníc, ako je tomu pri útokoch typu DoS, kde útočník typicky používa iba jedno zariadenie (počítačovú stanicu, mobilné zariadenie). Priebeh útoku DDoS je znázornený na obrázku 2.1. Týmto spôsobom je možné vygenerovať útok s niekoľko




Obr. 2.1: Princíp vytvárania útoku typu DDoS.

násobne vyššou intenzitou a väčším, negatívnym dopadom na cieľový systém (pravá časť obrázku). Zariadenia, ktoré sa podieľajú na takomto útoku, sa prezývajú ako zombie, anglicky Bot (prostredná, pravá časť obrázku). Jedná sa o zariadenia nad ktorými má útočník kontrolu pomocou oprávneného ale aj neoprávneného prístupu. Pomenovaním Bot označujeme aj počítačových robotov (malware), ktorých účelom je vykonávať autonómne nejakú činnosť. Infikované zariadenia, boti, následne kontaktujú hlavný server, anglicky Command and Control (C&C), pre získanie ďalších inštrukcií (prostredná, ľavá časť obrázku). Tento server ovláda útočník (ľavá časť obrázku), prostredníctvom ktorého dáva príkazy jednotlivým botom a môže tak vytvárať rozsiahle útoky typu DDoS. Skupina takto infikovaných zariadení, ktoré sú navzájom prepojené a pod kontrolou jedného alebo skupiny útočníkov, nazývame Botnet (prostredná, pravá časť obrázku).

Cieľom DoS a DDoS (skrátene (D)DoS) útokov nebývajú iba sieťové aplikácie a sieťové služby. Často krát sú cieľom dostupné zdroje zariadenia (pamäť, šírka pásma) a taktiež aj samotná sieťová infraštruktúra a jej prvky, ktorými sú tieto zariadenia pripájané. Preto

rozdeľujeme tieto útoky do dvoch hlavných kategórií a to na aplikačné útoky typu (D)DoS, a volumetrické útoky typu (D)DoS [21]. Týmto útokom sa venujú nasledujúce dve podkapitoly.

### 2.1.1 Aplikačné útoky typu (D)DoS

Aplikačné útoky (D)DoS sa zameriavajú na zraniteľnosti sieťovej služby a aplikácie. Typicky útočia na niektorú zo známych zraniteľností cieľového systému, ktorej zneužitie vedie k nedostupnosti pre ostatných užívateľov alebo k úplnému pádu systému [21]. Útoku prebieha tak, že útočník na základe odhalenej zraniteľnosti pošle na cieľovú službu špeciálne upravené správy, ktoré vyvolajú na strane cieľovej služby nekorektné chovanie. Pre tieto útoky je charakteristické využívanie prostriedkov a zneužívanie chýb na úrovni operačného systému a aplikačnej úrovni popísaných v predošlom texte 2.1. Z hľadiska objemu dát niesu tieto útoky významné. Obrana voči týmto útokom je typicky možná na koncových zariadeniach, kde sa zneužívané zraniteľnosti a mechanizmy budú opravovať. Medzi známe aplikačné útoky (D)DoS patria tieto [1]:

**Ping of Death** – Pri tomto útoku sa posiela na cieľový systém množstvo chybných paketov. Maximálna veľkosť IP paketu (vrátane hlavičky) je 65,353 bajtov. Na linkovej, spojovej vrstve sú rámce typicky obmedzené až na 1500 bajtov. V tomto prípade je veľký IP paket rozdelený do viacerých menších paketov IP, nazývaných ako fragmenty. Príjemca tieto fragmenty opätovne skladá do pôvodného paketu. Pri tomto útoku dochádza k manipulácii fragmentu paketu tak, aby sa opätovne zložený paket na strane príjemcu, javil väčší než 65,353 bajtov. To môže spôsobiť pretečenie pridelenej vyrovnávacej pamäte pre paket a spôsobiť odmietnutie legitimných paketov alebo pád celého systému. V dnešných moderných systémoch a aplikáciách, sa tieto chyby už prakticky nevyskytujú a útoky tohoto typu sú na ústupe.

**HTTP Flood** – Útok využíva na oko legitímne HTTP dotazy typu GET a POST. Útočník vyberá také dotazy, ktoré spôsobujú že je daný systém nútení pre ich spracovanie a vytvorenie odpovedí, využiť veľké množstvo svojich zdrojov. Pri veľkom počte takýchto dotazov dochádza k výraznému spomaleniu celého systému.

**(D)DOS SQL Injection** – Tento útok využíva zraniteľnosti, ktoré programátori typicky neošetrujú v informačných systémoch pri práci s databázovými systémami. Útočníkovi tak umožňujú vytvárať vlastné, upravené SQL (Structured Query Language) dotazy. Útok prebieha tak, že sa na systém posielajú dotazy na ktoré musia databázové systémy vynaložiť značné množstvo prostriedkov. Pri veľkom počte takýchto dotazov rýchlo dôjde k nedostupnosti databázového systému

**Zero-day útoky** – Do tejto kategórie patria všetky útoky, ktoré využívajú zatiaľ neznáme zraniteľnosti a chyby systémov, alebo zraniteľnosti na ktoré ešte nebola vydaná žiadna oprava.

### 2.1.2 Volumetrické útoky typu (D)DoS

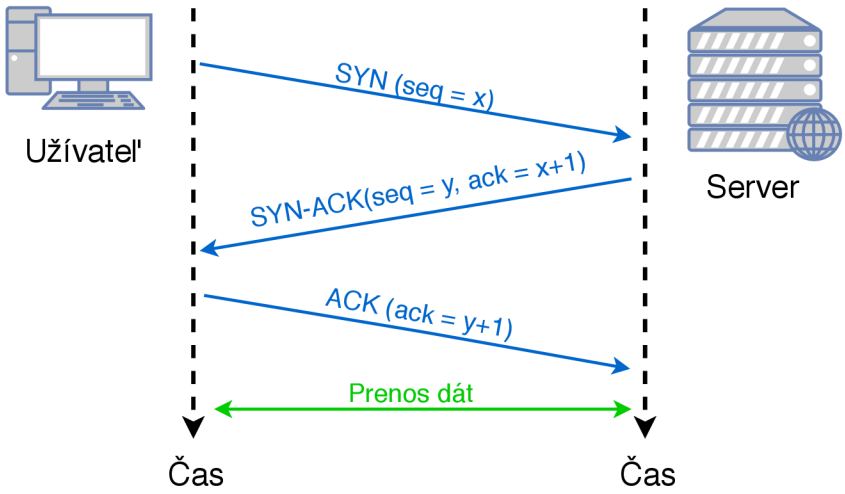
Účelom volumetrických útokov je vyčerpanie zdrojov sieťových zariadení, prípadne vyčerpanie celkových zdrojov sieťovej infraštruktúry, použitím hrubej sily. Cieľom týchto útokov typicky býva preťaženie sieťovej linky (šírky pásma), prostredníctvom ktorej sú zariadenia

pripojené, prípadne vyčerpanie pamäte alebo iných prostriedkov daného systému a zariadení zapojených v sieťovej infraštruktúre [21]. Tento typ útoku prebieha tak, že útočník posiela na cieľovú službu obrovské množstvo vygenerovaných správ a tým dôjde k už spomínaným problémom. Obrana proti týmto útokom na koncových systémoch, častokrát nieje možná, pretože už samotné prostriedky a zdroje, ktorými sú tieto systému pripojené, sú preťažené. Obranu je tak nutné riešiť na úrovni samotnej sieťovej infraštruktúry, ktorá disponuje dostatočnými kapacitami a je tak schopná škodlivé dáta zastaviť skôr než sa dostanú do cieľového systému. Medzi veľmi rozšírené volumetrické útoky patrí:

**UDP Flood** – Ako už z názvu vyplýva, jedná sa o zaplavovanie cieľového systému paketmi UDP (User Datagram protocol). Cieľom je útočiť na náhodné porty systému. To má za následok že systém opakovane kontroluje odposluchávanie aplikácie na danom porte. Ak sa nenájde žiadna aplikácia, tak systém odpovedá paketom ICMP o nedostupnosti „Destination Unreachable“. Tento proces vyčerpáva zdroje systému čo môže viesť k jeho nedostupnosti [14].

**ICMP (Ping) Flood** – Jedná sa o podobný princíp ako u UDP Flood útoku avšak tentokrát sa na cieľový systém posielajú ICMP pakety „Echo Request“ (ping), ktoré sa posielajú čo najrýchlejšie bez toho, aby sa čakalo na ich odpovede. Tento typ útoku môže vyčerpať šírku pásma pre príjem aj odosielanie na cieľovom systéme, pretože systém typicky odpovedá ICMP paketmi „Echo Reply“.

**SYN Flood** – Útok využíva známu slabosť pri vytváraní spojenia TCP (Transmission Control Protocol) pomocou takzvaného „three-way handshake“. Tvorba tohoto spojenia je znázornená na obrázku 2.2. Scenár útoku je že sa posiela veľké množstvo paketov

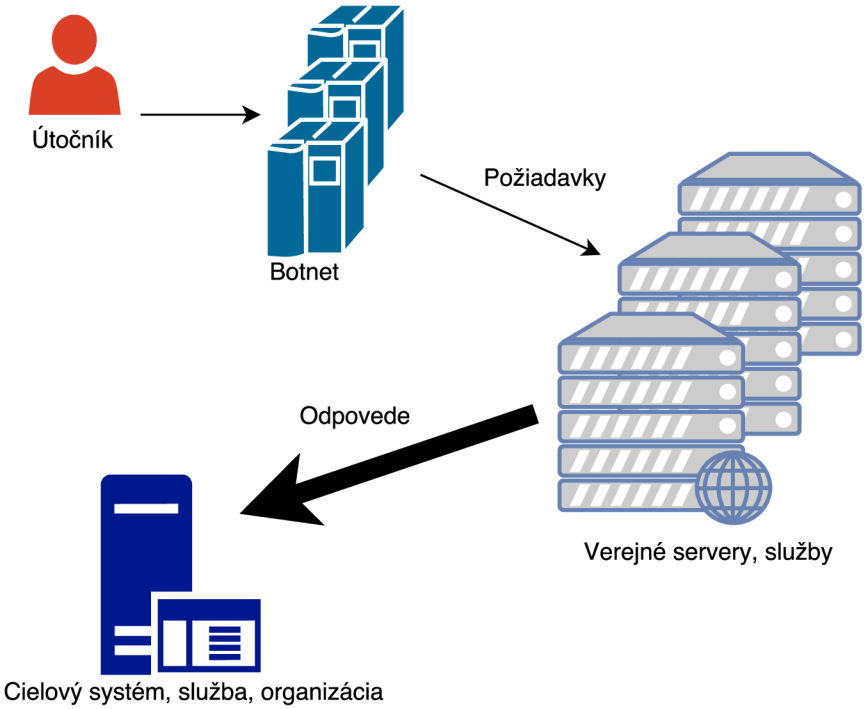



Obr. 2.2: Tvorba spojenia TCP pomocou „three-way handshake“.

SYN na všetky porty cieľového zariadenia. Zariadenie na všetky tieto žiadosti odpovedá paketmi SYN-ACK a na určitú dobu, pre každú žiadosť, alokuje potrebné zdroje. Server následne čaká na potvrdenie paketom ACK. Útočník už ale tieto pakety neposiela a tým je server nútení držať alokované zdroje po stanovenú dobu v časovači. Tým sa server dostane do stavu, kedy vyčerpá všetky svoje dostupné zdroje a už nie je schopný vytvárať nové spojenia ani pre legitímnych užívateľov. Útočník taktiež môže pri generovaní paketov SYN podvrhnúť falošnú IP adresu. To má za následok

že mu nebude doručená žiadna odpoveď od serveru, ale odpovede budú posielané na sfalšovanú IP adresu [13].

**Amplifikačné útoky** – Ďalším významným zástupcom sú amplifikačné útoky typy (D)DoS, kde sa využíva takzvaný princíp odrazu za pomoci iných verejne dostupných služieb. Priebeh takéhoto útoku (D)DoS je znázornený na obrázku 2.3. Princíp útoku založe-




Obr. 2.3: Znázornenie priebehu amplifikačného útoku (D)DoS.

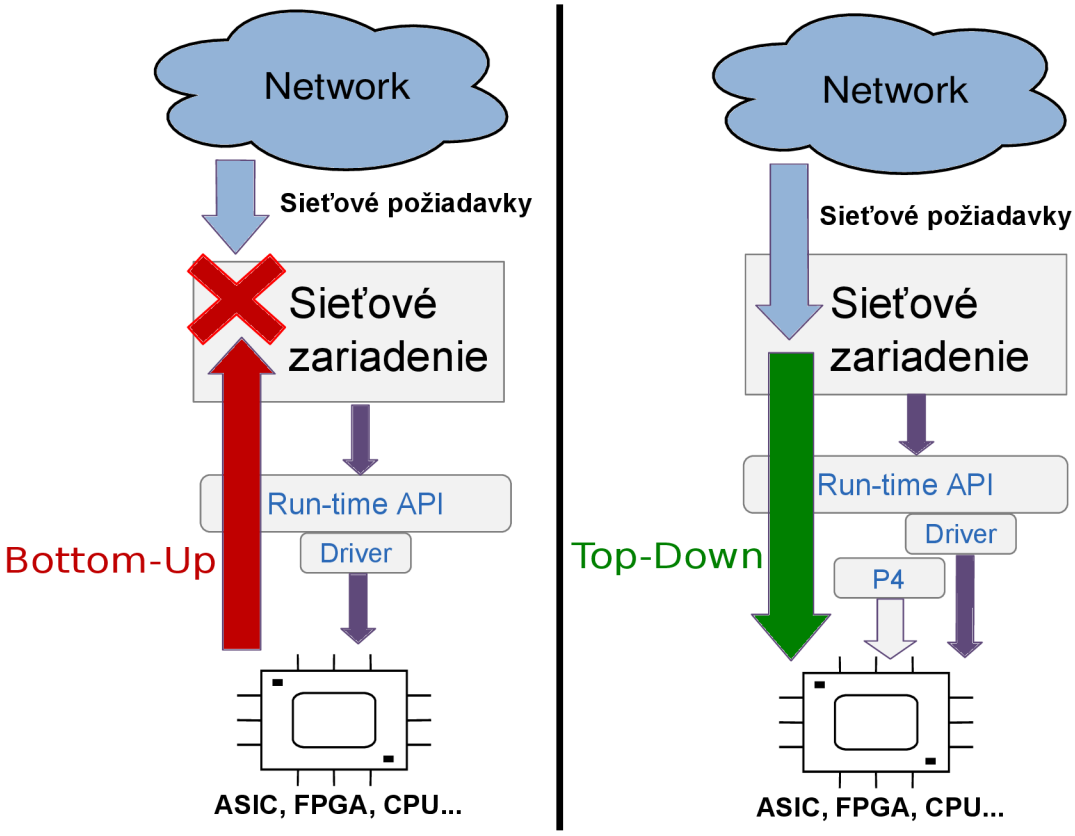
ného na odraze spočíva v tom, že sa posielajú dotazy na verejne dostupné služby, akou je napríklad systém doménových mien (DNS) (pravá časť obrázku) [7]. Tieto služby typicky odpovedajú výrazne väčšími odpoveďami, ktoré sú následne presmerované na cieľový systém (spodná časť obrázku), ktorý je tak zahltený obrovským množstvom dát. Na presmerovanie týchto odpovedí sa využíva princíp sfalšovania zdrojovej adresy IP za adresu cieľového systému. Týmto spôsobom je tak možné dosiahnuť výrazné zosilenie útoku (D)DoS. Pre vytvorenie amplifikačného útoku sa najčastejšie zneužívajú systémy ako DNS (Domain Name Service), NTP (Network Time Protocol) a SNMP (Simple Network Management Protocol). Priemerné zosilnenie DNS amplifikačného útoku, v pomere veľkosti dotazu a odpovede DNS serveru, sa typicky pohybuje v pomere 70:1 čo predstavuje sedemdesiat krát väčšie zosilnenie útoku [2]. Zosilnenie NTP amplifikačného útoku sa typicky nachádza v pomere, medzi veľkosťou dotazu a odpovede od serveru NTP, 20:1 až 200:1. To reprezentuje možné zosilnenie útoku až dvesto krát [5]. SNMP útoky môžu dosiahnuť 600 až 1700 násobné zosilnenie [6]. Tento spôsob útoku je možné realizovať iba na protokol SNMP vo verzii jedna a dva. Od tretej verzie SNMP už tento princíp útoku nie je možný, pretože sa vyžaduje overenie prostredníctvom mena a hesla a taktiež sa používa šifrované spojenie.

# Kapitola 3

# Rekonfigurovatelné sieťové zariadenia

Počítačové siete sa neustále vyvíjajú a vznikajú stále nové sieťové protokoly. Týmto zmenám sa tak musia neustále prispôsobovať aj samotné sieťové zariadenia. Pri vývoji súčasných sieťových zariadení je tak potrebné voliť prístup, ktorý nám umožní rekonfiguráciu zariadenia podla aktuálnych požiadavkov a stavu danej siete. Tento prístup nám tak výrazne predĺži životnosť sieťových zariadení v neustále sa rozvíjajúcich sietových infraštruktúrach.

Zaužívaným spôsobom vývoja sieťových zariadení je vývojový cyklus Bottom-Up (ľavá časť obrázku 3.1). Prvým krokom tohoto prístupu je dekompozícia zadanej úlohy, aplikáciu

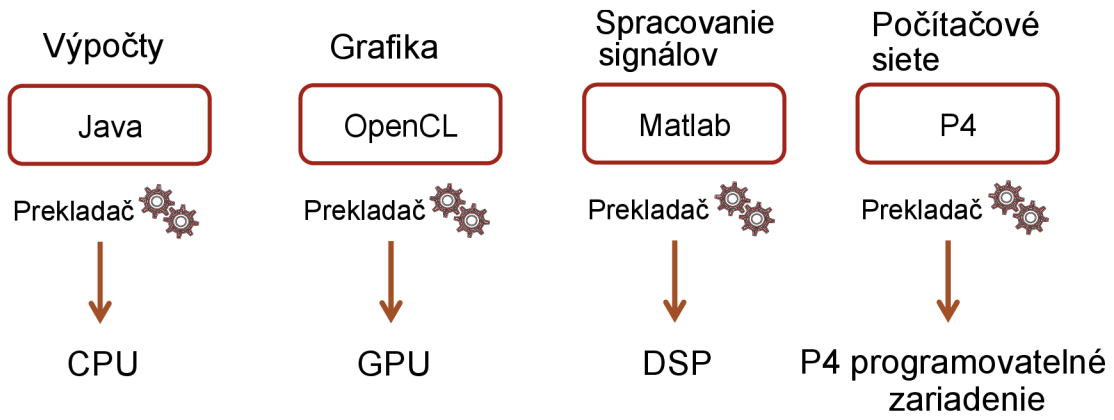



Obr. 3.1: Vývojový cyklus Bottom-Up a Top-Down [24]

cie (napr. filtrovanie paketov) na menšie a jednoduchšie komponenty, ktoré sa budú dať realizovať na požadovanej platforme. Vývoj následne prebieha tak, že sa najskôr implementujú komponenty (firmware) pre danú platformu (ľavá, spodná časť obrázku) a následne sa vytvori komunikačné a konfiguračné rozhranie, anglicky Application Programming Interface (API), pomocou ktorého budeme môcť takto vytvorený firmware konfigurovať (ľavá, prostredná časť obrázku).

Zásadný problém tohoto prístupu je že akákoľvek úprava a rozšírenie systému je veľmi zdĺhavá, pretože si musíme neustále prechádzať celým procesom dekompozície problému. Často krát je potreba vyvinúť úplne nové sieťové zariadenie. Týmto spôsobom sa neustále dostávame do situácie, kedy sieťové zariadenie nie je schopné reagovať, v rozumnom čase, na nové sieťové požiadavky (ľavá, horná časť obrázku). Pre vývoj sieťových zariadený je tak potrebné zvoliť metódu vývoja, ktorá nebude vyžadovať neustále (manuálne) zásahy do najnižšej vrstvy zariadenia (firmwaru), ale umožní nám tieto zmeny a rekonfiguráciu robiť automatizovane, na základe aktuálnych sieťových požiadavkov. Pre tento spôsob vývoja sa využíva princíp z hora na dol, anglicky Top-Down (pravá časť obrázku).

Vývojový cyklus Top-Down prebieha tak, že namiesto dekompozície zadanej aplikácie (napr. spracovanie sieťových paketov), si túto aplikáciu popíšeme vo vhodnom, vysoko úrovňovom, programovacom jazyku. Tento popis následne prejde automatizovaným prekladom, ktorý ho prevedie na nami požadovanú platformu. Prístup Top-Down sa bežne používa v oblasti programovania obecných výpočtov pre CPU (Central Processing Unit), grafických výpočtov pre GPU (Graphics Processing Unit) alebo spracovania signálov pre DSP (Digital Signal Processor) (znázornené na obrázku 3.2). V súčasnosti sa tento princíp čoraz častej-




Obr. 3.2: Prekladové nástroje [32].

