VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

ANALÝZY SÍŤOVÉHO PROVOZU NA PROCESORU NXP A FPGA

NETWORK TRAFFIC ANALYSIS USING NXP PROCESSOR AND FPGA

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. MICHAL ORSÁK**
AUTHOR

VEDOUCÍ PRÁCE **Ing. JAN KOŘENEK, Ph.D.**
SUPERVISOR

BRNO 2018

Abstrakt

Primárním cílem této práce je prozkoumat možnosti síťového procesoru NXP LS2088 společně s technologii FPGA. Sekundárním cílem je na této platformě zprovoznit a optimalizovat existující software pro analýzu aplikačních protokolů. Tento existující software je pevně svázán s FPGA firmwarem jedno-gigabitové platformy. Síťový procesor NXP LS2088 obsahuje celou řadu akcelerátorů a virtuální rekonfigurovatelnou síť. V rámci této práce byly podrobně prozkoumány vlastnosti hardwarových prostředků této platformy. Tyto znalosti byly následně využity pro přeportování existujícího řešení pro L7 analýzu. Portace byla provedena s důrazem na maximální výkon. Tato optimalizace zahrnovala implementaci hardwarové synchronizace vláken, přechod na knihovnu DPDK a další radikální změny. Výsledek této práce je především analýza a odladění jednotlivých subsystému síťového procesoru NXP, softwarový balík pro tuto novou platformu a optimalizace existujícího software.

Abstract

The primary goal of this thesis is to exploit possibilites of aa entirely new hardware based on NXP LS2088 and FPGA. The secondary goal is to create firmware for this processor working out-of-box and perform optimisations of existing software for L7 analysis. This software was deeply bound to a previous hardware platform. The network processor NXP LS2088 contains many hardware accellerators and a virtual reconfigurable network. This thesis exploits all hardware parts of on this platform. Many tweaks and optimizations were performed based on this analysis to achieve maximum efficieny of software for L7 analysis. There were many intensive optimisations like rewriting for the DPDK library and new hardware or hardware synchronization of worker threads of this application. The main result of this thesis is working platform with efficient L7 analysis software which actively uses accelerators in FPGA and NXP network processor. SDK for new platform is also prepared.

Klíčová slova

NXP LS2088, DPAA2, FPGA, siťové filtry, 10G Ethernet, LI, DPDK, ARM, ARM64, Yocto project, vestavěné systémy, hardware-software codesign

Keywords

NXP LS2088, DPAA2, FPGA, network filters, 10G Ethernet, LI, DPDK, ARM, ARM64, Yocto project, embedded system, hardware-software codesign

Citace

ORSÁK, Michal. *Analýzy síťového provozu na procesoru NXP a FPGA*. Brno, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jan Kořenek, Ph.D.

Analýzy síťového provozu na procesoru NXP a FPGA

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Jana Kořeneka, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Michal Orsák
23. května 2018

Poděkování

Poděkování celému týmu projektu Sprobe.

Obsah

Kapitola 1

Úvod

Rychlost síťových spojů roste značně rychleji než výkon výpočetních prvků. Předpokládá se, že rychlost globální IP komunikace naroste pětkrát od roku 2016 do 2021[2]. V době psaní této práce již existují výsledky, které tento trend potvrzují. Narůst výkonosti procesorů za stejné období bude pravděpodobně o řád nižší. V dnešní době už i malí lokální poskytovatelé internetového připojení začínají používat 10GE[1]. Na rychlostech těchto spojů již není možné provádět hloubkovou analýzu provozu s použitím běžně dostupného hardware. Hloubková analýza je úloha kdy se prohledává nejen hlavička paketu, ale i jeho často s použitím informací o protokolu a regulárních výrazů. Jsou tendence do síťových karet integrovat co nejvíce podpory pro protokoly vyšších síťových vrstev. Další trend je zpracovávání síťového provozu bez zásahu jádra operačního systému a velmi populární je taktéž použití technologie *FPGA*. Hloubková analýza aplikačních protokolů je nutná pro řadu systémů zabezpečení a detekcí.

Tato práce prozkoumává možnosti síťového procesoru *NXP LS2088* ve spojení s technologii *FPGA*. Tento síťový procesor obsahuje velké množství akcelerátorů pro zpracování síťového provozu a *virtuální rekonfigurovatelnou síť* vytvořenou s pomocí těchto akcelerátorů. Tato práce se zabývá využitím nové platformy pro existující aplikace pro analýzu protokolů aplikační vrstvy. Tato aplikace je pevně spojena s platformou a její portace zahrnuje vytvoření celého firmware pro nově vzniklou síťovou sondu vybavenou 10GE porty. Aby bylo možné analyzovat provoz v L7 vrstvě na sítích s propustností $10Gb/s$ je nutné využít hardwarovou akceleraci a všechny jádra procesoru. Hardwarová akcelerace je realizována pomocí technologie *FPGA* a akcelerátoru v *SoC* hlavního procesoru NXP.