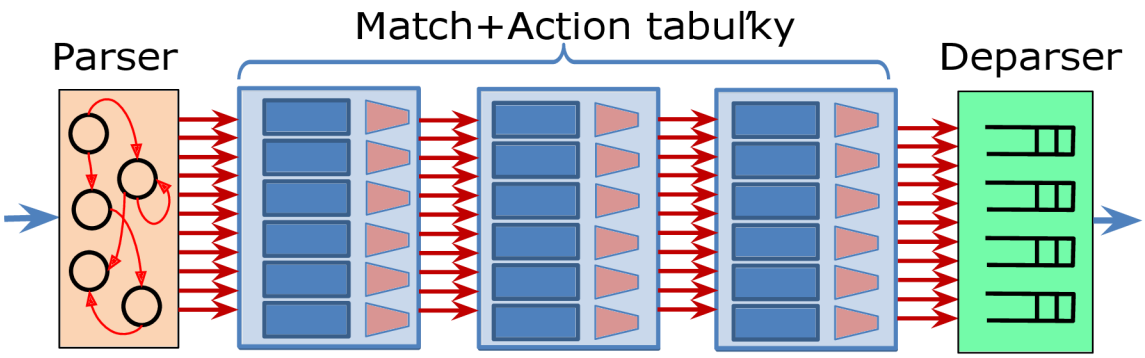
šie objavuje aj v oblasti počítačových sietí, kde popisujeme ako má vyzerať spracovanie sieťových paketov. Jedným z jazykov, ktoré nám umožňujú popisovať chovanie sieťových zariadený, je jazyk P4. Popis v jazyku P4 následne prechádza automatizovaným prekladom na nami požadované a podporované zariadenie (pravá, spodná časť obrázku 3.1).

## 3.1 Jazyk P4

Jazyk P4 (Programming Protocol-independent Packet Processors) patrí medzi deklaratívne, vysoko úrovňového jazyky. Tento jazyk slúži k popisu toho, akým spôsobom majú byť spracovávané pakety v sieťových zariadeniach a je tak vhodným nástrojom pre popisovanie sieťových prvkov ako sú napríklad prepínače, smerovače a NIC (Network Interface Card). Jazyk

je protokolovo nezávislý a umožňuje nám tak vytvárať sieťové zariadenia, ktoré sú schopné spracovávať aj neštandardné, užívateľom definované protokoly [8, 29]. To nám umožňuje rýchlo a flexibilne reagovať na stále novo vznikajúce protokoly a často sa meniace sieťové požiadavky.

Popis sieťového zariadenia sa v jazyku P4 skladá z troch hlavných častí a to parseru, Match+Action tabuliek a deparseru. Tieto časti sú navzájom prepojené v zreťazenej linke tak, ako je to vyobrazené na obrázku 3.3. Úlohou parseru je rozdeliť prijatý paket na sadu



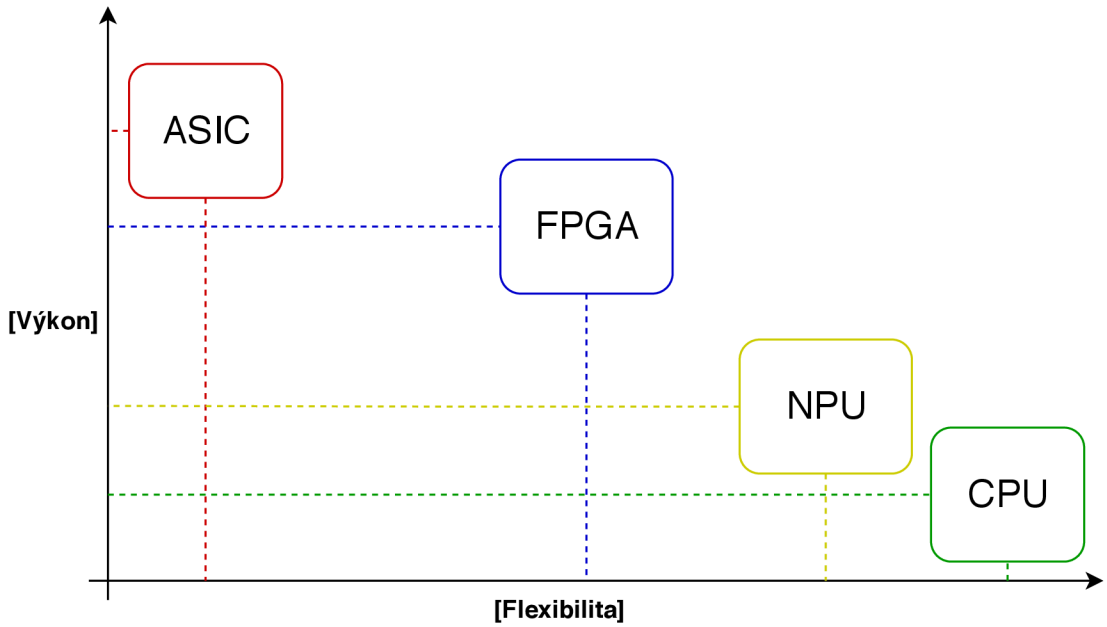

Obr. 3.3: P4 zreťazená linka [32].

protokolových hlavičiek z ktorých je paket tvorený (ľavá časť obrázku). Parser je v jazyku P4 popisovaný prostredníctvom stavov konečného automatu a prechodmi medzi týmito stavmi. Získane hlavičky protokolov sú ďalej spracovávané v postupnosti Match+Action tabuľkách (prostredná časť obrázku). Jednotlivé Match+Action tabuľky vyhľadávajú odpovedajúci záznam podla zvolenej podmnožiny získaných hlavičiek protokolov. Na základe nájdeného záznamu sa vykoná požadovaná akcia, napríklad modifikácia MAC adresy v príslušnej hlavičke protokolu. Upravené hlavičky protokolov sa následne dostanú do deparseru, ktorý z týchto hlavičiek opätovne poskladá sieťový paket a pošle ho na výstupné rozhranie (pravá časť obrázku).

Takto vytvorený popis sieťového zariadenia následne prechádza automatizovaným prekladom pre cieľové zariadenie. Každé zariadenie, ktoré má podporovať jazyk P4, musí byť poskytované spolu s prekladačom, ktorý umožňuje previesť popis v jazyku P4 na konkrétne prvky daného zariadenia. K tvorbe prekladača môžu byť využité verejne dostupné nástroje, knižnice, ktorých účelom je ušetriť prácu a čas pri samotnej implementácii. Takýmito nástrojmi sú napríklad P4-HLIR [27], ktorý umožňuje vytváranie prekladača pomocou jazyka python, alebo p4c, ktorý slúži pre tvorbu prekladača v jazyku C++ [26]. Zariadenie, poskytované s prekladačom jazyka P4, sú označované ako P4 targets. Týmto spôsobom je tak možné prekladať jazyk P4 na rôzne typy zariadený, s rozličnou architektúrou. Príkladom najpoužívanejších architektúr pre P4 zariadenia sú tieto:

**Central Processing Unit (CPU)** – Najčastejšie cielené zariadia sú softvérové nástroje pre CPU. Architektúra CPU poskytuje vysokú flexibilitu a ľahké nasadenie za relatívne nízku cenu a vynaložené úsilie. Nevýhodou týchto systémov je malý výkon a zlá škálovateľnosť pre spracovávanie vysoko rýchlostného sieťového prenosu v reálnom čase (pravá časť obrázku 3.4). Ako príklad prekladača, jazyka P4 na CPU, môžeme uviesť P4C-BEHAVIORAL [25].

**Network Processing Unit (NPU)** – Ďalším častým cieľom sú zariadenia založené na NPU. Jedná sa o integrované obvody so špecifikami funkcionalitami prispôsobenými




Obr. 3.4: Porovnanie technológií FPGA, ASIC a CPU.

a optimalizovanými pre rýchle a efektívne spracovanie sieťových dát. Sme však obmedzený iba na funkcionalitu, ktorú nám takéto zariadenie poskytuje. Prekladač jazyka P4, pre zariadenia NPU, poskytuje napríklad spoločnosť Netronome [22].

**Application Specific Integrated Circuit (ASIC)** – Zariadenia využívajúce na mieru vytvorené obvody ASIC, patria medzi najvýkonnejšie ale ich flexibilita je takmer nulová, pretože sa nedajú preprogramovať (ľavá časť obrázku 3.4). Príkladom prekladača jazyka P4 pre ASIC je Tofino od spoločnosti Barefood Networks [23].

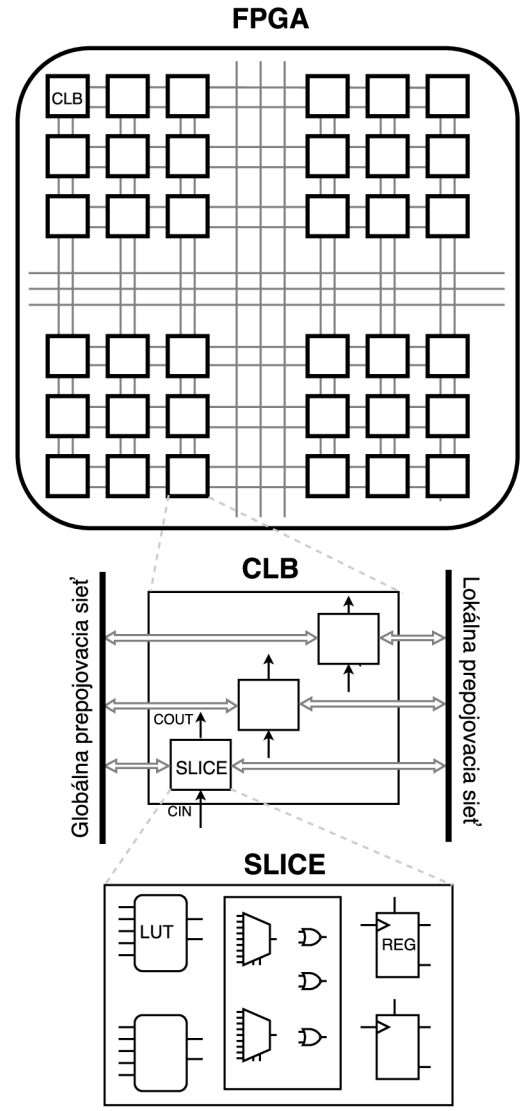
**Field Programmable Gate Array (FPGA)** – Zariadenia s technológiou programovateľných hradiel, sú ďalším typom architektúry, ktorá sa v súčasnosti stáva častým cieľom v oblasti vývoja jazyka P4. Možnosť rekonfigurácie obvodu spojená s vysokým výkonom, ktorý sa približuje k architektúre ASIC, robí z FPGA ideálneho kandidáta na spracovanie vysoko rýchlostných sieťových prenosov v reálnom čase. Ako prekladač jazyka P4 na FPGA môžeme uviesť sadu nástrojov P4FPGA [28]. V nasledujúcej časti tejto práce sa preto budeme zaoberať iba zariadeniami, ktoré využívajú technológiu FPGA.

## 3.2 Technológia FPGA

FPGA sú hardvérové obvody, ktoré môžu byť naprogramované tak, aby realizovali ľubovoľnú aplikáciu reprezentovanú kombinačnou a sekvenčnou logikou. Technológia FPGA tak ponúka relatívne vysokú flexibilitu v kombinácii s vysokou rýchlosťou spracovania. Výhodou technológie FPGA oproti ASIC je v cene a flexibilite kde nie je potrebné vytvárať úplne nový čip. V porovnaní s CPU je FPGA výrazne výkonnejšie avšak poskytuje výrazne menšiu flexibilitu. Spojeným tejto technológie a vysoko úrovňového jazyka P4, dostávame

ideálnu kombináciu prostriedkov pre vývoj zariadenia na spracovanie vysoko rýchlostných sieťových dát v reálnom čase.

Základná štruktúra obvodu FPGA je znázornená na obrázku 3.5. FPGA obvody sú




Obr. 3.5: Štruktúra obvodu FPGA.

tvorené maticou prepojených, konfigurovateľných logických blokov CLB (Configurable Logic Block) (vrchná a prostredná časť obrázku). Bloky CLB sú ďalej zložené z menších blokov nazývaných slice (spodná časť obrázku). Každý blok slice obsahuje funkcionálne generátory, multiplexory, registre, prípadne ďalšie komponenty s možnosťou konfigurácie (spodná časť obrázku). Pri tvorbe obvodu, prepojení (na základe navrhnutej aplikácie) je snaha o čo najkratšie prepojenia medzi logickými jednotkami tak, aby sa dosiahlo čo možno najväčšej možnej frekvencie obvodu. Súčasťou FPGA sú obvody na pripojenie externých zariadení ako sú napríklad rýchle sériové spoje (Ethernet, PCI-Express) alebo externé pamäte.

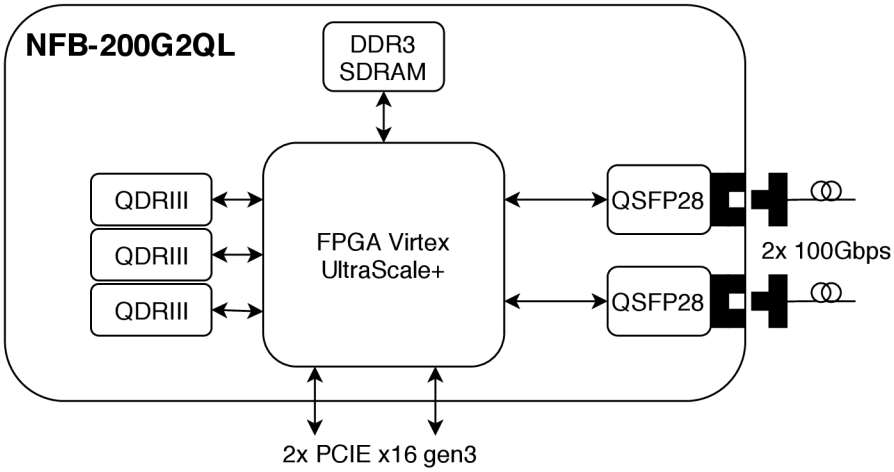
Pri tvorbe aplikácií, pre obvod FPGA, sa využívajú programovacie jazyky určené pre popis digitálnych obvodov ako sú VHDL (Hardware Description Language) alebo objektovo orientovaný SystemVerilog. Takto popísaný obvod následne prechádza syntézou, kde sa

jednotlivé prvky obvodu a ich prepojenie mapujú na primitívna (komponenty) čipu FPGA. Tieto jazyky patria medzi nízko úrovňové a vývoj je tak časovo náročný. Cieľom je preto používať vysoko úrovňové jazyky akým je napríklad jazyk P4, ktorý je výrazne flexibilnejší a pohodlnejší. Obecne ale platí, že čím abstraktnejší jazyk je pre popis obvodu použitý, tým menšiu kontrolu má vývojár nad tým, ako bude výsledný obvod vyzerať (z akých prvkov danej architektúry bude obvod tvorený). Nepopierateľnou výhodou je ale to, že sa obvod v týchto jazykoch vytvára jednoduchšie a ďalšie rozširovanie funkcionality obvodu je výrazne rýchlejšie.

## 3.3 Hardvérové akcelerátory COMBO a NFB

Hardvérové akcelerátory rodiny COMBO a NFB sú špecializované karty, ktoré používajú technológiu FPGA na spracovávanie vysoko rýchlostných počítačových sietí. Tieto akcelerátory sú vyvíjané v rámci vedeckovýskumnej skupiny Liberouter, ktorá patrí pod združenie CESNET [9].

Jedným z týchto akcelerátorov je karta NFB-200G2QL [15]. Blokové schéma karty, je znázornené na obrázku 3.6. Karta umožňuje pripojenie až dvoma portami PCI-Express generácie, pre možnosť prenosu dát do pamäte počítača plnou rýchlosťou 200 Gbps (spodná časť obrázku). Hlavnou častou karty je čip FPGA od výrobcu Xilinx s architektúrou UltraScale+ (prostredná časť obrázku) [34]. Karta disponuje sieťovými, optickými rozhraniami (QSFP28 transciever), kde každé rozhranie je schopné prenášať dáta o rýchlosti 100 Gbps (pravá časť obrázku). Dostupné sú taktiež externé pamäte (QDRIIIe SRAM) priamo pripojené k obvodu FPGA (ľavá časť obrázku).

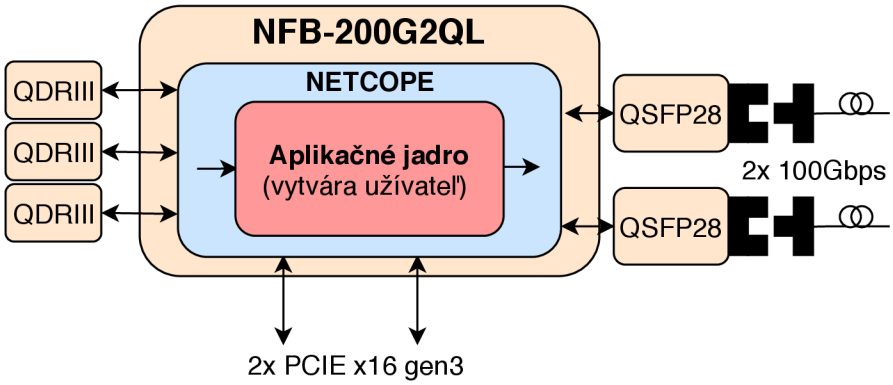



Obr. 3.6: Blokové schéma karty NFB-200G2QL [15].

Pre tieto akceleračné karty sú poskytované už vopred vytvorené komponenty, ktoré môžeme použiť pri vývoji vlastnej aplikácie. Takto poskytované komponenty sa nazývajú ako IP cores. Jedným z nich je modulárne vývojové prostredie NetCOPE. Jedná sa o platformu, ktorá poskytuje infraštruktúru, sadu komponent a softvérových nástrojov, pre podporu rýchleho vývoja sieťových aplikácií na hardvérových akcelerátoroch COMBO. Účelom platformy NetCOPE je uľahčiť prácu vývojárovi samotnej aplikácie pre danú kartu. Dostupné sú tak už vopred pripravené adaptéry na komunikáciu po zbernici PCI, príjem paketov z optických rozhraní (transciever), vyrovnávacie pamäte a mnoho ďalších komponent, ktoré

by si musel každý vývojár implementovať samostatne [10, 31]. Súčasťou tejto platformy sú aj softvérové knižnice, nástroje a ovládače, ktoré umožňujú komunikovať s hardvérovým akcelerátorom. V súčasnosti je platforma NetCOPE podporovaná na všetkých hardvérových akcelerátoroch COMBO vrátane NFB-200G2QL2.

Z pohľadu vývojára firmwaru je hlavnou častou, komponentov platformy NetCope takzvané aplikačné jadro, zobrazené na obrázku 3.7. Do tejto komponenty sú vyvedené všetky




Obr. 3.7: Zapojenie aplikačného jadra v platforme NetCOPE.

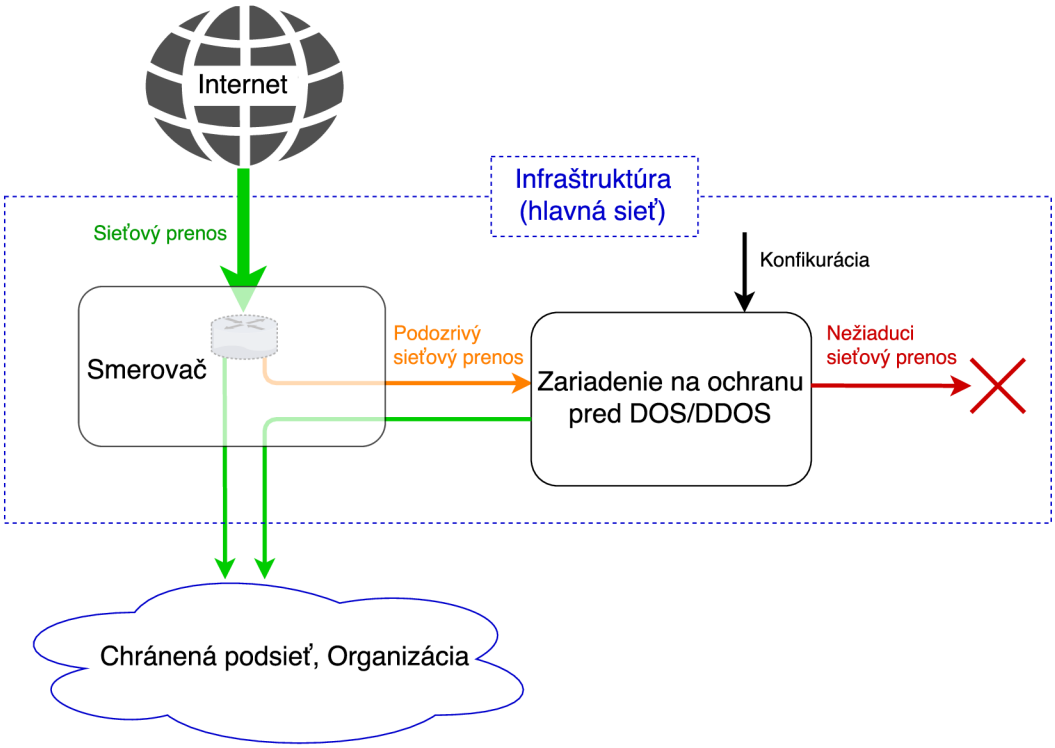
dôležité rozhrania ako sú vstupné a výstupné sieťové rozhrania, rozhranie pre softwarovú konfiguráciu, rozhranie pre externé pamäte a podobne. Úlohou vývojára je doplniť požadovanú funkcionalitu a zapojiť si vlastný fimrware (aplikáciu) do tohto aplikačného jadra, komponenty.

Pre akcelerátory NFV a platformu NetCOPE je dostupný aj prekladač jazyka P4. Podporou jazyka P4 a vývojom prekladača pre tieto akcelerátory sa zaoberá vedecko výskumná skupina Liberouter [19, 8]. Tomuto prekladaču sa podrobnejšie venuje kapitola 4.3.

Pomocou akcelerátoru NFB-200G2QL, platformy NetCOPE a vyvíjaného prekladača jazyka P4, tak máme možnosť vytvárať výkonné, flexibilné a cenovo dostupné zariadenia.

## 3.4 DDoS Protektor

Cieľom volumetrických útokov typu (D)DoS typicky nie je priamo koncové zaradenie (služba), ale ich cieľom býva samotná sieťová infraštruktúra. Obrana voči týmto útokom častokrát nieje na koncových zariadeniach možná. Preto je potrebné zabezpečiť ochranu už na úrovni sieťovej infraštruktúry, ktorá je schopná obrovské množstvo dát (generovaných týmito útokmi) preniesť a spracovať. Bežné počítače založené na platforme obecných procesorov CPU niesu schopné také množstvo dát spracovať v reálnom čase. Preto sa k týmto účelom využívajú výkonnejšie platformy ako sú FPGA a ASIC. Centralizované zariadenie na ochranu voči volumetrickým útokom (D)DoS, ktoré využíva technológiu FPGA, už vzniklo vrámci bakalárskej práce [18] na ktorú táto práca nadväzuje a je jej pokračovaním. Zariadenie bolo taktiež prezentované prostredníctvom článku [17]. Nevýhodou súčasného riešenia je zdĺhavý vývoj a s tým spojená pomalá reakcia na stále nové a prepracovanejšie útoky typu (D)DoS. Spôsob nasadenia a fungovania zariadenia je znázornený na obrázku 3.8. Princíp centralizovaného zariadenia spočíva v tom, že v danej infraštruktúre (modré ohraničenie na obrázku) bude exitovať iba jedno takéto zariadenie (pravá časť obrázku). Následne bude na toto zaradenie presmerovaná sieťová prevádzka (určená pre danú podsieť, organizáciu), prostredníctvom smerovacích zariadení danej infraštruktúry (ľavá, prostredná




Obr. 3.8: Spôsob nasadenia zariadenia na ochranu pred DoS/(D)DoS.

časť obrázku). Tento sieťový tok následne zariadenie očistí od nežiaducich dát (paketov). Takto vyčistená sieťová prevádzka bude poslaná naspäť do sieťovej infraštruktúry a následne do danej podsiete, organizácii (spodná časť obrázku).

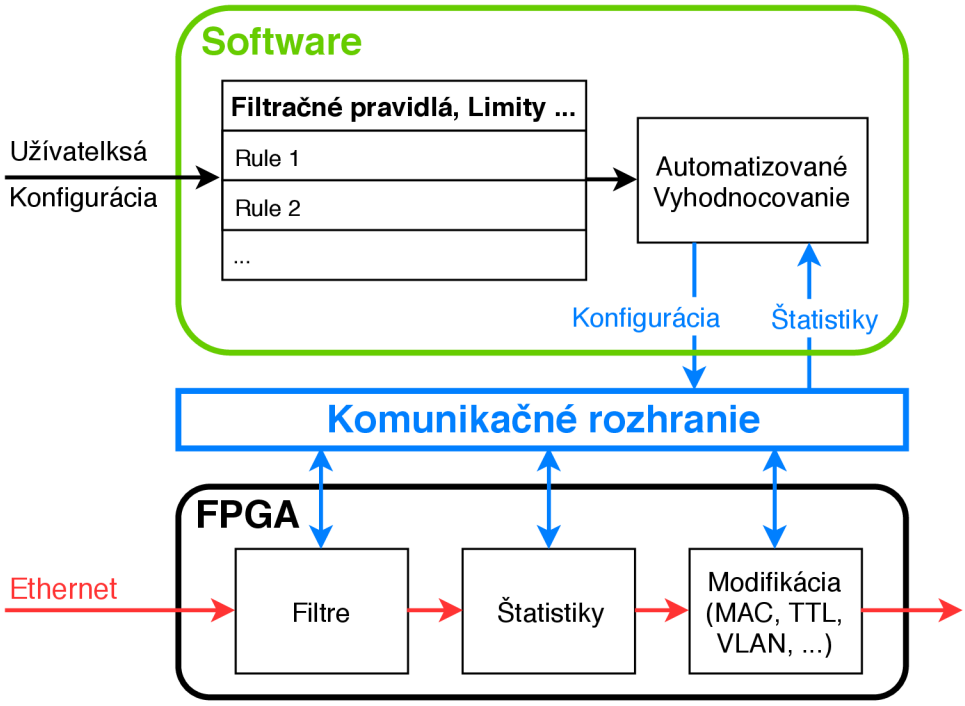
Schéma a fungovanie samotného zariadenia je vyobrazené na obrázku 3.9. Zariadenie je tvorené troma časťami a to:

**Softvérom** na detekciu útokov, nežiaducich dát a konfiguráciu ostatných častí podla aktuálneho stavu sieťovej prevádzky (horná, zelená časť obrázku).

**Komunikačným rozhraním,** ktoré umožňuje vzájomnú výmenu dát medzi softvérovou a hardvérovou častou zariadenia (modrá prostredná časť obrázku).

**Hardvérovým akcelerátorom FPGA** na spracovanie sieťového prenosu podla zvolenej konfigurácie zo softvérovej časti zariadenia (čierna, spodná časť obrázku).

Firmware a všetky jeho moduly, pre FPGA, boli implementované prostredníctvom nízko úrovňových jazykov, určených pre popis hardvéru, ako sú VHDL a SystemVerilog. Vývoj jednotlivých častý firmware je v týchto jazykoch značne zložitý a časovo náročný. V priebehu nasadenia a testovania celého zariadenia sa ukázalo, že akákoľvek potrebná zmena v spôsobe spracovania paketov a pridávania nových funkcionalít je náročná a zaberá veľa času. Ďalším problémom pri tomto procese je zanášanie množstva skrytých chýb do celého systému, ktoré sa veľakrát odhalia až pri samotnom nasadený zariadenia. Cieľom je tak pri vývoji firmware, použiť abstraktnejšie programovacie jazyky, ktoré by výrazne urýchlili vývoj a zlepšili jeho prenositelnosť medzi rôznymi architektúrami. Jedným z týchto jazykov je práve jazyk P4 a možnosť jeho prekladu do jazyka VHDL.




Obr. 3.9: Zariadenie na ochranu pred DoS/(D)DoS.

### 3.4.1 Nedostatky zariadenia

Aby sme vysvetlili prečo je súčasné zariadenie nevyhovujúce dnešným podmienkam, tak uvažujme nasledujúcu situáciu, ktorá vychádza zo skutočných požiadavkov na zariadenie.

Vo firmwaru zariadenia sa nachádza takzvaný prefixový filter, ktorý vyhľadáva záznamy na základe zdrojovej IP adresy. Od užívateľa, ktorý naše zariadenie používa, sme dostali požiadavok na rozšírenie tohoto filtru o možnosť filtrovania prefixu nezávisle pre každú VLAN, čo vedie na pridanie ďalšej položky podla ktorej budeme filtrovať. Jedná sa konkrétne o položku ID z hlavničky protokolu VLAN. Aby sme tento stav dociellili je potrebné urobiť ručný zásah do firmwaru zariadenia. Tento zásah pozostáva z niekoľkých časovo náročných krokov. Prvým krokom je získanie hodnoty ID z paketu. Následne je potrebné upraviť implementáciu samotného filtru tak aby bol schopný filtrovať aj na základe novej položky ID. Implementácia filtru v jazyku VHDL je značne komplikovaná a jej úprava o možnosť filtrácie podľa novej položky môže trvať značnú dobu. Následne je potrebné overiť funkčnosť tejto úpravy. To znamená upraviť simulačné prostredie pre tento filter a odladiť prípadné chyby ktoré sme do systému zaniesli našou úpravou. Po overení funkcionality je potrebné overiť že filter stále dosahuje požadovanú dátovú priepustnosť, výkonnosť. Tento krok vedie na ďalšie optimalizácie implementácie filtru. Keď už máme upravený filter plne funkčný a dosahuje požadovaný výkon, tak je ešte potrebné rozšíriť softwarové rozhranie, ktorým sa tento filter konfiguruje, tak aby sme boli schopný pridávať záznamy pre možnosť filtrácie pomocou hodnoty ID.

Z uvedeného postupu je evidentné že celý tento proces zaberá veľké množstvo práce a času. Uvedený príklad sa týka iba jedného požiadavku pre konkrétne nasadenie. Požiadavky pri nasadení zariadenia v rôznych sieťach sa však môžu výrazne líšiť. Individuálny prístup ku každému vzniknutému požiadavku je vzhľadom na náročnosť celého procesu úpravy takmer nemožný. Tento prístup ďalej vedie na realizáciu obecného firmwaru, ktorý

vie úplne všetko. To ale taktiež nemusí byť výhodné, máloktoré nasadenie zariadenia by vyžadovalo všetku funkcionalitu, ktorú by takéto zariadenie poskytovalo.

Popis v jazyku P4 nám umožní deaktivovať nepotrebné časti a pripraviť tak zariadenie pre potreby konkrétneho nasadenia. Cieľom tejto práce je tak vytvoriť flexibilné a zároveň výkonné zariadenie s využitím jazyka P4 a celý proces pridávania a odoberania nových funkcionalít čo naviac zautomatizovať.

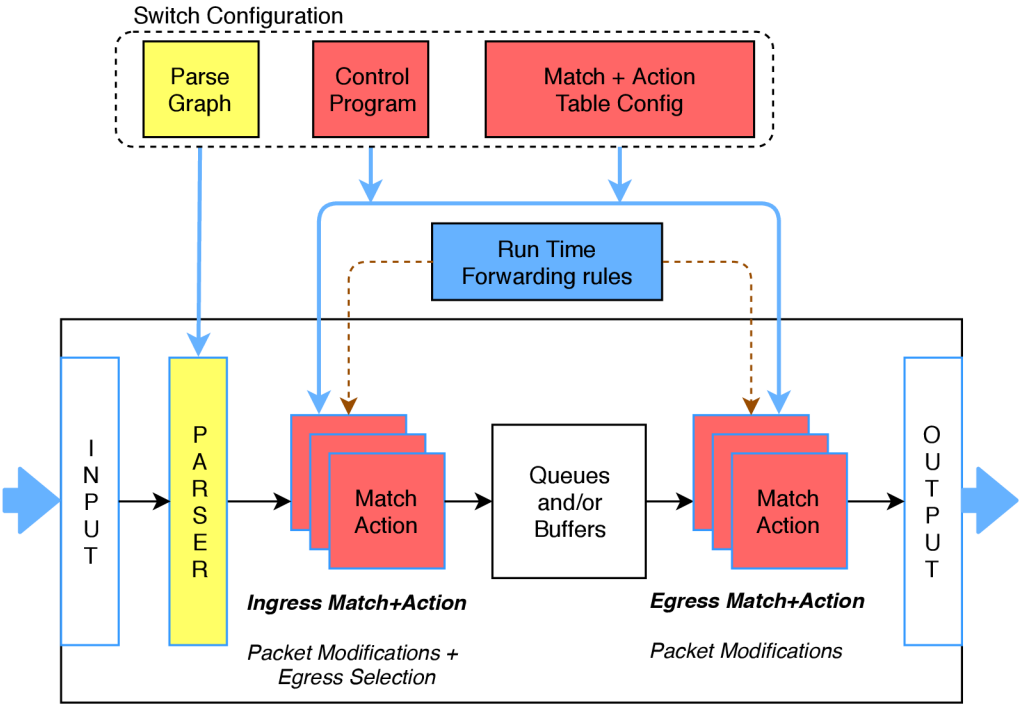
# Kapitola 4

# Špecifikácia jazyka P4

V súčasnosti existujú dva štandardy jazyka P4 a to starší $P4_{14}$ [29] a novší $P4_{16}$ [30]. Nový štandard $P4_{16}$ zachováva princípy staršieho $P4_{14}$, rozširuje ho o nové konštrukcie a prináša zvýšenie flexibility celého jazyka. Prvá časť kapitoly sa zaoberá štandardom jazyka $P4_{14}$, kde sa podrobne venujeme jednotlivým konštrukciám samotného jazyka a spôsobom akým sa tieto konštrukcie definujú a ako pracujú. Druhá časť je venovaná štandardu $P4_{16}$, ktorý prináša do jazyka určité zmeny a nové konštrukcie.

## 4.1 Štandart $P4_{14}$

Abstraktný pohlaď na fungovanie zariadenia popísaného v jazyku $P4_{14}$ je znázornený na obrázku 4.1. Štandard jazyka $P4_{14}$ [29] je založený na modely, ktorý pozostáva z parseru




Obr. 4.1: Abstraktný model P4 zariadenia [29].

(žltá časť obrázku) a postupnosti Match+Action tabuliek (červená časť obrázku) nachádzajúcich sa medzi vstupom a výstupom. Úlohou parseru je vyhľadať a získať hlavičky z každého prichádzajúceho paketu. Tieto hlavičky sú následné spracovávané v postupnosti Match+Action tabuliek. Match+Action tabuľky vyhľadávajú záznam podla zvolenej podmnožiny hlavičiek protokolov daného paketu a vykonávajú zvolenú akciu na základe nájdeného záznamu. Jazyk $P4_{14}$ sa tak zameriava na špecifikáciu parseru (vrátane hlavičiek), Match+Action tabuliek (vrátane akcií) a kontrolu toku dát cez tieto tabuľky. Toto riadi programátor pri vytváraní programu a tým tak špecifikuje konfiguráciu P4 zariadenia ako je uvedená na obrázku 4.1. Tvorbu programu je možné rozdeliť na päť základných krokov a to na:

1. Definícia formátu a štruktúry hlavičiek každého protokolu, ktoré bude zariadenie podporovať.

2. Definícia spôsobu parsovania pre získavanie protokolových hlavičiek z paketu. Parser je v jazyku $P4_{14}$ definovaný prostredníctvom konečného automatu. Priechody medzi stavmi automatu sú riadené na základe zvolených položiek z hlavičiek.

3. Definícia tabuliek, ktoré obsahujú záznamy pre mapovanie sieťových tokov na akcie. Každý záznam v tabuľke vyvolá jednu akciu. Vstupné dáta (položky z hlavičiek) sú prevedené na kľúč, na základe ktorého sa uskutoční vyhľadávanie v tabuľke.

4. Definícia akcií, ktoré sa budú vykonávať nad daným paketom. $P4_{14}$ poskytuje základnú sadu primitívnych akcií pre prácu a modifikáciu spracovávaných dát. Užívateľ si môže následne definovať svoje vlastné akcie, ktoré sú zložené z primitívnych akcií alebo už z vytvorených užívateľských akcií.

5. Definícia riadenia toku dát, kontrolného programu, ktorý jednotlivé časti spojuje dohromady. Špecifikuje sa poradie a kritéria, podľa ktorých budú Match+Action tabuľky aplikované. Dostupné sú rôzne režimy ako je výber podľa spustenej akcie, výber na základe nájdenia alebo nenájdenia záznamu v tabuľke, a podobne.

Takto špecifikované zariadenie je tvorené zreťazenou linkou tak, ako je to znázornené na abstraktnom modely (obrázok 4.1). Výsledné zariadenie je zložené z parseru, Match+Action tabuliek, front a deparseru, ktoré pracujú nasledovne:

- Parser každý paket rozdelí na hlavičky jednotlivých protokolov, ktoré sú následne poskytované Match+Action tabuľkám (predná, žltá časť obrázku).

- Zreťazené Match+Action tabuľky, nasledujúce hneď za parserom (predná, červená časť obrázku), rozhodujú o tom, na ktoré sieťové porty majú byť spracované pakety odoslané, prípadne či majú byť zahodené.

- Nasleduje mechanizmus front (biela, prostredná časť obrázku), ktoré realizujú distribúciu dát podľa konfigurácie zo vstupných Match+Action tabuliek. Fronty umožňujú meniť poradie spracovávaných paketov a tým tak realizovať služby pre rezerváciu a riadenie dátových tokov (Quality of Service, QoS).

- Zreťazené Match+Action tabuľky, nasledujúce za frontamy (zadná, červená časť obrázku), slúžia k modifikácii hodnôt jednotlivých položiek v hlavičkách ešte pred tým, než budú pakety opätovne zložené z hlavičiek.

- Na záver sa v deparseri (pravá časť obrázku), zo spracovaných a modifikovaných hlavičiek, opätovne zloží výsledný paket a je poslaný na príslušný, výstupný sieťový port.

### 4.1.1 Špecifikácia hlavičiek

Definícia hlavičiek je v jazyku $P4_{14}$ veľmi podobná štruktúram v jazyku C. Pri tvorbe novej hlavičky je použitá jednoduchá syntax vo formáte *header_type [name] {..}* kde *[name]* značí názov hlavičky. Následne sa jednotlivé položky hlavičky definujú vo formáte *[name]:[size]* v sekcii *fields*, kde *[size]* značí veľkosť položky v bitoch. Príklad vytvorenia hlavičky ipv5_t môže vyzerať nasledovne:

```
// Deklarácia novej hlavičky
header_type ipv5_t {
    fields {
        src : 8;
        dst : 8;
        frag : 2;
    }
}
```

Výpis 4.1: Deklarácia hlavičky v jazyku $P4_{14}$.

Vo výpise 4.1 je uvedená hlavička kde sú známe veľkosti všetkých jej políčok a výslednú veľkosť hlavičky spočítame ako ich súčet. Existujú však aj protokoly, ktoré majú veľkosť celej hlavičky uvedenú priamo v jej definícii, pretože veľkosť všetkých políčok nie je pevná. Príkladom je protokol IPv6 a jeho rozširujúce hlavičky. V jazyku $P4_{14}$ sa myslelo aj na túto situáciu a preto je možné pomocou kľúčového slova *length* definovať vzorec pre výpočet veľkosti hlavičky. Pomocou kľúčového slova *max_length* môžeme uviesť aj maximálnu veľkosť danej hlavičky v bajtoch. Deklarácia takejto hlavičky vyzerá nasledovne:

```
// Definícia novej hlavičky
header_type ipv7_t {
    fields {
        src : 8;
        dst : 8;
        totalLen : 16;
        frag : 2;
    }
    // Nastavenie veľkosti hlavičky
    length : (totalLen + 1) * 8;
    max_length : 1024;
}
```

Výpis 4.2: Deklarácia hlavičky zo zadanou veľkosťou v jazyku $P4_{14}$.

### 4.1.2 Špecifikácia parsovania

Parsovanie je v $P4_{14}$ realizované prostredníctvom konečného automatu, ktorý pozostáva z definície stavov a prechodov do nasledujúcich stavov. Príklad definície stavu automatu a jeho prechodov vyzerá nasledovne:

```
// Globálna premenná typu eth
header ethernet eth;
// Definícia stavu automatu
parser ethernet {
    // Získanie hlavičky z paketu
    extract(eth);
    // Prechod do nasledujúceho stavu alebo kontrolného programu
    switch(eth.ethertype) {
        case 0xAB00 : ingress_control;
        case 0x8100 : vlan;
```

```
        case 0x9100 : vlan;
        case 0x0800 : ipv4;
        case 0xA100 mask 0xF100 : myProto;
    }
}
```

Výpis 4.3: Definícia stavu automatu v jazyku $P4_{14}$.

Ako prvé sa vytvára stav automatu pomocou syntaktického zápisu *parser [name]{..}*. V tele tejto konštrukcie určujeme premennú, kam sa majú vyparsované dáta uložiť. To dosiahneme kľúčovým slovom *extract* tak ako v našom prípade. Dáta budu uložené do premennej *eth* typu *header*, ktorú už máme vytvorenú. Pre definíciu prechodu do ďalšieho stavu automatu sa použije konštrukcia *switch* a jej parameter bude niektorá z už vyparsovaných položiek, v našom prípade *eth.ethertype*. Parameter môže následne obsahovať viac políčok, ktoré tak tvoria jeden binárny vektor, ktorý sa následne porovnáva s hodnotu v sekciách *case*.

Pri porovnávaní však môžu nastať situácie, kedy nebudeme chcieť porovnávať na presnú zhodu hodnôt. K tomu slúži kľúčové slovo *mask*. Pred porovnaním sa s príslušnou hodnotou a maskou vykoná logický súčin a výsledok sa následne porovná s hodnotou v *case*. V našom príklade bude pre hodnotu 0xA100 použitá najprv maska 0xF100.

Výsledkom sekcie *switch* nemusí byť iba nasledujúci stav automatu. Môže to byť taktiež kontrolný program, v našom príklade to je *ingress_control*. Pri tomto prípade je parsovanie paketu ukončené a spustí sa spracovanie v Match+Action tabuľkách.

Začiatok vykonávania programu P4 začína vždy na nami definovanom, prvom stave automatu. Ten môže byť definovaný nasledovným spôsobom:

```
// Počiatočný stav automatu (paresru)
parser start {
    return parse_eth;
}
```

Výpis 4.4: Definícia prvého stavu parseru v jazyku $P4_{14}$.

### 4.1.3 Špecifikácia Match+Action tabuliek

Match+Action tabuľky sú používane pre mapovanie získaných hlavičiek na najvhodnejšiu akciu. Definícia Každej tabuľky je zložená z dvoch sekcií. Vytvorenie Match+Action tabuľky môže vyzerať nasledovne:

```
// Definícia Match+Action tabuľky
table filter {
    // Položky podla ktorých budeme vyhľadávať
    reads {
        ipv4.srcAddr : lpm;
        tcp.dstPort : range;
    }
    // Zoznam dostupných akcií pre jednotlivé záznamy v tabuľke
    actions {
        PushVlan;
        Permit;
        NoOp;
    }
}
```

Výpis 4.5: Definícia Match+Action tabuľky jazyku $P4_{14}$.

Sekcia uvedená kľúčovým slovom *reads* obsahuje políčka a hodnoty, ktoré majú byť použité pri vyhľadávaní záznamu. Každé toto políčko môže byť v tabuľke vyhľadávané podľa

zvoleného, podporovaného algoritmu. Pre vyhľadávanie v tabuľkách je dostupných niekoľko rozličných režimov a to:

**exact match**
Záznam v tabuľke sa musí presne zhodovať s vyhľadávanou hodnotou.

**ternary**
Nad vyhľadávanou hodnotou sa uskutoční maskovanie a až potom dôjde k samotnému vyhľadaniu záznamu. V tomto prípade môže dôjsť k viacnásobnej zhode a preto musíme riešiť prioritu jednotlivých záznamov.

**lpm (longest prefix match)**
Jedná sa o variantu predošlého prípadu, kedy sa maskuje $N$ koncových bitov a porovnáva sa tak iba zvolený prefix.

**range**
K zhode zo záznamom dôjde v prípade že vyhľadávaná hodnota sa nachádza v špecifikovanom rozsahu hodnôt.

**valid**
Vyhodnocuje korektnosť hľadanej položky. Využívané najme pri kontrole platnosti hlavičiek v paketoch.

V sekcii *action* sa nachádza zoznam podporovaných akcií, ktoré môžu byť následne použité pri vytváraní záznamov v tabuľke.

Vo výpise 4.5 sú použité dve políčka dvoch protokolov. Prvé je zdrojová adresa protokolu IPv4, ktorá sa porovnáva algoritmom LPM. Druhá je cieľový port protokolu TCP, ktorý sa porovnáva na rozsah (*range*). Následne sa v sekcii *action* nachádza zoznam definovaných operácií. Tieto akcie môžu mať vstupné parametre. Tieto parametre sú uložené vo vytvorených záznamoch v Match+Action tabuľkách.

### 4.1.4 Špecifikácia akcii

Jazyk $P4_{14}$ definuje množinu základných, primitívnych akcií. Tieto akcie môžu byť následne použité pri definovaní zložitejších akcií. Programátor si tak môže definovať vlastné, zložitejšie akcie, zložené z primitívnych akcií a už definovaných vlastných akcií. Definícia, ich zápis a predávanie parametrov, je veľmi podobná princípom z jazyka C. Parametre sú vyčítané a predané akciám z vybraného záznamu z príslušnej tabuľky. Parametre sú pritom vkladané do tabuľky pri vytváraní jej záznamov. Nasledujúci príklad takejto akcie je prevzatý z [8]:

```
1  // Deklarácia užívateľskej akcie
2  action add_mTag(up1, up2, down1, down2, egr_port) {
3      // Pridanie halvičky typu mTag
4      add_header(mTag);
5      // Prekopírovanie položky VLAN ethertype do hlavičky mTag
6      copy_field(mTag.ethertype, vlan.ethertype);
7      // Nastavenie hodnoty VLAN's ethertype na požadovanú hodnotu
8      copy_field(vlan.ethertype,0xaaaa);
9      // Nastavenie ostatných hodnôt hlavičky mTag
10     set_field(mTag.up1, up1);
11     set_field(mTag.up2, up2);
12     set_field(mTag.down1, down1);
13     set_field(mTag.down2, down2);
14
```

```
15      // Výber požadovaného výstupného rozhrania
16      set_field(metadata.egress_poert, egr_port);
17  }
```

Výpis 4.6: Deklarácia užívatelskej akcie v jazyku $P4_{14}$.

Pri tejto akcii sa za hlavičku VLAN vloží hlavička mTag. Hlavička mTak už bola predom definovaná užívateľom. Z hlavičky VLAN sa skopíruje hodnota políčka ethertype do rovnakého políčka v mTag. Hodnota ethertype sa následne u VLAN tagu nastaví na hodnotu 0xaaaa, čo bude znamenať že za VLAN hlavičkou nasleduje mTag. Následne sa nastavujú ďalšie políčka hlavičky mTag. Na poslednom riadku sa definuje výstupný port (pole *metadata.egress_spec*). Akcie *add_header, set_field* a *copy_field* patria medzi primitívne akcie jazyka. Primitívne akcie umožňujú pridávanie a odoberanie hlavičiek, získavanie a modifikáciu jednotlivých položiek, bitové a aritmetické operácie alebo akcie nad celým paketom ako je jeho zahodenie. Ich kompletný zoznam a popis je možné nájsť v špecifikácii [29].

Jazyk taktiež definuje konštrukcie register, counter a meter. Sú to štruktúry ktoré dokážu uchovať stav dlhšie ako pre jeden paket. Môžeme v nich počítať a akumulovať výsledky zo spracovania jednotlivých paketov. K týmto štruktúram alebo pamäťovým bunkám majú prístup jednotlivé akcie. Vytvorenie registru može vyzerať nasledovne:

```
1  // Definícia registru
2  register tmp {
3      // Velkosť registeru v bitoch
4      width : 8;
5      // Počet registrov, instancí (pole)
6      instance_count : 10;
7  }
```

Výpis 4.7: Deklarácia registru v jazyku $P4_{14}$.

Na príklade môžeme vidieť vytvorenie pola registrov pomenované ako *tmp*, ktoré má 10 položiek a každá položka má veľkosť 8 bitov.

### 4.1.5 Špecifikácia toku dát

Posledným krokom je definovanie toho, akým spôsobom a v akom poradí budú jednotlivé tabuľky uplatňované. To docielime vytvorením kontrolného programu. Ako príklad môžeme uviesť nasledujúci príklad kontrolného programu:

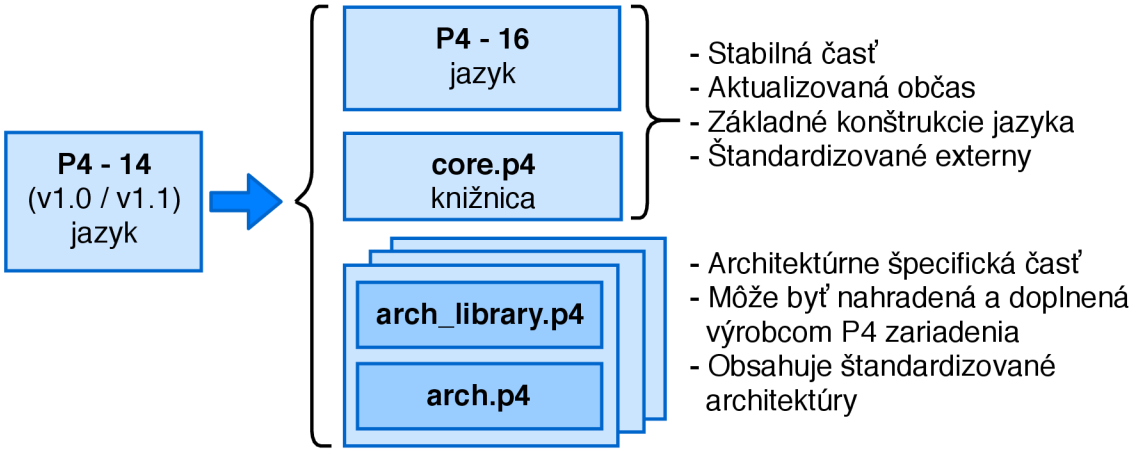
```
1  // Definícia kontrolného programu
2  control main() {
3      // Aplikovanie Match+Action tabuľky
4      apply(remove_mTag);
5      // Kontrola platnosti vyparsovanej hlavičky
6      if (valid(ipv5)){
7          apply(whitelist);
8      } else {
9          apply(drop);
10     }
11 }
```

Výpis 4.8: Definícia kontrolného programu v jazyku $P4_{14}$.

Tabuľky sú spúšťané príkazom *apply* za takých podmienok a v takom poradí, ako je uvedené v kontrolnom programe. V uvedenom príklade sa najprv uplatní tabuľka na odstránenie hlavičky protokolu mTag. Následne sa v konštrukcii *if{}else{}* zistí, či paket obsahuje protokol IPv5. V prípade že paket tento protokol obsahuje, spustí sa tabuľka *whitelist*, v opačnom prípade sa spustí tabuľka *drop*.

## 4.2 Štandart $P4_{16}$

V porovnaní so štandardom $P4_{14}$ prináša $P4_{16}$ niekoľko zásadných zmien. Vývoj z predchádzajúcej verzie je znázornený na obrázku 4.2. Prvá zmena je v tom že veľký počet jazykových




Obr. 4.2: Evolúcia jazyka P4 [30].

funkcií boli zo samotného jazyka odstránené a presunuté do samostatnej knižnice (pravá, horná časť obrázku). Tento presun sa týkal najme funkcií ako sú čítače, registre, jednotky pre výpočet kontrolných súčtov a mnoho ďalších. Pre tento účel bola zavedená nová konštrukcia jazyka nazývaná extern, pomocou ktorej sú tieto funkcie a jednotky definované. Výsledkom je transformácia z komplexného jazyka $P4_{14}$, obsahujúci viac ako 70 kľúčových slov, na relatívne malí jazyk $P4_{16}$ s menej ako 40 kľúčovými slovami. Súčasťou jazyka je taktiež knižnica zo základnými, štandardizovanými konštrukciami (extrenamy) používaných vo väčšine programov v jazyku P4.

Konštrukcia extern taktiež dovoľuje pridávať aj vlastné rozširujúce bloky, ktoré realizujú nejakú špecializovanú funkcionalitu. Tieto externy sú však závislé na danej platforme a výrobcovia ich musia dodávať vo forme knižnice spolu so samotným prekladačom jazyka P4, pre dané zariadenie (pravá, spodná časť obrázku). Pomocou externu definujeme iba rozhranie daného bloku spolu s popisom toho, čo daný extern vykonáva. O následné doplnenie funkcionality externu, na miesto v programe P4 kde sa používa, sa stará prekladač dodaný s príslušným zariadením. Príklad definície externu môže vyzerať nasledovne:

```
1 // Definícia externu dodaného so zariadením a jeho prekladačom
2 // Jednotka pre výpočet kontrolného súčtu
3 extern Checksum16 {
4     Checksum16();  // Konštruktor
5     void clear();  // Pripravenie jednotky na výpočet
6     void update<T>(in T data);  // Pridanie dát do výpočtu
7     bit<16> get();  // Vrátenie výsledku pre pridané data
8 }
```

Výpis 4.9: Definícia externu v jazyku $P4_{16}$.

Na príklade môžeme vidieť pridanie externu *Checksum*16, ktorý obsahuje konštruktor, funkciu *clear*() na vyčistenie jednotky z predchádzajúceho výpočtu, funkciu *update*() na pridanie požadovanej položky do výpočtu a funkciu *get*(), ktorá nám vráti výsledok.

Ďalšiu významnou zmenou štandardu $P4_{16}$ je že sa tu nenachádza žiadny abstraktný model, ktorého sa musí samotný jazyk a programy P4 držať. Tieto modely, architektúry,

dodáva výrobca spolu s P4 zariadením. Pre zariadenie môže byť dostupné niekoľko rozličných architektúr. Dostupných je taktiež niekoľko štandardizovaných architektú, ktoré by mali byť podporované na väčšine P4 zariadení. Pre tento účel zavádza jazyk $P4_{16}$ opakovane použiteľné bloky pre popis jednotlivých, programovateľných častí jazyka. Sú to parser, state, control a package. Pomocou týchto konštrukcií sa deklarujú prototypy blokov ako je parser, deparser, kontrola toku dát (postupnosť Match+Action tabuliek) a podobne. Pomocou konštrukcie package si následne vytvorime architektúru, ktorá obsahuje definovanú postupnosť z už vopred deklarovaných blokov, prototypov. Samotnú definíciu a teda funkcionalitu týchto blokov vytvára až užívateľ pri písaní programu P4, pre danú architektúru. Vytvorenia takejto architektúry môže vyzerať nasledovne:

```
// Pridanie štandardizovaných konštrukcií jazyka P4_16.
# include <core.p4>

// Prototyp parseru.
// @param <H> - typ hlavičiek - definované užívateľom
// @param b - Vstupný paket
// @param parsedHeaders - Výstup parseru, získané hlavičky
//
parser Parser<H>(packet_in b, out H parsedHeaders);

// Postupnsoť Match+Action tabuliek
// @param <H> - typ hlavičiek - definované užívateľom
// @param headers - hlavičky získané z parseru
control Pipe<H>(inout H headers);

// Prototyp Deparseru.
// @param <H> - typ hlavičiek - definované užívateľom
// @param b - výstupní paket
// @param outputHeaders - získané, upravené hlavičky
//
control Deparser<H>(inout H outputHeaders, packet_out b);

// Deklarácia modelu ktorý musí byť použitý užívateľom.
// Definuje postupnosť volania jednotlivých blokov
// @param <H> - typ hlavičiek - definované užívateľom
//
package ARCH_1<H>(
    Parser<H> p, // Najprv sa zavolá parser.
    Pipe<H> m, // Následne sa volá postupnosť Match+Action tabuliek.
    Deparser<H> d); // Na koniec sa volá deparser.
}
```

Výpis 4.10: Definícia architektúry v jazyku $P4_{16}$.

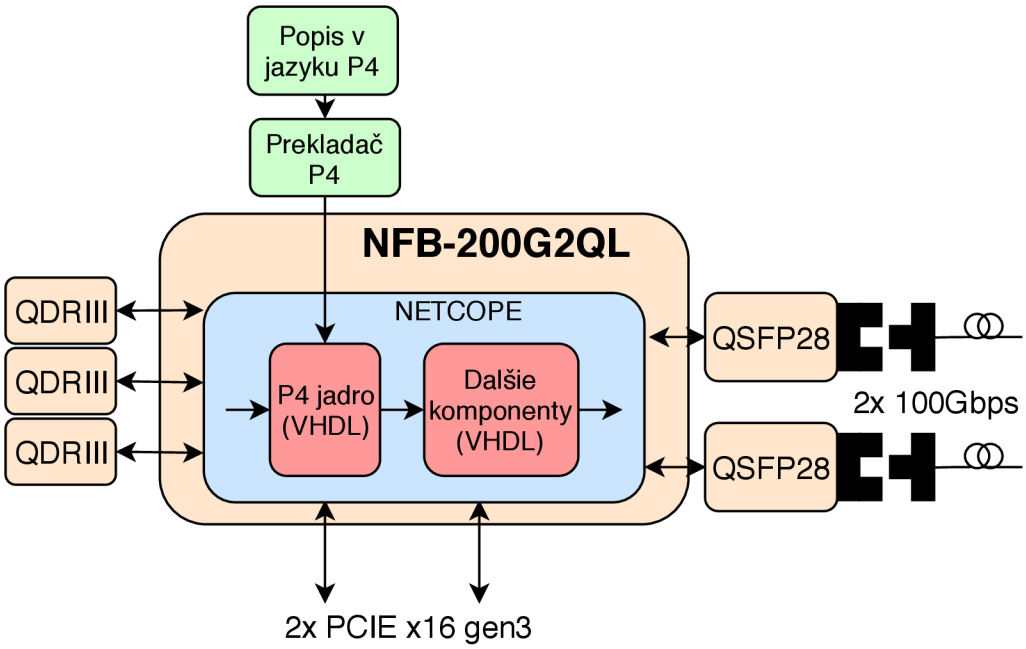
Na príklade vidíme definíciu prototypov z ktorých sa bude skladať naša výsledná architektúra. Najprv je definovaný parser, následne prototyp sekcie pre Match+Action tabuľky a na záver deparser. Z týchto konštrukcií je následne vytvorená architektúra $ARCH\_1$.

Hlavnými výhodami nového štandardu $P4_{16}$ je že výrobca P4 zariadenia nieje odkázaný iba na konštrukcie samotného jazyka ale môže si špecializované bloky dodať pomocou externov. Taktiež nie sme viazaný jedným typom architektúry, ako to je u štandardu $P4_{14}$ a jeho abstraktného modelu, ale môžeme si definovať vlastné architektúry, ktoré bude zariadenie podporovať. Užívateľ si tak zvolí P4 zariadenie, vhodnú architektúru, ktorú zariadenie podporuje, a v jazyku P4 si popíše jeho funkcionalitu obdobne ako ako u staršieho štandardu $P4_{14}$. Princíp popisu jednotlivých konštrukcií (stavy automatu, akcie, hlavičky, atď.) ostáva

rovnaký, môže sa však líšiť v syntaktickom zápise. Kompletná špecifikácia jazyka $P4_{16}$ je dostupná na [30].

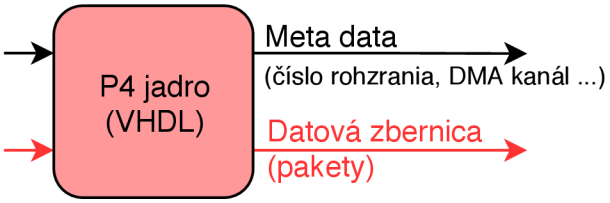
## 4.3 Prekladač jazyka P4 pre FPGA

Úlohou takéhoto prekladača je previesť popis zariadenia v jazyku P4, do popisu obvodu v jazyku VHDL, ktorý je následne možne syntetizovať na čip FPGA. V tejto práci sa budeme zaoberať prekladačom pre hardvérové akcelerátory COMBO a NFB. Podporou jazyka P4 a vývojom prekladača pre tieto akcelerátory sa zaoberá vedecko výskumná skupina Liberouter pod vedením Ing. Pavel Benáček, Ph.D. [19, 8]. Tento prekladač prevádza popis jazyka P4 na VHDL komponentu (firmware), v tejto práci nazývanú ako P4 jadro. Toto výsledné P4 jadro (ľavá, červená časť obrázku 4.3) je následne možné zapojiť ako VHDL komponetu do platformy NetCOPE (modrá časť obrázku) a preložiť na hardvérové akcelerátory NFB (oranžová časť obrázku).




Obr. 4.3: Zapojenie p4 jadra v platforme NetCOPE na akcelerátore NFB.

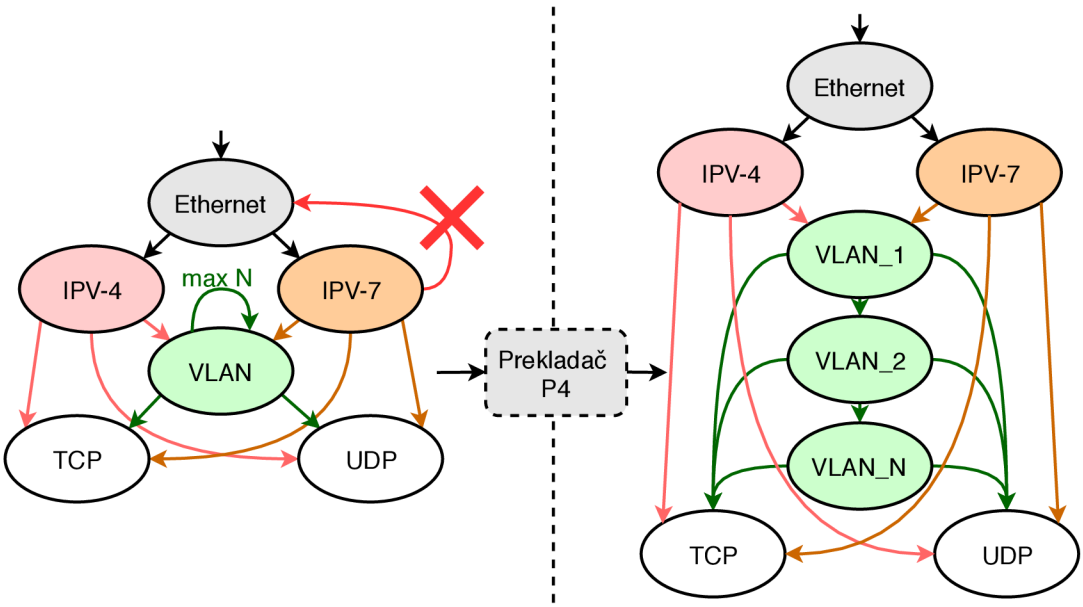
Aby sme boli schopný jednotlivé časti zariadenia, na ochranu pred (D)DoS, previesť do jazyka P4, musíme sa podrobne oboznámiť s vlastnosťami a nedostatkami samotného jazyka P4 a prekladača tohto jazyka pre akcelerátory NFB.




Obr. 4.4: Rozhranie P4 VHDL jadra.

Prvým významným obmedzením je že aktuálne dostupný prekladač podporuje iba štandard $P4_{14}$. Prekladač pre štandard $P4_{16}$ je v súčasnosti vo vývoji, v aktuálnom stave však nieje prakticky použiteľný. Komponenty, ktoré nebudeme vedieť popísať iba prostredníctvom štandardu $P4_{14}$, budú v novej architektúre zariadenia dočasne zapojené za jadrom P4 (pravá, červená časť obrázku 4.3). Pre tieto komponety bude realizovaný návrh popisu v jazyku $P4_{16}$ . Prakticky ale nebudú realizované v jadre P4, pretože súčasný prekladač tento štandard nepodporuje.

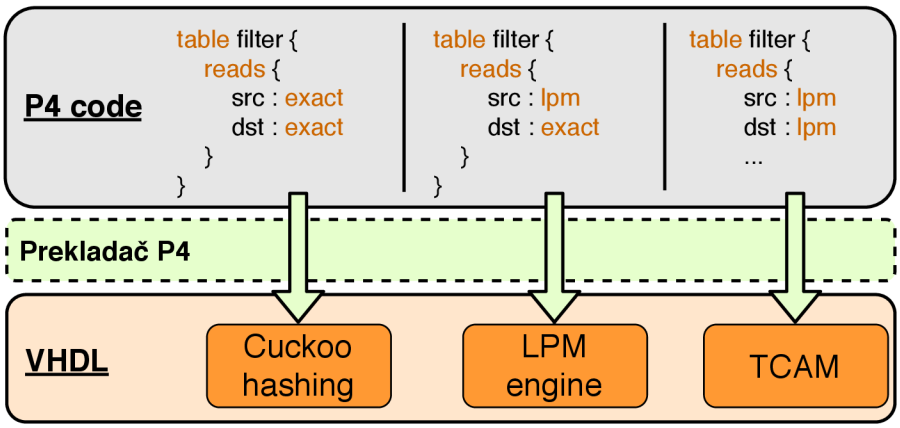
Ďalej je potrebné si uvedomiť, že vstup a výstup jadra P4, má iba jedno vstupné a výstupné rozhranie pre sieťové pakety (spodná časť obrázku 4.4) a jedno pre takzvané metadata (horná časť obrázku). Na rozhraní metadát, sa pre každý paket, prenášajú informácie ako je číslo sieťového rozhrania, číslo kanálu DMA, veľkosť paketu a podobne. V prípade že budeme pracovať s viacerými sieťovými rozhraniami, musíme distribúciu paketov na príslušné rozhrania realizovať mimo jadro P4. Informáciu o tom, na ktoré sieťové rozhrania má paket prísť, nám P4 jadro poskytne prostredníctvom metadát.




Obr. 4.5: Príklad automatu na parsovanie sieťových paketov v jazyku P4.

Dalším obmedzením prekladača je že automat, ktorý popisuje spôsob parsovania paketov, musí byť acyklický. Automat, ktorý obsahuje spätnú väzbu (červená šípka na obrázku 4.5), nebude môcť prekladač preložiť do popisu v jazyku VHDL. Výnimkou je spätná väzba samého do seba. V tomto prípade musí programátor priamo vyjadriť, koľko krát maximálne sa môže tento stav automatu opakovať. Prekladač na základe tejto informácie vie koľkokrát má tento stav zduplikovať a vytvoriť tak automat bez spätných väzieb (pravá strana obrázku 4.5)

Pri preklade Match+Action tabuliek využíva prekladač niekoľko rozličných implementácií vyhľadávacích tabuliek v jazyku VHDL. To ktorá implementácia bude použitá záleží na množstve a typu vyhľadávaní položiek, ktoré má tabuľka realizovať. V prípade že bude Match+Action tabuľka vyžadovať viac vyhľadávaní typu LPM alebo ternary, použije sa implementácia pomocou pamäte typu TCAM (Ternary Content Addres-sable Memory [33]) (pravá časť obrázku 4.6). Implementácia pomocou TCAM umožňuje realizovať všetky druhy vyhľadávaní, ktoré može Match+Action tabuľka požadovať. Táto implementácia je však ná-




Obr. 4.6: Prevod Match+Action tabuliek na VHDL komponenty.

ročná na zdroje čipu FPGA. Preto ak je to možné, tak sa zvolí menej náročná implementácia v jazyku VHDL. Aktuálne prekladač obsahuje dve ďalšie, menej náročné, implementácie a to LPM engine, ktorý dokáže realizovať jedno vyhľadávanie typu LPM (prostredná časť obrázku) a Cuckoo hashing, ktorý sa použije v prípade že chceme vyhľadávať iba na presnú zhodu hodnôt, exact (ľavá časť obrázku).

# Kapitola 5

# Návrh zariadenia

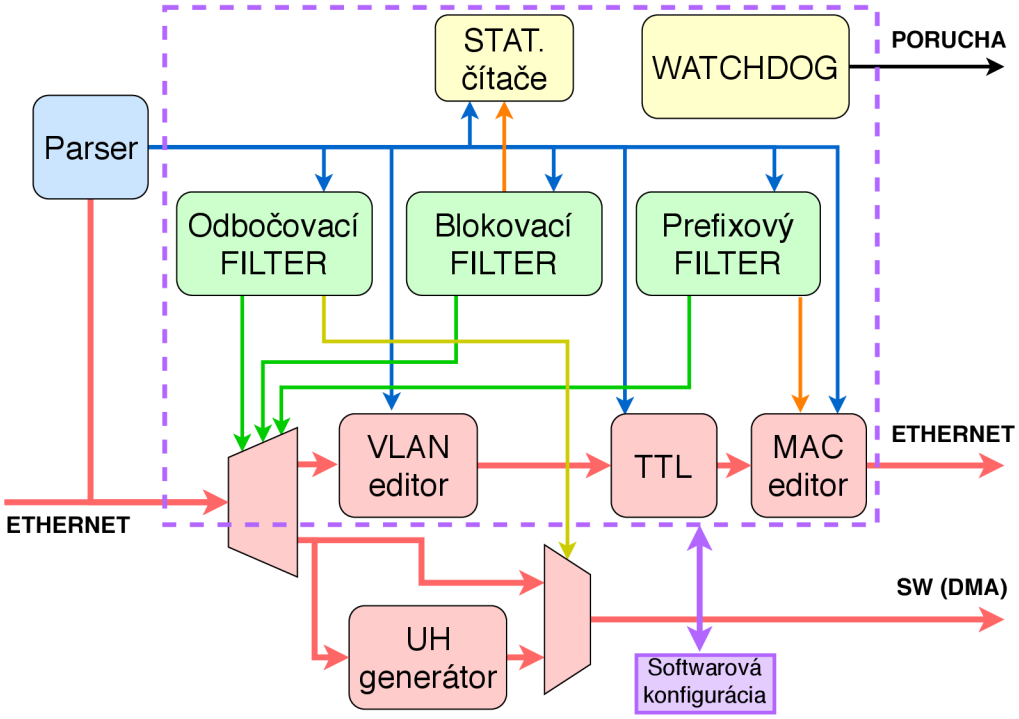
Kapitola sa zaoberá návrhom architektúry zariadenia s využitím jazyka P4 (kapitola 4), ktoré bude slúžiť na ochranu voči (D)DoS útokom (kapitola 2.1) a ktoré bude fungovať na platforme hardvérových akcelerátorov NFB (kapitola 3.3). Hlavným cieľom je tak vytvoriť novú a flexibilnejšiu architektúru, ktorá následne nahradí aktuálne používané riešenie (kapitola 3.4), ktoré už nevyhovuje súčasným podmienkam a nárokom, ktoré sú na zariadenie kladené, predovšetkým to je flexibilita a jednoduchá rozšíriteľnosť zariadenia.

V prvej časti sa kapitola venuje rozboru jednotlivých častí (komponent) zariadenia a spôsobu akým ich je alebo nie je možné realizovať prostredníctvom jazyka P4. Na základe týchto zistení a uskutočneného návrhu, sa kapitola ďalej venuje návrhu architektúry celého zariadenia, ktorá v sebe bude zahrnovať možnosť používania nových častí popísaných v jazyku P4.

## 5.1 Popis komponent zariadenia v jazyku P4

Aby sme boli schopný nahradiť súčasné, nevyhovujúce riešenie novou architektúrou, je potrebné aby sme zachovali celú funkcionalitu, ktorú už aktuálne riešenie poskytuje. Preto sa v tejto kapitole budeme venovať návrhu, akým spôsobom bude alebo nebude možné jednotlivé časti zariadenia realizovať prostredníctvom jazyka P4.

Ako prvé sa musíme pozrieť z akých jednotiek sa zariadenie skladá. Zjednodušená schéma súčasnej architektúry zariadenia je zobrazená na obrázku 5.1. Prvou a hlavnou komponentou celého zariadenia je parser (modrá časť obrázku), ktorá získava potrebné údaje z paketov. Ďalšími komonentami zariadenia sú jednotky ktoré realizujú filtráciu na základe nakonfigurovaných pravidiel (zelená časť obrázku). Na základe ich výsledku sa rozhoduje na aké rozhranie a v akom formáte bude daný paket zaslaný. Sem patrí odbočovací filter, blokovací filter a prefixový filter. Druhou kategóriou sú komponenty ktoré vykonávajú úpravu samotných paketov na dátovej zbernici (červená časť obrázku). Sem patrí jednotka na výmenu VLAN hlavičiek, dekrementácia hodnoty TTL, editor MAC adries a generátor hlavičiek UH. Špeciálnou kategóriou sú komponenty ako sú štatistické čítače a watchdog (žltá časť obrázku). Ich úlohou je uchovávať a poskytovať užívateľovi štatistické a stavové informácie o aktuálnom stave zariadenia. Podrobné špecifikácie a popis implementácie jednotlivých komponent, v jazyku VHDL, sú uvedené v bakalárskej práci [18].




Obr. 5.1: Architektúra súčastného zariadenia na ochranu proti (D)DoS.

### 5.1.1 Parser HFE (Header Field Extractor)

Ide o komponentu, ktorá v súčasne používanom riešení, slúži na získanie údajov z hlavičiek jednotlivých sieťových protokolov. Táto komponenta tak musí byť schopná postupne extrahovať informácie z jednotlivých úrovní zapuzdrenia protokolov. Okrem samotných dát sú k vybraným položkám poskytované informácie o ich polohe v prenášaných dátach (paketoch) na dátovej zbernici. To je realizované prostredníctvom ofsetu (adresy), ktorý obsahuje informácie o poradovom čísle dátového slova a číslo konkrétneho bajtu na dátovej zbernici. Informácie o polohe danej hlavičky je využívaná v moduloch uskutočňujúcich modifikáciu paketu na dátovej zbernici.

V jazyku P4 je celý paket rozkladný na protokolové hlavičky a na konci je z nich opätovne zložený. Samotné pakety už niesu prenášané prostredníctvom samotnej dátovej zbernice. Informácie o polohe jednotlivých hlavičiek a dát v pakete, už tak nebudú v novom zariadení potrebné. Modifikáciu jednotlivých dát budeme vykonávať priamo v získaných hlavičkách.

Pre naše nové zariadenie tak môžeme parser bez problémov popísať konečným automatom tak, ako to bolo vysvetlené v kapitole 4.1.2. Pri popise parseru v jazyku P4 ale musíme dbať na to, aby výsledný automat bol acyklický. Automat obsahujúci spätnú väzbu by zo súčasne dostupným prekladačom nebolo možné preložiť na popis v jazyku VHDL (kapitola 4.3). Výnimkou je spätná väzba samého do seba, ako je napríklad vidieť pri parsovaní hlavičiek VLAN:

```
1 // Počet podporovaných VLAN hlavičiek
2 #define VLAN_DEPTH 4
3 // Konštrukcia pre podporu viacnásobnej hlavičky daného protokolu
4 header vlan_t vlan_[VLAN_DEPTH];
5 // Stav pre získanie hlavičiek VLAN
6 parser parse_vlan {
```

```
7       extract(vlan_[next]);
8       return select(latest.etherType) {
9           ETHERTYPE_VLAN : parse_vlan; // Spätná väzba samého do seba
10          ETHERTYPE_IPV4 : parse_ipv4;
11          ETHERTYPE_IPV6 : parse_ipv6;
12          ETHERTYPE_MPLS : parse_mpls;
13      }
14  }
```

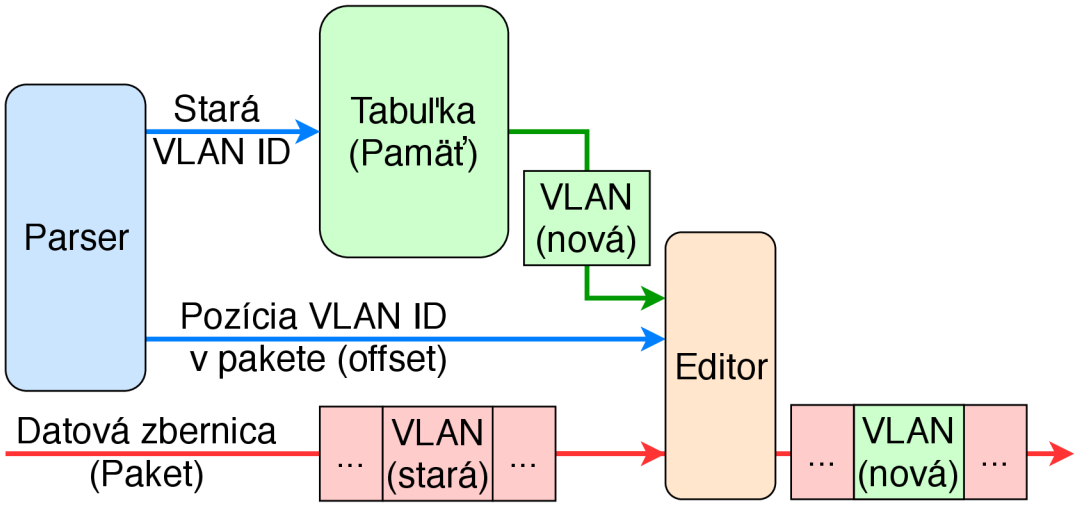
Výpis 5.1: Stav automatu na parsovanie VLAN hlavičiek v jazyku P4.

V tomto prípade nám prekladač umožňuje požiť špeciálnu konštrukciu *header [header_type] [name]_[size]*, kde hodnota *[size]* udáva koľkokrát sa môže hlavička VLAN v pakete nachádzať. Prekladač z tejto konštrukcie vygeneruje odpovedajúci počet nových stavov tak, aby bol výsledný automat acyklický. Z uvedeného príkladu pre VLAN hlavičky, prekladač vygeneruje celkom štyri stavy, pre možnosť získania až štyroch VLAN hlavičiek z paketu.

Komponentu na parsovanie paketov, pre naše zariadenie, tak môžeme kompletne realizovať prostredníctvom jazyka P4.

### 5.1.2 Modul pre výmenu VLAN

Na základe hodnoty hlavičky VLAN sa vykoná jej výmena za novú, užívateľom definovanú. Modul konkrétne realizuje výmenu dvanásť bitovej položky ID, ktorá je jednou z hodnôt v hlavičke protokolu VLAN. Hodnota ID a jej poloha v pakete je získavaná z parseru (modrá časť obrázku 5.2). Následne sa vyhľadá záznam v príslušnej tabuľke, ktorý nám určí novú hodnotu položky ID (zelená časť). Ako adresa do tejto tabuľky sa použije pôvodná hodnota ID, získaná z parseru. Následne sa na dátovej zbernici, kadiaľ nám chodí celý paket, vykoná výmena položky ID za novú (pravá časť).




Obr. 5.2: Principiálne schéma modulu pre výmenu VLAN ID.

V jazyku P4 môžeme túto funkcionalitu realizovať ako jednu Match+Action tabuľku. Za základe hodnoty ID bude v tabuľke vyhľadaná nová hodnota ID a následne sa zavolá akcia, ktorá túto výmenu v hlavičke VLAN skutoční. Aby sme mohli pokryť všetky možné hodnoty položky ID, musí mať babuľka kapacitu aspoň $2^{12}$ (4096) záznamov. Jeden záznam pre každú možnú hodnotu ID. Zápis komponenty v jazyku P4 môže vyzerať nasledovne:

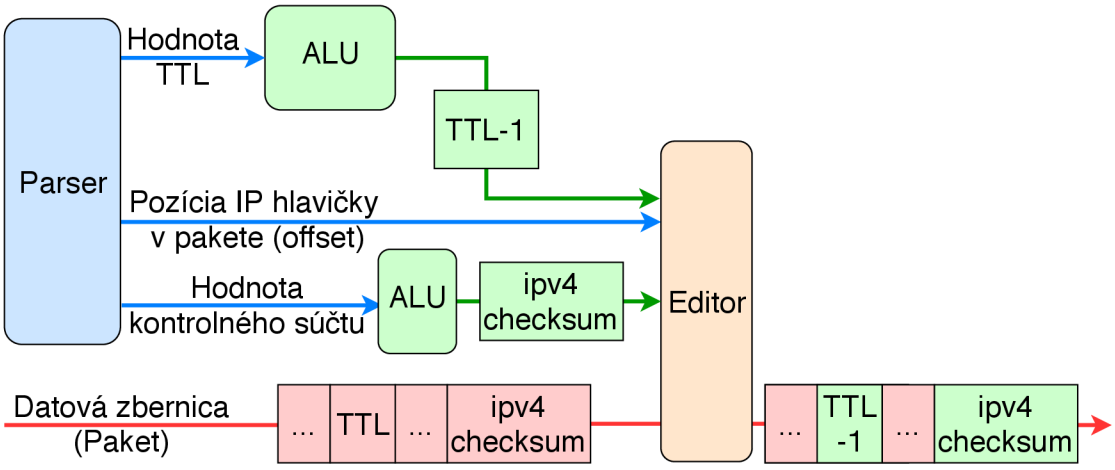
```
// Vymení starú hodnotu ID v hlavičke VLAN za novú
action replace_vlan_tag(in bit<12> new_vlan_tag) {
    modify_field(vlan_tag.vid, new_vlan_tag);
}

table table_vlan_replace {
    reads {
        // Nájdeme odpovedajúci záznam
        vlan_tag.vid : exact;
    }
    actions {
        // Vykonáme výmenu
        repalce_vlan_tag;
        // Nevykonáme nič
        NoOp;
    }
    // 4096 možných hodnôt ID = 4096 možných záznamov
    max_size: 4096;
}
```

Výpis 5.2: Výmena hodnôt VLAN ID v jazyku P4.

Tabuľka *table_vlan_replace* vyhľadáva záznam podla položky ID v hlavičke VLAN. Na základe nájdenia záznamu môže tabuľka buďto vykonať požadované nahradenie pomocou akcie *replace_vlan_tag* alebo nevykoná nič. V tomto prípade sa volá prázdna akcia NoOp, ktorá patrí medzi primitívne akcie.

### 5.1.3 Modul pre kontrolu TTL

Modul získa hodnotu TTL a offset začiatku hlavičky IP v pakete z parseru (modrá časť obrázku 5.3). Hodnotu TTL a prípadne kontrolný súčet protokolu IPv4, upraví pomocou ALU jednotiek (zelená časť). Následne nimi nahradí pôvodné hodnoty v pakete (spodná časť). V prípade protokolu IPv6 sa bude jednať o položku Hop Limit. Následne komponenta realizuje prepočet kontrolného súčtu v prípade protokolu IPv4.




Obr. 5.3: Principiálne schéma modulu pre dekrementáciu TTL.

Popis komponenty pre dekrementáciu hodnoty TTL protokolu IPv4, bude v jazyku P4 vyzerať nasledovne:

```
// Zoznam položiek z ktorých budme počítať kontrolný súčet
field_list ipv4_fields {
    ipv4.srcAddr;
    ...
}

// Funkcia na prepočet kontrolného súčtu
field_list_calculation ipv4_csum {
    input{
        // zvolíme dané položky
        ipv4_fields;
    }
    // zvolíme algoritmus a šírku výstupu
    algorithm : csum16;
    output_width : 16;
}

// Akcia na dekrementáciu TTL
action decrement_ttl () {
    // Znížime TTL o 1
    add_to_field(ipv4.ttl,-1);
    // Prepočítame kontrolný súčet
    modify_field_with_hash_based_offset(ipv4.checksum,0,ipv4_csum,65536);
}

// Tabuľka ktorá uplatní danú akciu na každý paket
table table_decrement_ttl {
    actions {
        decrement_ttl;
        NoOp;
    }
}
```

Výpis 5.3: Dekrementácia hodnoty TTL v jazyku P4.

Komponenta je reprezentovaná ako Match+Action tabuľka *table_decrement_ttl*, ktorá uplatňuje danú akciu na všetky pakety. Akcia *decrement_ttl* následne zníži hodnotu TTL v ipv4 hlavičke o 1. Ďalej je potrebné opätovne prepočítať kontrolný súčet IP hlavičky. K tomuto účelu slúži funkcia *ipv4_csum*. Zvolíme algoritmus *algorithm : csum16*, ktorý realizuje nami požadovanú operáciu. Následne je táto hodnota v akcii *decrement_ttl* použitá na nahradenie starého kontrolného súčtu v hlavičke IPv4. Predtým je však hodnota TTL znížená o 1 v príkaze *add_to_field*.

### 5.1.4 MAC editor

Komponenta vykonáva prepis zdrojovej a cieľovej MAC adresy za nové alebo uskutočňuje ich vzájomnú výmenu. Zdrojová a cieľová MAC adresa sú získavané z parseru, obdobne ako v predchádzajúcich komponentách.

Výmena MAC adries bude v P4 popísaná pomocou jednej Match+Action tabulky, ktorá bude obsahovať vždy iba jeden záznam a zvolenú akciu tak uplatní na každý vstupný paket. V jazyku P4 by táto komponetna vyzerala nasledovne:

```
// Pomocný register
register tmp_mac {
    // Veľkosť v bitoch
    width    : 48;
    // Dostupné iba z jednej tabuľky
```

```
6       static : mac_replace;
7       // Iba jeden register
8       instance_count : 1;
9   }
10
11  action swap_mac() {
12      // Prekopírovanie DST do registru
13      register_write(tmp_mac, 0, ethernet.dst_addr);
14      // Prekopírovanie SRC do DST
15      modify_field(ethernet.dst_addr, ethernet.src_addr);
16      // Prekopírovanie DST z registru do SRC
17      register_read(ethernet.src_addr, tmp_mac, 0);
18  }
19
20  action new_mac(src, dst) {
21      // Skopírovanie nových hodnôt
22      modify_field(ethernet.dst_addr, dst);
23      modify_field(ethernet.src_addr, src);
24  }
25
26  // Tabuľka iba s jedným, defaultným záznamom
27  table mac_replace {
28      // Podporované akcie
29      actions {
30          swap_mac;
31          new_mac;
32          NoOp;
33      }
34  }
```

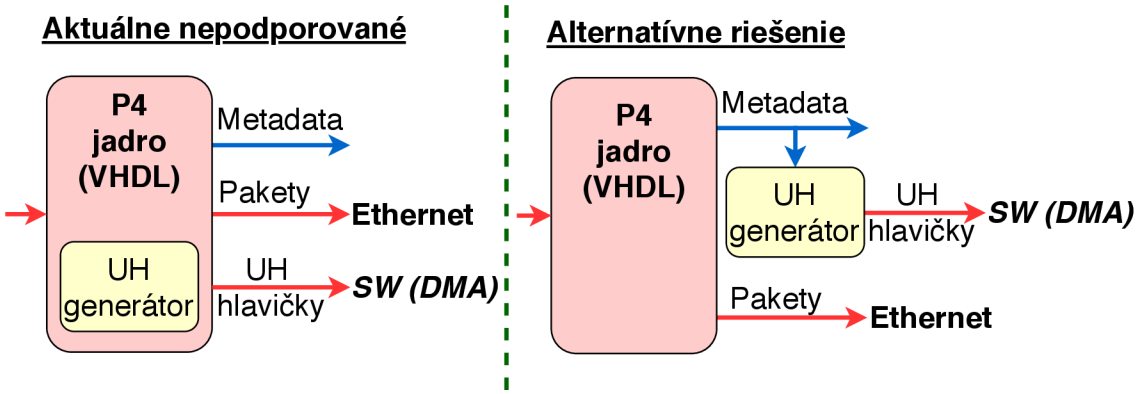
Výpis 5.4: Výmena MAC adries v jazyku P4.

Tabuľka *mac_replace* nerealizuje vyhľadávanie pre žiadnu položku. Obsahuje iba jeden možný záznam, ktorý sa uplatní pre všetky pakety. Záznam môže volať tri akcie. Akcia *new_mac* realizuje prepis nových MAC adries. Akcia *swap_mac* realizuje vzájomnú výmenu zdrojovej a cieľovej MAC adresy. K tomuto účelu využíva akcia jeden pomocný register *tmp_mac*. V prípade že nechceme robiť žiadnu akciu, bude mať záznam nastavenú primitívnu akciu *NoOp*.

### 5.1.5 UH generátor

Jedná sa o komponentu ktorá zoberie vybrané položky získané z paketu (ako sú napríklad IP adresy, protokoly, porty, MAC adresy a podobne) a následne z nich zloží výrazne menší dátový paket v požadovanom formáte, nazývaný ako UH hlavička. Tento špeciálny paket sa následne môže poslať do software miesto pôvodného paketu. Výsledkom je že software sa už nemusí zaoberať parsovaním celého paketu, ale rovno získa potrebné údaje v požadovanom formáte.

Pri prevode tejto komponenty do jazyka P4 narážame na obmedzenie samotného prekladača. Výsledne jadro P4 nám poskytuje vždy iba jedno dátové rozhranie pre pakety a jedno rozhranie pre metadata (popísané v kapitole 4.3). Pre daný paket tak nie sme schopný na tomto jednom rozhraní naraz poskytnúť jak dáta samotného paketu, tak dáta vytvorenej hlavičky UH. K tomuto účelu by sme potrebovali aspoň dve dátové rozhrania P4 jadra (ľavá časť obrázku 5.4). V prípade že budeme chcieť posielať do software UH hlavičky miesto paketov, budeme musieť tieto UH hlavičky vytvárať za jadrom P4 v špeciálnej komponente, ktorá vytvori hlavičky UH z poskytnutých metadát jadra P4 (pravá časť obrázku).




Obr. 5.4: Spôsob zapojenia UH generátoru.

Položky metadat, z ktorých má byť následne vytvorená hlavička UH, môže užívateľ zvoliť tak že v jazyku P4 pridá do potrebnej štruktúry nové položky. Hodnotu týchto položiek následne nastaví pomocou nejakej Match+Action tabuľky. Príklad pridania hodnoty zdrojovej IP adresy na výstup týchto metadat by vyzeral nasledovne:

```
// Datová štruktúra metadat
header_type metadata_t {
    fields {
        ...
        // Pridanie novej položky do štruktúry
        IPv4_src : 4;
        ...
    }
}
// Metadata pomenované ako metadata, budú dostupné na výstupe P4 jadra
metadata metadata_t metadata;

// Akcia na nahranie hodnôt do metadat
action set_metadata (){
    ...
    // nakopírovanie požadovanej hodnoty
    modify_field(metadata.IPv4_src, ipv4.src)
    ...
}

// Match+Action tabulka nastavý metadata pre každý paket
table table_set_metadata {
    actions {
        set_metadata;
    }
}
```

Výpis 5.5: Rozšírenie metadát o vlastnú položku v jazyku P4.

Match+Action tabuľka *table_set_metadata* vykoná nastavenie požadovaných položiek, pre každý paket, prostredníctvom akcie *set_metadata*.

Táto komponenta bude plne realizovateľná pomocou štandardu $P4_{16}$, pretože už nie sme viazaný pevne daným abstraktným modelom. Môžeme si vytvoriť vlastnú architektúru, ktorá bude obsahovať viacero výstupných rozhraní.

### 5.1.6 Odbočovací filter

Ide o filter, ktorý podla vybraných položiek jednotlivých protokolov, určuje na ktoré sieťové rozhrania bude paket doručený alebo či má byť paket zahodení. Filter taktiež určuje aké ďalšie operácie nad paketom, majú vykonať nasledujúce komponenty (napr. výmena hodnoty VLAN, výmena MAC adresy apod.).

Aktuálne dostupný prekladač jazyka P4 na VHDL, využíva niekoho rôznych implementácií pre realizáciu vyhľadávacích tabuliek, ako bolo popísané v kapitole 4.3. Hlavným problémom tohoto filtru je to, že je potrebné realizovať až dve vyhľadávania typu LPM, pre zdrojovú a cieľovú IP adresu. Počet vyhľadávaní typu LPM priamo súvisí s tým aká implementácia bude použitá.

V prípade že budeme chcieť filter, ktorý bude obsahovať rádovo desiatky pravidiel, môžeme celý filter popísať ako jednu veľkú Match+Action tabuľku, kde všetky položky budú vyhľadávané ako ternary (obecnejšia verzia algoritmu LPM). Z tejto konštrukcie vytvorí prekladač kód VHDL, ktorý bude používať pamäť typu TCAM (Ternary Content Addressable Memory) [33]. Táto pamäť je značne náročná na zdroje čipu FPGA a je preto vhodná iba pre malí počet položiek.

Ak budeme potrebovať rádovo tisíce pravidiel, tak musíme zvoliť taký popis v jazyku P4, ktorý umožní prekladaču zvoliť implementáciu, ktorá bude výrazne menej náročnejšia a optimalizovanejšia na zdroje obvodu FPGA. To docielime tým, že Match+Action tabuľka bude obsahovať maximálne jedno vyhľadávanie typu LPM. Celý filter, ktorý obsahuje niekoľko LPM, tak možeme realizovať viacerými, prepojenými Match+Action tabuľkami, kde každá z nich bude obsahovať iba jedno LPM.

V prípade nášho zariadenia nám bude stačiť kapacita rádovo desiatky pravidiel. Preto zvolíme jednoduchší popis v jazyku P4, ktorý bude prekladač realizovať pomocou pamäte TCAM. Popis filtru bude v jazyku P4 vyzerať nasledovne:

```
// Akcia na uloženie výsledku vyhľadania záznamu.
// Tieto údaje budú na výstupnom rozhraní dostupné spolu zo spracovanými paketmi.
action filter_record(ethSend, dmaSend, id, ...) {
    modify_field(metadata.uh, uh);
    modify_field(metadata.eth_send, ethSend);
    modify_field(metadata.id, id);
    ...
}

// Odbočovací filter
table filter_t {
    reads {
        // Zloženie vyhľadávacieho kľúča
        ethernet.valid      : ternary;
        ethernet.dstAddr    : ternary;
        ipv4.valid          : ternary;
        ipv4.srcAddr        : ternary;
        ...
    }
    actions {
        set_ifc_info;
        NoOp;
    }
    size : FILTER_SIZE; // Rádovo desiatky pravidiel
}
```

Výpis 5.6: Popis odbočovacieho filtru v jazyku P4.

Jedná sa o jednu Match+Action tabuľku, ktorá realizuje vyhľadávanie podla zvolených položiek a pomocou akcie *filter_record* nastavuje príslušné metadat na požadované hodnoty.

### 5.1.7 Štatistická jednotka

Cieľom komponenty je počítať počet bajtov a paketov blokovaných sieťových prenosov. Na základe týchto štatistík softvér uskutočnuje vyhodnocovanie a vykonáva prípadnú rekonfiguráciu jednotlivých komponent zariadenia. Celá jednotka je tvorená sadou adresovatelných čítačov, ktoré na príslušné miesto započítavajú počet bajtov a paketov. Čítač, ktorý bude pre daný paket použitý, je vybraný na základe nájdenia záznamu v príslušnom filtri. Týchto čítačov je potrebné mať veľké množstvo, rádovo tisíce až stotisíce, čo by využívalo značné pamäťové zdroje FPGA. Preto táto jednotka využíva externú pamäť, ktorá je na platforme NFB pripojená k čipu FPGA.

Súčasný prekladač jazyka $P4_{14}$ však neumožňuje používanie externých pamätí a preto nemôžme túto komponnetu aktuálne realizovať ani v jazyku P4. Vo výslednej architektúre tak bude táto komponenta dočasne zapojená za jadrom P4, ako je to znázornené na obrázku 4.3. Táto situácia sa bude dať vyriešiť až s prekladačom, ktorý bude podporovať štandard $P4_{16}$. V tomto prípade budeme môcť túto komponentu pridať ako extern pre platformu NFB, ktorý bude používať externú pamäť. Príklad takéhoto externu v jazyku $P4_{16}$ môže vyzerať nasledovne:

```
extern stats_unit {
    // Konštruktor
    sats_unit();
    // Inicializácia celej jednotky
    void clear();
    // Pripočítanie veľkosti paketu k danému čítačku
    void update(in bit<48> pac_size, in bit<24> addr);
}
```

Výpis 5.7: Štatistická jednotka ako extern v jazyku P4.

Pomocou funkcie *update* dochádza k prepočítavaniu požadovanej hodnoty na dané pamäťové miesto.

### 5.1.8 Blokovací filter

Jedná sa o filter ktorý slúži k zahadzovaniu nežiaducich sieťových dát na základe softwarovej konfigurácie. Ku každému záznamu v tabuľke je pridelený štatistický čítač v ktorom si zaznamenávame počet zahodených paketov a bajtov k daným záznamom. K tomuto účelu slúži komponenta popísaná v predošlom texte (kapitola 5.1.7). Filter musí byť schopný uchovať rádovo až sto tisíce záznamov, z tohoto dôvodu je aj tu využívaná externá pamäť nachádzajúca sa mimo čip FPGA. Preto ani táto komponenta nemôže byť aktuálne realizovaná v jazyku $P4_{14}$, pretože prekladač neumožňuje použitie externých pamätí. Dočasne bude komponenta, vo výslednej architektúre, zapojená za jadrom P4. Do budúcnosti však máme dve možné riešenia ako túto situáciu riešiť.

Prvá z možností je že s príchodom prekladača s podporou štandardu $P4_{16}$ budeme môcť túto komponentu pridať ako extern pre platformu NFB. Zápis filtru pomocou externu v štandarde $P4_{16}$ môže vyzerať nasledovne:

```
extern filter {
    filter(); // Konštruktor
    // Vykoná sa vyhľadanie záznamu
```

```
    // Vracia informáciu o úspechu a prípadne adresu do štatistickej jednotky
    bit<1> search<T>(in T data, out bit<24> addr);
}
```

Výpis 5.8: Blokovací filter ako extern v jazyku P4.

Extern obsahuje jednu funkciu *search*(), ktorá realizuje vyhľadávanie. V prípade že dôjde k zhode s niektorým záznamom, tak funkcia vráti aj adresu štatistického čítača.

Druhou variantov je že do samotného prekladača pridáme ďalšiu možnú implementáciu Match+Action tabuľky, ktorá bude využívať externé pamäte. Popis tabuľky by tak v jazyku P4 vyzeral nasledovne:

```
// Blokovací filter
#pragma FILTER_EXTERN_MEM
table filter_t {
    reads {
        ipv4.valid          : ternary;
        ipv4.srcAddr        : ternary;
        ...
    }
    actions {
        NoOp;
        ...
    }
    size : FILTER_SIZE;
}
```

Výpis 5.9: Blokovací filter ako Match+Action tabuľka v jazyku P4.

V tomto prípade by sa pomocou pragmy *FILTER_EXTERN_MEM* vynútila implementácia využívajúca externé pamäte.

### 5.1.9 Prefixový filter

Filter slúži k tomu aby sme mohli časť sieťových dát smerujúcich do niektorej z podsietí, ktorá je definovaná cieľovým prefixom, presmerovať tak aby prechádzalo nami vybraním sieťovým zariadením. To docielime zmenou cieľovej MAC adresy a pakety tak budú presmerované na príslušné zariadenie. Jedná sa o filter ktorý realizuje vyhľadávanie LPM (Longest Prefix Match) podľa cieľových IP adries. Výsledkom úspešnej zhody je nová cieľová MAC adresa, ktorá má byť v pakete nastavená. Ďalej filter môže rozhodnúť o tom či bude paket presmerovaní na sieťové rozhranie smerujúce do software alebo nie.

Prevod filtru do jazyka P4 je v tomto prípade pomerne jednoduchý a nenarážame na žiadne významné obmedzenia prekladača.

```
// Akcia ktorá vykoná výmenu MAC adresy a nastavenie výstupného rozhrania
action replace_mac_and_send_sw (new_addr, sw_send){
    // Nastavenie novej MAC adresy
    modify_field(ethernet.dst_addr, new_addr);
    // Poslanie paketu do software
    modify_field(metadata.sw_send, sw_send)
}

// Prefixový filter
table table_ipv4_filter {
    reads {
        ipv4.dstAddr : lpm;
    }
    actions {
```

```
15          repalce_mac_and_send_sw;
16          NoOp;
17      }
18      max_size: SIZE;
19  }
```

Výpis 5.10: Popis prefixového filtru v jazyku P4.

Jedná sa o jednu Match+Action tabuľku ktorá realizuje vyhľadávanie LPM podľa cieľovej IP adresy. Dostupná akcia umožňuje výmenu MAC adresy za novú a nastavenie výstupného rozhrania do software.

### 5.1.10 Watchdog

Jednotka ktorej úlohou je detektovať že nastala porucha a na základe tejto detekcie vykonať požadovanú akciu. Komponenta obsahuje čítač na meranie času, ktorý počíta počet taktov (hodinových cyklov). V prípade že čítač prekročí stanovenú hodnotu, zadanú v čase (sekundy, minúty), tak sa vyvolá signalizácia poruchy. Obslužný software, alebo samotný užívateľ, musí tento čítač v pravidelných intervaloch nulovať aby nedošlo k vyvolaniu poruchy na strane firmwaru.

Týmto spôsobom je firmware, fungujúci v obvode FPGA, schopný poznať že na strane softvérovej časti došlo k nejakej poruche (obslužný sowtware havaroval alebo sa zasekol) a tento stav je následne schopný indikovať aj na ostatných komunikačných kanáloch. Napríklad sa začne vysielať chybová hláška na rozhraniach ethernetu alebo sa úplne odpoja, na akceleračnej karte sa rozsvietia varovné led diódy a podobne.

V jazyku $P4_{14}$ nie sme schopný takúto funkcionalitu realizovať, nie sme schopný počítať počet hodinových cyklov a odvodzovať na základe toho ubehnutý čas. Komponenta watchdog tak musí byť zapojená za jadrom P4. V prípade štandardu $P4_{16}$ budeme môcť túto komponentu realizovať ako extern. Príklad externu by vyzeral nasledovne:

```
1  extern watchdog {
2      watchdog(bit<1> default_state); // Konštruktor
3  }
```

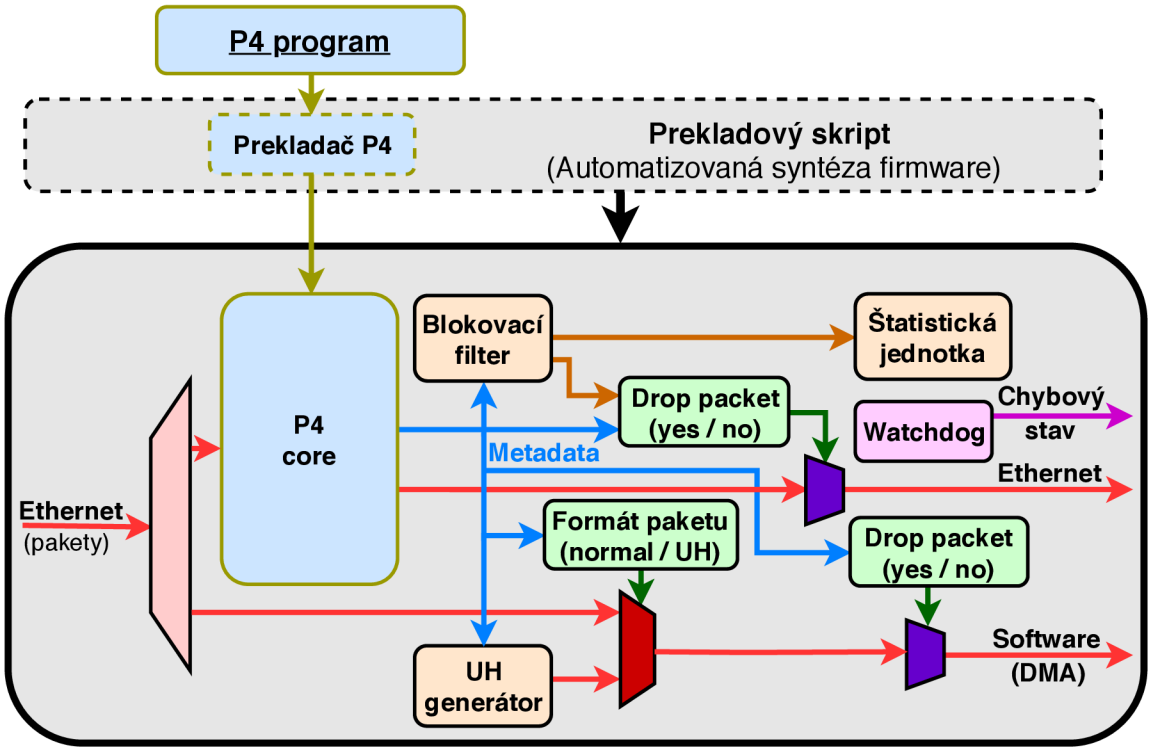
Výpis 5.11: Watchdog ako extern v jazyku P4.

Jedná sa o jednotku ktorá vôbec nesúvisí so spracovaním paketov a preto neobsahuje ani žiadne funkcie. Pomocou konštruktoru je ale možné nastaviť počiatočný stav tejto komponety, teda či bude zapnutá alebo nie. Instancia tohto externu následne spôsobí že sa vo výslednom firmware P4 objaví aj watchdog.

## 5.2 Architektúra zariadenia

Na základe uskutočneného návrhu jednotlivých časti zariadenia, môžeme vytvoriť návrh architektúry celého firmwaru zariadenia. Navrhovaný firmware bude následne zapojený ako aplikačné jadro do platformy NetCOPE (popísané v kapitole 3.3). Schéma výsledného firmware je zobrazená na obrázku 5.5.

Firmware obsahuje jedno vstupné rozhranie (červená šípka na ľavej strane obrázku 5.5) kadiaľ prichádzajú sieťové pakety tak ako boli prijaté na sieťovom rozhraní akceleračnej karty a spracované platformou NetCOPE. Na druhej strane firmware (pravá časť obrázku) sa nachádzajú dve výstupné sieťové rozhrania. Jedno z nich je určené na posielanie spracovaných paketov na výstupné rozhranie akceleračnej karty NFB. Druhé slúži k posielaniu



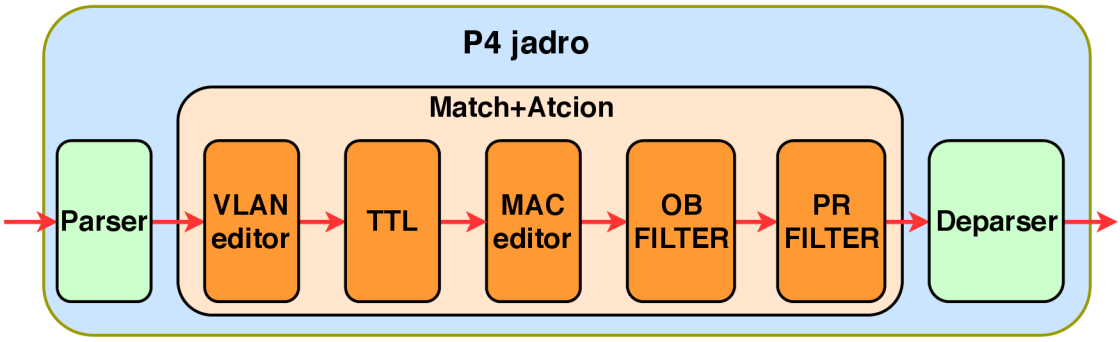

Obr. 5.5: Architektúra zariadenia na ochranu pred (D)DoS s využitím jazyka P4.

paketov do softwaru prostredníctvom DMA prenosov. Software požaduje nezmenené pakety tak ako boli prijaté na vstupnom rozhraní. Z tohto dôvodu je hneď na vstupe firmwaru zapojená komponenta, ktorá vstup rozdelí na dve nezávislé dátové cesty (červený demultiplexor na ľavej strane). Jedna je určená pre pakety smerujúce do software a druhá na výstup akceleračnej karty. Pakety na vstupe sú zduplikované na obe tieto dátové cesty.

Na softvérové rozhranie môže byť miesto sieťových paketov poslaná takzvaná hlavička UH, ktorá obsahuje údaje získané z metadat z výstupu P4 jadra. Komponenta pre generovanie tejto hlavičky (spodná oranžová časť obrázku) je popísaná v kapitole 5.1.2. Tesne za touto komponentov sa nachádza takzvaný paketový multiplexor (tmavo červený multiplexor). Multiplexor prenesie zo zvoleného vstupu na výstup sieťový paket a na všetkých ostatných vstupných rozhraniach daný paket zahodí. O tom či má byť paket do softwaru poslaný vo formáte hlavičky UH alebo nie, rozhoduje jadro P4. Multiplexor vyberie požadovaný vstup na základe vopred zvolenej položky z metadat jadra P4 (zelená časť nad paketovým multiplexom).

Posledná komponenta, ktorá sa nachádza na ceste smerujúcej do softwaru, je takzvaný paketový zahadzovač (fialový multiplexor v dolnej časti). Jeho úlohou je zahadzovať pakety ktoré nechceme aby sa dostali až do softwaru. Jedná sa o obdobu paketového multiplexoru, popísaného v predošlom texte, ktorý má však iba jedno vstupné rozhranie. O tom či bude paket zahodený alebo prepustený na výstup, opätovne rozhoduje jadro P4 prostredníctvom dopredu vybranej položky z metadat.

Dátová cesta určená pre výstupné rozhranie akceleračnej karty, začína spracovaním v jadre P4 (obrázok 5.6). Tu sa postupne nachádza parser paketov, module pre výmenu VLAN, modul pre kontrolu TTL, MAC editor, odbočovací filter a prefixový filter tak ako boli popísané v kapitole 5.1. Ďalej sa tu môžu nachádzať aj ďalšia požadovaná funkciona-




Obr. 5.6: Štruktúra jadra P4.

lita, ktorú si už doplní alebo pozmení samotný užívateľ v jazyku P4. Výstupom P4 jadra sú spracované pakety na dátovej ceste (červená šípka obrázku 5.5) a metadata k daným paketom (modrá šípka), tak ako to bolo popísané v kapitole 4.3.

Na základe dopredu vybraných položiek z metadat, pokračuje vyhodnocovanie v blokovacom filtri (horná, prostredná, oranžová časť obrázku 5.5), ktorý je popísaný v kapitole 5.1.9. Filter, na základe nakonfigurovaných pravidiel zo softwaru, zistí či má byť paket zahodení. Ak filter rozhodne o zahodení paketu, tak je samotný paket a jeho veľkosť (v bajtoch) započíta na príslušné miesto v štatistickej jednotke (oranžový blok na pravej strane). Tieto údaje sú následne poskytnuté softwarovej časti zariadenia.

Ako posledné sa na dátovej ceste, pre výstup z akceleračnej karty, uskutočňuje zahadzovanie paketov (horný, fialový multiplexor) obdobne ako je tomu pri softwarovej ceste v predošlom texte. Tentokrát sa o zahodení paketu nerozhoduje iba na základe vybranej položky z metadat, ale berie sa do úvahy aj výstup z blokovacieho filtru (vykoná sa logický OR).

Posledná komponenta zapojená v architektúre je watchdog (ružová časť obrázku), popísaný v kapitole 5.1.10. V prípade poruchy nastaví komponenta špeciálny, jedno bitový signál, ktorý poskytuje platforma NetCOPE. Nastavenie tohoto signálu spôsobí že sa na výstupných sieťových rozhraniach začne miesto posielania paketov signalizovať chybový stav a na samotnej akceleračnej karte sa rozsvietia príslušné LED diódy.

# Kapitola 6

# Implementácia

Kapitola sa zaoberá postupom akým boli jednotlivé časti navrhnutého zariadenia realizované a implementované. Prvá časť je venovaná implementácií navrhnutého firmwaru. Druhá časť sa venuje implementácii softwarového rozhrania pre komunikáciu a konfiguráciu firmwaru.

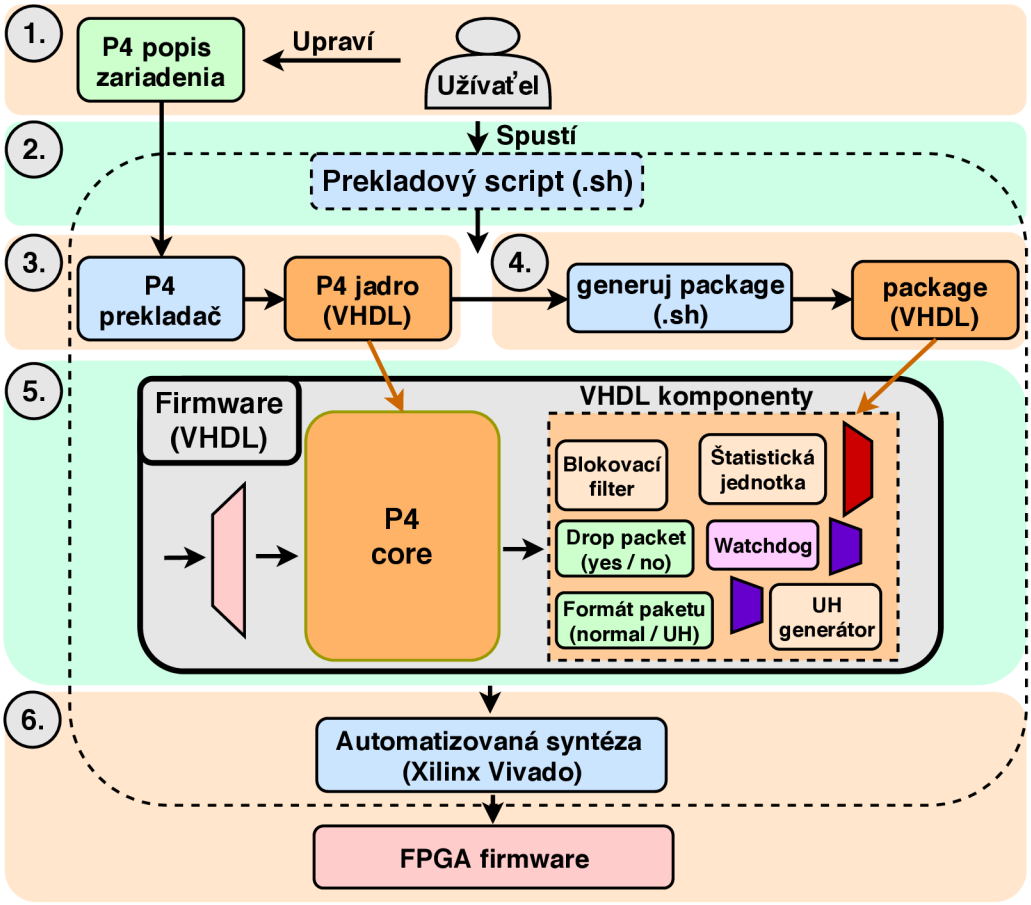
## 6.1 Firmware

Pri tvorbe zariadenia bol kladený dôraz na čo najväčšiu automatizáciu celého procesu prekladu a generovania výsledného fimrwaru (popísaného v kapitole 5). Cieľom bolo dosiahnuť to že akákoľvek zmena funkcionality, ktorú by užívateľ potreboval pridať do firmwaru, by bola realizovaná iba zmenou programu v jazyku P4. Všetky ostatné kroky, od prekladu jazyka P4 až po vygenerovanie výsledného firmwaru, sú plne automatizované. Celý tento proces je vyobrazený na obrázku 6.1.

Prvým krokom implementácie bola realizácia popisu zariadenia v jazyku P4, podla vytvoreného návrhu, popísaného kapitole 5.1 (časť obrázku označená ako 1.). Nad takto vytvoreným popisom sa následné volá prekladač, popísaný v kapitole 4.3. Výsledkom tohto prekladača je VHDL komponenta (P4 jadro), ktorá realizuje popísanú funkcionalitu v jazyku P4 (časť obrázku označená ako 3.).

Problém tohto P4 jadra je že počet ani typ všetkých signálov, na výstupe rozhrania pre metadata, nieje statický a môže sa meniť podla zvoleného popisu v jazyku P4. Aby tieto metadata mohli byť automatizovane prepojené s ďalšími časťami fimwaru, je potrebné toto rozhranie previesť na jednotné dátové typy. Konkrétne sa jedná o štruktúry record v jazyku VHDL, ktoré sú veľmi podobné dátovým štruktúram v jazyku C. Ďalším krokom tak bola implementácia skriptu (v jazyku Shell script), ktorý toto rozhranie prevedie na nami požadované VHDL štruktúry (časť obrázku označená ako 4.). Skript ďalej generuje pomocné funkcie, ktoré realizujú serializáciu a deserializáciu vygenerovaných štruktúr. Tieto funkcie následne uľahčujú pripojenie týchto štruktúr ku komponentám ako sú blokové pamäte, FIFO fronty a podobne. Vygenerované štruktúry a funkcie sú poskytované ostatným častiam firmwaru vo forme knižnice, balíčku VHDL.

Následne bolo potrebné vytvoriť a implementovať celú architektúru zariadenia tak ako bola navrhnutá v kapitole 5.2. Prvým krokom bola implementácia komponet VHDL ako sú blokovací filter, štatistická jednotka, generátor UH, watchdog, paketový multiplexor a demultiplexor a mnoho ďalších pomocných komponent (pravá strana obrázku v časti označenej ako 5.). Počas implementácie boli všetky časti a komponenty testované prostredníctvom




Obr. 6.1: Priebeh prekladu od jazyka P4 až po firmware FPGA.

simulácií v prostredí Modelsim. Na pripojenie k VHDL jadru P4, využívajú všetky komponenty generované dátové štruktúry a funkcie, popísané predošlom texte. Pri implementácii firmwaru a komponent VHDl, boli využité základné generické komponenty poskytované platformou NetCOPE ako sú pamäte FIFO, aritmetické jednotky ALU, komponenty slúžiace na komunikáciu s jednotlivými rozhraniami platformy NetCOPE a ďalšie komponenty umožňujúce prepojovanie jednotlivých logických celkov (modulov). Všetky implementované moduly majú jednotnú štruktúru, ktorá je zložená z entity názov_modulu_ENT.vhd (rozhranie modulu) a architektúry názov_modulu_ARCH.vhd, ktorá obsahuje implementáciu danej funkcionality.

Po implementovaní všetkých potrebných komponent bol vytvorený výsledný firmware celého zariadenia (časť obrázku označená ako 5.), ktorý je zapojený do platformy NetCOPE ako aplikačné jadro. Celý firmware následne prechádza plne automatizovanou syntézou (časť obrázku označená ako 6.), ktorej výsledkom je konfigurační súbor pre obvod FPGA dostupný na akceleračných kartách NFB.

Výsledkom celého procesu implementácie firmwaru je že užívateľ je schopný jeho chovanie meniť prostredníctvom vysoko úrovňového jazyka P4 bez toho, aby potreboval znalosti z oblasti návrhu digitálnych obvodov a fungovania danej platformy. Po zmene chovania v jazyku P4 užívateľ spustí automatizovaný prekladový skript (časť obrázku označená ako 2.), ktorého výsledkom je firmware pre obvod FPGA, ktorý realizuje požadovanú funkcionalitu.

## 6.2 Softwarové rozhranie

Softvérové rozhranie je určené na konfiguráciu a získavanie informácii o aktuálnom stave firmwaru. Rozhranie je rozdelené na dve časti, rozhranie pre komunikáciu s jadrom P4 a rozhranie pre komunikáciu s ostatnými komponentami nachádzajúcich sa vo firmwaru.

Spolu s prekladačom jazyka P4 je pre komunikáciu s jadrom P4 poskytovaná knižnica formou jazyka C. V tomto prípade tak nieje potrebné nič implementovať a užívateľ môže pristupovať k jednotlivým častiam jadra P4 prostredníctvom tohto rozhrania. Štruktúra jadra P4 je priamo uložená a zakódovaná vo firmwaru. Toto softwarové rozhranie je schopné si túto štruktúru načítať a tým sa dynamicky prispôsobovať rôznym konfiguráciám jadra P4. Užívateľ si tak vie predne zistiť aké a koľko Match+Action tabuliek je v danom firmwaru, akého dátového typu sú jednotlivé položky a podobne.

Pre ostatné komponenty bolo prevzaté softwérové rozhranie ktoré už vzniklo v rámci bakalárskej práce [18]. Jedná sa sadu knižníc jazyka C a taktiež softvérových nástrojov, pomocou ktorých je možné konfigurovať firmvér z príkazovej riadky. Toto rozhranie bolo rozšírené a upravené tak aby vyhovovalo novému firmwaru.

K realizácii samotného prenosu dát medzi softvérovým rozhraním a firmvérom je využité konfiguračné a riadiace rozhranie poskytované platformou NetCOPE označované taktiež ako MI32. Prostredníctvom rozhrania MI32 je možné prenášať 32 bitové slová medzi pamäťou softvéru a stavovými registrami (prípadne pamäťami) firmwaru. Rozhranie je rozdelené na jednotlivé moduly (obdobne ako firmware), ktoré sú určené na konfiguráciu jednotlivých modulov ako sú blokovací filter, štatistická jednotka, watchdog a celého jadra P4.

# Kapitola 7

# Dosiahnuté výsledky

Kapitola popisuje výsledky dosiahnuté pri realizácii navrhnutého firmwaru v jednotlivých častiach vývoja až po dosiahnuté výsledky získané laboratórnym testovaním na reálnom hardvéry. Prvá časť je venovaná výsledkom dosiahnutým syntézou implementovaného firmwaru, ktorý prevádza popis navrhnutého firmvéru z VHDL na prvky cieľovej technológie. Ďalej sa kapitola venuje dosiahnutým výsledkom z procesu implementácie celého firmwaru (firmware + platforma NetCOPE), kedy dochádza k mapovaniu prvkov obvodu na konkrétnu cieľovú architektúru (čip FPGA). Kapitola ďalej popisuje akým spôsobom bola testovaná funkcionalita jednotlivých častí vytvoreného firmwaru. Nakoniec sa kapitola venuje výsledkom merania dátovej priepustnosti celého systému, ktoré boli získané počas laboratórneho testovania na reálnom hardvéry v laboratórnych podmienkach.

## 7.1 Výsledky syntézy firmwaru

Syntéza je proces, pri ktorom dochádza k prevodu popísaného obvodu v jazyku VHDL na ekvivalentné vyjadrenie obvodu prostredníctvom obecných logických prvkov. Výsledkom syntézy je výsledný obvod zložený z logických prvkov cieľovej technológie nazývaný taktiež ako netlist. Výsledkom je taktiež odhad spotreby využitých zdrojov na cieľový čip FPGA. Syntéza bola realizovaná softvérovým nástrojom Vivado vo verzii 2018.2 pre čip FPGA Virtex UltraScale+, nachádzajúci sa na akceleračnej karte rodiny NFB-200G2QL.

Výsledky odhadu spotrebovaných zdrojov sú znázornené v tabuľke 7.1. Tabuľka na ľavej strane obsahuje požadované zdroje pôvodného zariadenia popísaného v bakalárskej práci [18]. Na pravej strane je odhad potrebných zdrojov nového firmwaru, ktorý využíva jazyk P4. Znázornená je spotreba základných elementov ako sú LUT, registre a spotreba blokových pamätí BRAM. V zátvorkách je ďalej udávaná percentuálna spotreba jednotlivých zdrojov na čipe FPGA.

| | **Pôvodný firmware** | **Nový fimrware s P4 jadrom** | **Nárast** |
|---|---|---|---|
| LUT | 35567 (5%) | 79302 (10%) | 122% |
| Registre | 50079 (3%) | 108771 (7%) | 117% |
| BRAM | 97 (7%) | 62 (5%) | -37% |

Tabuľka 7.1: Využité zdroje čipu FPGA po syntéze obvodu.

Z výsledkov je vidieť že nový firmware je viac než dvoj násobne väčší pre LUT a registre. V prípade blokových pamätí BRAM je spotreba o čosi menšia. To je spôsobené tým že značná časť funkcionality firmwaru, ktorá sa nachádza v jadre P4, je generovaná. Samotné jadro P4 zaberá väčšinu týchto zdrojov. Naproti tomu celý firmware pôvodného zariadenia bol vytvorený ručne a bolo tak vykonaných množstvo optimalizácií obvodu pre každú komponentu. Tento výsledok sme však očakávali. Naším cieľom bolo docieliť výrazného zvýšenia flexibility celého zariadenia aj za cenu zväčšenia celého obvodu.

## 7.2 Výsledky implementácie firmwaru

Implementácia je proces, pri ktorom dochádza k mapovaniu logických celkov výsledného obvodu z procesu syntézy na jednotlivé dostupné prvky cieľovej architektúry, k ich presnému rozmiestneniu a taktiež k ich vhodnému prepojeniu na čipe FPGA. Do tohto procesu taktiež vstupujú podmienky špecifikované užívateľom ako sú perióda hodinového signálu alebo pridelenie portov z popisu entity na konkrétne pini čipu FPGA. Výsledkom tohto procesu sú informácie o spotrebovaných zdrojoch čipu FPGA, informácia či sa podarilo dosiahnuť požadovanú frekvenciu a bitstream, prostredníctvom ktorého je následne konfigurovaný čip FPGA.

K procesu implementácie bol taktiež použitý softvérový nástroj Vivado vo verzii 2018.2. Implementácia bola uskutočnená vrátane použitej platformy NetCOPE. Porovnanie dosiahnutých výsledkov pôvodného a nového firmwaru sú znázornené v tabuľke 7.2. Z výsledkov

| | **Pôvodný firmware** | **Nový fimrware s P4 jadrom** | **Nárast** |
|---|---|---|---|
| LUT | 125747 (14%) | 223186 (27%) | 77% |
| Registre | 141075 (9%) | 171065 (11%) | 21% |
| BRAM | 209 (15%) | 198 (14%) | -5% |

Tabuľka 7.2: Využité zdroje čipu FPGA po implementácií obvodu.

môžeme znova pozorovať že nový firmware je o niečo väčší, čo je zapríčinené tým že oproti pôvodnému firmwaru, je značná časť obvodu generovaná automatizovanými nástrojmi.

Jedným z najťažších úloh pri návrhu obvodu pre čip FPGA je práve dosiahnutie splnenia požiadavku pre taktovaciu frekvenciu. Naše kritérium bolo stanovené na hodnotu 200 MHz, čo zodpovedná dátovej priepustnosti 100 Gbps. Riešením tohto problému je takzvané zmenšovanie kritickej cesty. Kritická cesta je časť obvodu, kde dochádza k najdlhšiemu oneskorovaniu signálnu kvôli veľkému množstvu, za sobov idúcich, prvkov kombinačnej logiky. Jednou z techník odstraňovania týchto kritických ciest je vkladanie registrov do tejto cesty, čím dôjde k jej rozdeleniu a tým jej skráteniu. Vkladanie registrov do obvodu spôsobuje zvyšovanie latencie, čo v niektorých častiach obvodu, kde je závislosť na tejto latencii (kvôli využívaniu spätnej väzby), zapríčiní nefunkčnosť a je tak potrebné danú časť obvodu vhodne upraviť prípadne prepracovať.

Pri odstraňovaní kritickej cesty sa podarilo dosiahnuť stanovenej frekvencie 200 MHz a tým tak dosiahnuť teoretickú dátovú priepustnosť 100 Gbps.

## 7.3 Funkcionálne testovanie

V laboratórnom prostrední boli prostredníctvom simulačného prostredia Modelsim, hardvérového testeru Spirent TestCenter [11] a hadvérového akcelerátoru NFB-200G2QL [15] realizované testy na overenie správnej funkcionality jednotlivých častí vytvoreného firmvéru.

Napríklad testovanie jednotky na výmenu MAC adries, popísanej v kapitole 5.1.4, prebiehalo nasledovne. Bolo overované správne fungovanie nahrádzania cieľových a zdrojových MAC adries v paketoch. Taktiež bola testovaná funkcionalita vzájomnej výmeny týchto adries. Ako prvé bola overená funkcionalita komponenty pomocou simulácie v simulačnom prostredí Modelsim. Následne bola komponent testovaná priamo na hardvérom akcelerátore. Pri tomto teste boli na pripojené zariadenie (akceleračnú kartu obsahujúci vytvorený firmware) postupne posielané náhodne vygenerované sieťové pakety hardvérovým testerom. Následne boli na tomto hardvérovom testeri sledované prijímané pakety z pripojenej akceleračnej karty. U týchto paketov bola následne uskutočnená kontrola MAC adries, či testovaný firmvér správne vykonal úpravu paketov podľa zvolenej konfigurácie. Konkrétne boli testované nasledovné konfigurácie: keď nemala byť vykonaná žiadna modifikácia MAC adresy, výmena zdrojovej MAC adresy, výmena cieľovej MAC adresy, výmena oboch MAC adries a vzájomná výmena cieľovej a zdrojovej MAC adresy.

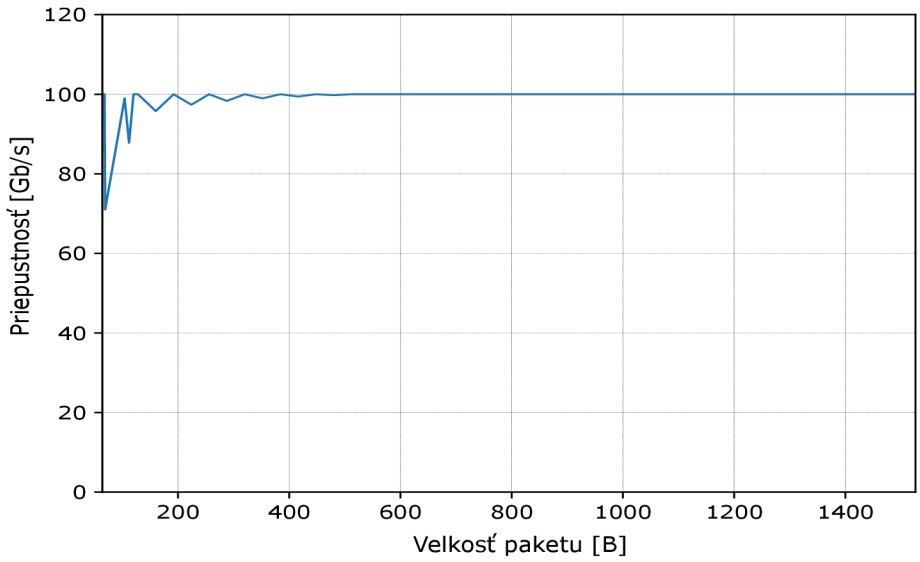
Funkcionalita ostatných častí zariadenia, popísaných v kapitole 5.1 prebiehala rovnakým alebo veľmi podobným spôsobom ako v prípade jednotky na výmenu MAC adries. Týmto testami tak bolo úspešne overené správne fungovanie vytvoreného zariadenia.

## 7.4 Dátová priepustnosť

Meranie priepustnosti vytvoreného firmvéru bolo uskutočnené v laboratórnych podmienkach. Testovanie prebehlo na hardvérovej akceleračnej karte NFB-200G2QL vo verzii jedného, 100G, ethernetového portu. Generovanie testovacích dát bolo realizované hardvérovým generátorom Spirent TestCenter. Pre účely tohto testovania boli ethernetové rámce generované so záhlavím protokolov IPv4 aj IPv6 s náhodnými hodnotami a s náhodnými vygenerovanými dátami bez transportného protokolu. Jadro firmvéru bolo nastavené tak, aby prijímané sieťové pakety prechádzali spracovaním cez všetky vyhodnocovacie a riadiace moduly, z ktorých je jadro firmvéru tvorené.

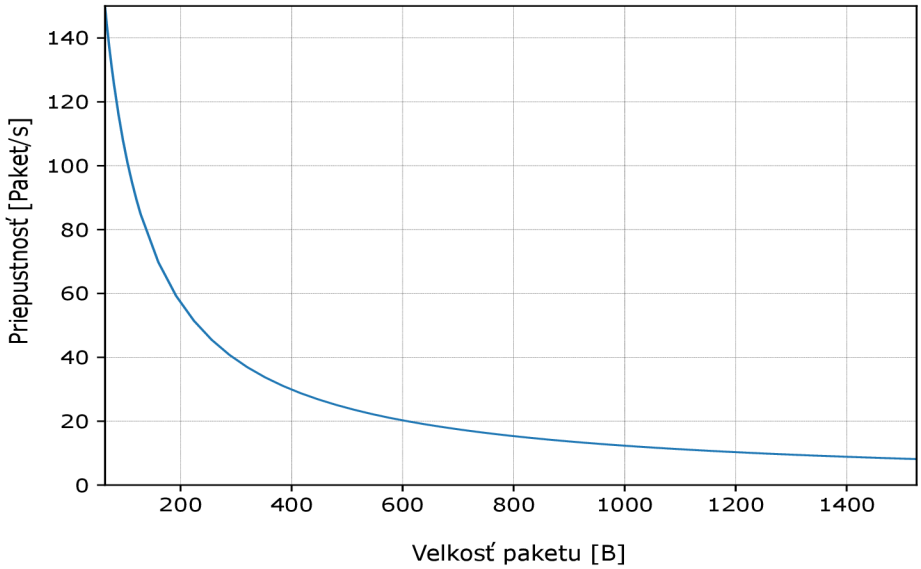
Meranie prebiehalo takým spôsobom že na vstupné ethernetové rozhranie boli posielané ethernetové rámce v takom množstve, aby bola zaručená plná saturácia pripojenej 100G linky. Následne boli v pravidelných intervaloch vyčítané čítače rámcov zo vstupných sieťových blokov firmvéru, ktoré sú poskytované v rámci použitej platformy NetCOPE. Výsledná dátová priepustnosť bola odvodená na základe pomeru medzi počtom všetkých prijatých a spracovaných rámcov firmvérom, a počtom všetkých prichádzajúcich rámcov na pripojené rozhranie. Meranie bolo vykonané pre rôzne dĺžky ethernetových rámcov v rozsahu 64–1526 bajtov. Výsledky merania sú znázornené na grafoch 7.1 a 7.2.

Graf 7.1 znázorňuje dátovú priepustnosť v Gbps v závislosti na dĺžke prijímaných ethernetových rámcoch. Na grafe sú znázornené výsledky kde sieťové pakety prechádzajú spracovaním cez všetky komponenty firmwaru a taktiež dochádza k prenosu dát do operačnej pamäte prostredníctvom DMA. Z grafu vidíme že požadovanú dátovú priepustnosť dosahujeme až od veľkosti paketov 500 bajtov. Pre menšie rámce je priepustnosť menšia ako 100 Gbps. Toto zníženie priepustnosti je zapríčinené jadrom P4, ktoré je generované pomocou prekladača jazyka P4 a pre stanovenú frekvenciu obvodu 200 MHz nedosahuje




Obr. 7.1: Meranie dátovej priepustnosti zariadenia v jednotkách Gbps.

priepustnosť 100 Gbps. Toto obmedzenie by malo byť eliminované v novšej verzií prekladača, ktorý bude podporovať štandard jazyka $P4_{16}$.




Obr. 7.2: Meranie dátovej priepustnosti zariadenia v paketoch za sekundu.

Na grafe 7.2 je znázornený ďalší pohlaď na dosiahnuté výsledky z testovania dátovej priepustnosti. Tento graf znázorňuje dátovú priepustnosť v počte prenesených paketov za jednu sekundu pri ethernetových rámcoch v rozsahu 64-1526 bajtov. Krivka znázorňuje datovú priepustnosť prepočítanú na počet prenesených rámcov danej dĺžky za jednu sekundu.

## 7.5 Rozšíriteľnosť zariadenia

Hlavným cieľom celej práce bolo docieliť výrazné zvýšenie flexibility celého zariadenia a tým tak značne zmenšiť úsilne a skrátiť čas, ktoré sú potrebné pre vývoj nových funkcionalít zariadenia. Tento ciel sme dosiahli integrovaním vysoko úrovňového jazyka P4 do procesu rozširovania funkcionality firmwaru zariadenia. Flexibilitu zariadenia môžeme demonštrovať na niekoľkých príkladoch.

Ako prvé uvažujme tu istú modelovú situáciu ako v kapitole 3.4.1, kde chceme rozšíriť prefixový filter o možnosť filtrovania aj na základe hodnoty ID z protokolu VLAN. V pôvodnom zariadení, kde sem celú túto úpravu museli robiť ručným zásahom do firmwaru, nám tento proces trval veľmi dlho a s vynaložením veľkého úsilia. V novom firmwaru je prefixový filter realizovaný pomocou jazyka P4 (kapitola 5.1.9). Pridanie tejto funkcionality tak bude znamenať iba úpravu príslušného P4 zdrojového kódu nasledovne:

```
// Prefixový filter
table table_ipv4_filter {
    reads {
        ipv4.dstAddr : lpm;
        // Pridanie novej položky pre filtráciu
        vlan.id      : exact;
    }
    actions {
        repalce_mac_and_send_sw;
        NoOp;
    }
    max_size: SIZE;
}
```

Výpis 7.1: Rozšírenie prefixového filtru v jazyku P4.

Jediná úprava ktorú sme potrebovali vykonať bolo pridanie novej položky *vlan.id* do sekcie *reads*. Následne už stačí zavolať plne automatizovaný skript, ktorý sa postará o vytvorenie nového firmwaru (kapitola 6.1).

Ako druhý príklad si môžeme uviesť pridanie novej položky do hlavičky UH pomocou UH generátoru (kapitola 5.1.5). V pôvodnom firmwaru zariadenia by sme si opätovne museli prejesť celým cyklusom vývoja od získania potrebnej hodnoty z paketu, cez testovanie dopadu našej úpravy na funkčnosť a výkonnosť firmwaru, až po úpravu softvérového rozhrania. V novom firmwaru máme tento generátor čiatočne realizovaný v jazyku P4. Pridanie novej položky do hlavičky UH by tak vyzeralo nasledovne:

```
header_type metadata_t {
    fields {
        ...
        IPv4_src : 4;
        // Pridanie novej položky do štruktúry
        vlan_id : 3;
        ...
    }
}
// Metadata pomenované ako metadata, budú dostupné na výstupe P4 jadra
metadata metadata_t metadata;
// Akcia na nahranie hodnôt do metadat
action set_metadata (){
    ...
    modify_field(metadata.IPv4_src, ipv4.src)
    // nakopírovanie požadovanej hodnoty
```

```
17      modify_field(metadata.vlan_id, vlan.id)
18      ...
19  }
```

Výpis 7.2: Rozšírenie metadát o vlastnú položku v jazyku P4.

Postačí nám rozšíriť štruktúru *metadata* o požadovanú položku a následne ešte zabezpečiť nastavenie požadovanej hodnoty v akcii *set_metadata*. Automatizovaný prekladový skript sa opätovne postará o vytvorenie nového firmwaru.

Nielenže sme schopný ľahko upravovať funkcionalitu firmwaru ale sme schopný pridávať aj úplne novú. Napríklad pridanie nového filtru by vyžadovalo iba vytvorenie podobných konštrukcií v jazyku P4 z ktorých je tvorený prefixový filter.

Výsledkom celej práce je tak výkonné a flexibilné zariadenie, ktorého funkcionalitu môže meniť aj užívateľ, ktorý nemá znalosti z oblasti vývoja digitálnych obvodov. Plne bude dostačovať znalosť jazyka P4 a prípadných obmedzení prekladových nástrojov tohto jazyka na danú platformu.

# Kapitola 8

# Záver

Cieľom diplomovej práce bolo vytvoriť flexibilné a hardvérovo akcelerované zariadenie pre ochranu pred (D)DoS útokmi s využitím technológie FPGA a vysoko úrovňového jazyka P4. Pri návrhu a následnej realizácii firmwaru bol kladený dôraz na dosiahnutie čo možno najväčšej možnej flexibility zariadenia pomocou jazyka P4, na dosiahnutie požadovanej frekvencie obvodu 200 MHz a taktiež na splnenie všetkých špecifikácií, ktoré vyplynuli na základe analýzy danej problematiky. Súčasťou implementácia firmwaru bola tiež realizácia softvérového a konfiguračného rozhrania, ktoré umožňuje konfiguráciu vytvoreného firmwaru.

V rámci riešenia diplomovej práce som sa najprv musel dôkladne oboznámiť s problematiku bezpečnosti počítačových sietí so zameraním na amplifikačné a volumetrické útoky typu (D)DoS. Ďalej som sa podrobne oboznámil s možnosťami a vlastnosťami jazyka P4 a jeho následným využitím pri návrhu zariadenia. Následne som sa venoval technológii programovateľných hradiel FPGA s záväznosťou na hardvérové akcelerátory rodiny COMBO a NFB určené pre spracovanie sieťových dát, ktoré túto technológiu používajú. Práca sa ďalej venovala rozborom aktuálne už existujúceho zariadenia na ochranu pred (D)DoS, ktoré vzniklo v rámci mojej bakalárskej práce [18]. Konkrétne som sa zameral na nedostatky tohto riešenia a zisťovaniu toho prečo toto zariadenie už nevyhovuje súčasnom podmienkam a nárokom, ktoré sú naň kladené. Následne som sa venoval podrobnému rozboru dostupných štandardov jazyka P4 a rozboru vlastností dostupných prekladových nástrojov tohoto jazyka pre technológiu FPGA.

Po získaní všetkých potrebných znalostí som sa venoval tomu akým spôsobom by bolo alebo nebolo možné popísať jednotlivé časti navrhovaného zariadenia v jazyku P4. Ďalej som pristúpil k návrhu celého firmwaru zariadenia. Na základe vytvoreného návrhu som implementoval celý firmware, ktorý som následne integroval pomocou platformy NetCOPE na príslušný hardvérový akcelerátor.

Správnu funkcionalitu vytvoreného firmvaru som následne overil prostredníctvom laboratórneho testovania na akceleračnej karte NFB-200G2QL s využitím hardvérového testovacieho zariadenia Spirent TestCenter, ktorý umožňuje generovanie a monitorovanie sieťového prenosu pri rýchlostiach 100 Gbps. V tomto laboratórnom prostredí som taktiež otestoval a overil reálnu priepustnosť celého systému na rýchlosti 100 Gbps.

Výsledkom celej práce je firmware pre akcelerované zariadenie na ochranu pre (D)DoS útokmi, ktorého funkcionalitu je možné meniť prostredníctvom vysoko úrovňového jazyka P4. Vďaka tomuto jazyku môže meniť funkcionalitu firmwaru aj užívateľ, ktorý nedisponuje znalosťami z oblasti návrhu digitálnych obvodov pre technológiu FPGA.

Táto práca bude ďalej pokračovať v rámci vedecko výskumného projektu pre Ministerstvo Vnútra Českej Republiky pod názvom Adaptívna ochrana pred DDoS útokmi. Mojím cieľom, ako riešiteľa tohoto projektu, bude ďalší rozvoj celého zariadenia a nasadenie vytvoreného zariadenia v reálnych sieťových podmienkach. Celé zariadenie pre ochranu pred (D)DoS útokmi bude taktiež nasadené na hlavnej linke akademickej siete CESNET, ktorá v súčastnosti pracuje na prenosových rýchlostiach 100 Gbps.

# Literatúra


[1] DISTRIBUTED DENIAL OF SERVICE ATTACK (DDOS) DEFINITION. Imperva Incapsula: DDoS Knowledge Center – Latest Threats and Mitigation Methods, [Online; navštívené 21-02-2019].
URL https://www.incapsula.com/ddos/ddos-attacks.html

[2] DNS Amplification | DDoS Attack Glossary. Imperva Incapsula: DDoS Knowledge Center – Latest Threats and Mitigation Methods, [Online; navštívené 06-02-2019].
URL
https://www.incapsula.com/ddos/attack-glossary/dns-amplification.html

[3] ISTR, internet security thread report. Symantec Corporation: 2018 Internet Security Threat Report, [Online; navštívené 06-02-2019].
URL https://www.symantec.com/security-center/threat-report

[4] NO SOONER DID THE INK DRY: 1.7TBPS DDOS ATTACK MAKES HISTORY. NETSCOUT Systems, Inc., [Online; navštívené 06-02-2019].
URL
https://www.netscout.com/blog/security-17tbps-ddos-attack-makes-history

[5] NTP Amplification | DDoS Attack Glossary. Imperva Incapsula: DDoS Knowledge Center – Latest Threats and Mitigation Methods, [Online; navštívené 06-02-2019].
URL
https://www.incapsula.com/ddos/attack-glossary/ntp-amplification.html

[6] SNMP REFLECTION/AMPLIFICATION | DDoS Attack Glossary. Imperva Incapsula: DDoS Knowledge Center – Latest Threats and Mitigation Methods, [Online; navštívené. 06-02-2019].
URL https://www.incapsula.com/ddos/attack-glossary/snmp-reflection.html

[7] DDoS Amplification Attacks. Noction Network Intelligence, 2018, [Online; navštívené 20-03-2019].
URL https://www.noction.com/blog/ddos-amplification-attacks

[8] Benáček, P.: *Generation of High-Speed Network Device from High-Level Description.* Dizertační práce, České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.
URL https://fit.cvut.cz/sites/default/files/PhDThesis-Benacek.pdf

[9] CESNET: Cards. Liberouter / CESNET TMC group, [Online; navštívené. 12-02-2019].
URL https://www.liberouter.org

[10] CESNET: NetCOPE. Liberouter / CESNET TMC group, [Online; navštívené. 09-03-2019].
URL https://www.liberouter.org/technologies/netcope/

[11] Communications, S.: Spirent TestCenter Appliances. [Online; navštívené. 28-04-2019].
URL https://www.spirent.com/products/testcenter/platforms/appliances

[12] Douligeris, C.; Mitrokotsa, A.: DDoS attacks and defense mechanisms. In *Proceedings of the 3rd IEEE International Symposium on Signal Processing and Information Technology (IEEE Cat. No.03EX795)*, IEEE, 2004, ISBN 0-7803-8292-7, s. 190–193, doi:10.1109/ISSPIT.2003.1341092.
URL http://ieeexplore.ieee.org/document/1341092/

[13] Juniper Networks: Understanding SYN Flood Attacks. JUNOS Software Security Configuration Guide, [online]. [cit. 08-03-2019].
URL http://www.juniper.net/techpubs/software/junos-es/junos-es93/junos-es-swconfig-security/understanding-syn-flood-attacks.html#id-3412834128

[14] Juniper Networks: Understanding UDP Flood Attacks. JUNOS Software Security Configuration Guide, [online]. [cit. 08-03-2019].
URL http://www.juniper.net/techpubs/software/junos-es/junos-es93/junos-es-swconfig-security/understanding-udp-flood-attacks.html#id-60351

[15] Kekely, L.; Špinler, M.; Štepán Friedl; aj.: Demonstration of Full-Duplex Packet Transfers over PCI Express with Sustained 200 Gbps Throughput. 2018.
URL https://www.liberouter.org/wp-content/uploads/2018/12/3optFA5vcJq2MkBSpUYVft.pdf

[16] Kořenek, J.: *Rychlé vyhledávání regulárních výrazů s využitím technologie FPGA*. Dizertačná práca, Vysoké učení technické v Brně, Fakulta informačních technologií, 2010.
URL http://www.fit.vutbr.cz/study/DP/PD.php?id=162

[17] Kuka, M.: Hardvérovo akcelerované zariadenie pre ochranu pred (D)DoS útokmi. Excel@FIT: Studentská Konference Inovací, Technologií a Vědy v IT, [Online; navštívené. 7-1-2019].
URL http://excel.fit.vutbr.cz/submissions/2017/032/32.pdf

[18] Kuka, M.: *Hardwarově akcelerované zařízení pro ochranu před DoS útoky*. Bakalářská práce, Vysoké učení technické v Brně, Fakulta informačních technologií, 2017.
URL http://www.fit.vutbr.cz/study/DP/BP.php?id=19924

[19] Liberouter, CESNET z.s.p.o.: P4 language. [Online; navštívené. 10-03-2019].
URL https://www.liberouter.org/technologies/p4/

[20] Mallikarjunan, K. N.; Muthupriya, K.; Shalinie, S. M.: A survey of distributed denial of service attack. IEEE, 2016, ISBN 978-1-4673-7807-9, doi:10.1109/ISCO.2016.7727096.
URL https://ieeexplore.ieee.org/document/7727096/

[21] Minařík, P.: Metody ochrany před útoky typu DDoS. 2015, [Online; navštívené 06-03-2019].

URL https://www.systemonline.cz/it-security/metody-ochrany-pred-utoky-typu-ddos.htm

[22] Netronome Systems, Inc.: Programming NFP with P4 and C. [Online; navštívené. 10-01-2019].
URL https://www.netronome.com/media/redactor_files/WP_Programming_with_P4_and_C.pdf

[23] Networks, B.: TOFINO, World's fastest P4-programmable Ethernet switch ASICs. [Online; navštívené. 20-04-2019].
URL https://barefootnetworks.com/products/brief-tofino/

[24] P4 Language Consortion: P4 Language Tutorial. Presentation, p4.org, [Online; navštívené. 20-04-2019].
URL https://p4.org/assets/P4_tutorial_01_basics.gslide.pdf

[25] P4 Language Consortium: P4-BEHAVIORAL. [Online; navštívené. 10-01-2019].
URL https://github.com/p4lang/p4c-behavioral

[26] P4 Language Consortium: P4-C. [Online; navštívené. 10-01-2019].
URL https://github.com/p4lang/p4c

[27] P4 Language Consortium: P4-HLIR. [Online; navštívené. 10-1-2019].
URL https://github.com/p4lang/p4-hlir

[28] P4 Language Consortium: P4-VHDL. [Online; navštívené. 10-01-2019].
URL http://p4fpga.github.io/

[29] P4 Language Consortium: The $P4_{14}$ Language Specification. [Online; navštívené. 05-01-2019].
URL https://p4.org/p4-spec/p4-14/v1.0.5/tex/p4.pdf

[30] P4 Language Consortium: The $P4_{16}$ Language Specification. [Online; navštívené. 05-01-2019].
URL https://p4.org/p4-spec/docs/P4-16-v1.1.0-spec.pdf

[31] Puš, V.; Kekely, L.; Kořenek, J.: Design Methodology of Configurable High Performance Packet Parser for FPGA. IEEE, 2014, ISBN 978-1-4799-4558-0, doi:10.1109/DDECS.2014.6868788.
URL https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6868788

[32] Stephen Ibanez: P4 – NetFPGA. Presentation, github.com, [Online; navštívené. 20-04-2019].
URL https://github.com/NetFPGA/NetFPGA-public/blob/master/2017-apr-summit/Ibanez_P4-NetFPGA.pdf

[33] Xilinx technology company: Ternary Content Addressable Memory(TCAM) Search IP for SDNet, November 2017, [Online; navštívené. 11-1-2019].
URL https://www.xilinx.com/support/documentation/ip_documentation/tcam/pg190-tcam.pdf

[34] Xilinx technology company: UltraScale Architecture and Product Data Sheet: Overview, November 2018, [Online; navštívené. 15-01-2019].
URL https://www.xilinx.com/support/documentation/data_sheets/ds890-ultrascale-overview.pdf