Firmware FPGA až na výjimky není součástí této práce a je zmíněn jen okrajově v sekci 5.2. Součástí této práce je však komunikace s těmito akcelerátory a jejich využití. V *FPGA* firmware je několik akcelerátorů, které slouží k předfiltrování vstupních síťových toků a jejich distribuci na výpočetní vlákna software pro analýzu L7 síťového provozu. V NXP SoC je pak virtuální síť, které zajišťuje zbytek distribuce na výpočetní vlákna a na procesoru v SoC běží zejména vlastní analyzátor aplikačních protokolů. *FPGA* je nově s hlavním procesorem propojeno pouze prostřednictvím ethernetových portů. Platforma vyžaduje rychlou spolehlivou komunikaci s *FPGA* s nízkou latencí, což TCP po Ethernetu obecně neposkytuje. Chování aplikací muselo být tedy optimalizováno a konfigurační rozhraní muselo být přemostěno.

Přechod na novou platformu, je spojen s komplikacemi na několika úrovních. Softwarový balík dodávaný výrobcem pro tento nový procesor není dosud hotový. To znamená, že řada

[1] 10 Gigabit Ethernet

ovladačů má jen omezenou funkcionalitu nebo zcela chybí a současně nelze jejich vývoj více urychlit. Dále softwarový balík pro starou platformu je postaven na jiném buildovacím systému a má závislosti na velmi starší kompilátor a linuxové jádro. Dalším problémem je, že SoC NXP je do jisté míry prvním svého druhu. Architektonické změny si taktéž vyžádali změnu komunikace s FPGA a zavedení paralelního zpracování.

Portace jako celek je popsán v kapitole 5. Síťový analyzátor *PaSt* je popsán v sekci 3, port této aplikace je popsán v sekci 5.3. Akcelerátory v FPGA jsou popsány v sekcí 5.2 a způsob paralelizace v 5.4. *Virtuální rekonfigurovatelná síť* a akcelerátory procesoru *NXP* je popsána sekci 5.6. Knihovna pro přístup ke vzdáleným hardwarovým prostředkům je popsaná v sekci 3.3. Sekce 5.9 obsahuje dosavadní poznatky o závislostech konfigurace SoC na jeho výkonu v daných aplikacích. V této sekci jsou také uvedeny zásady správné konfigurace, s nimiž je možné dosáhnout maximálního výkonu této platformy.

Kapitola 2

Sprobe

Projekt Sprobe je zaměřen na vývoj sond síťového provozu, které umožňují analyzovat síťový provoz na L7 aplikační vrstvě. Tyto sondy jsou navrženy pro použití u lokálních poskytovatelů internetu *ISP*. Samotná sonda je kompaktní zařízení, které je připojeno na linku prostřednictvím TAP. Tyto sondy tedy nezasahují do síťového provozu a nejsou tedy detekovatelné.

V rámci tohoto projektu v minulosti vznikla platforma pro analýzu síťového provozu na L7 vrstvě. Tato síťová sonda využívá filtr a další akcelerátory v *FPGA* k detekci síťových toků, které obsahují zadanou signaturu. Nalezené toky sonda odesílá na ukládací server. Díky svým akcelerátorům dokáže tato sonda pracovat bez zpomalení i na saturované lince za předpokladu, že zájmový provoz tvoří dostatečně malou část síťového toku, což je pro obvyklé nasazení této sondy typické.

Tato stará platforma je omezená svým výkonem na dva porty o rychlosti $1Gb/s$. Platforma je založena na čipu *SoC Xilinx Zynq*. Tento čip obsahuje *FPGA* a i procesor v jediném pouzdře. Jejich propojení je zajištěno pomocí sběrnic s nízkou latencí a vysokou propustností. Některé z těchto sběrnic jsou také cache koherentní. Tato $1Gb/s$ sonda obsahuje v FPGA jednu zřetězenou linku pro zpracování paketů. Tato zřetězená linka zajišťuje předfiltraci, vyhledávání regulárních výrazů a detekci protokolů. Tento starý firmware FPGA je popsán v sekci 3.1. Výsledný datový tok proudí do jediné instance aplikace PaSt, popsané v sekci 3. Aplikace PaSt je hlavním jádrem celé platformy. Pro DMA přenosy mezi FPGA a software je použit vlastní hardware se specifickými ovladači pro obsluhu.

V rámci tohoto projektu vzniká v době psaní této práce nová verze této sondy. Tento nový model bude vybaven čtyřmi ethernetovými porty o rychlosti $10Gb/s$. Tato nově vzniklá platforma je popsána v sekci 2.

Žádná část staré platformy není přímo použitelná pro nově vznikající hardwarovou platformu a proto je nutné provést požadované změny. Architektura staré a nové platformy je diametrálně odlišná v mnohých směrech. Nová platforma obsahuje *FPGA Intel Arria 10* a síťový procesor *NXP LS2088a*. Firmware *FPGA* je popsán v sekci 5.2. Hardware a software pro tento síťový procesor je popsán v sekci 4. Procesor obsahuje řadu akcelerátorů, které by bylo výhodné využít. Tyto akcelerátory jsou popsány v sekci 4.1 a jejich použití je popsáno v sekci 5.3.

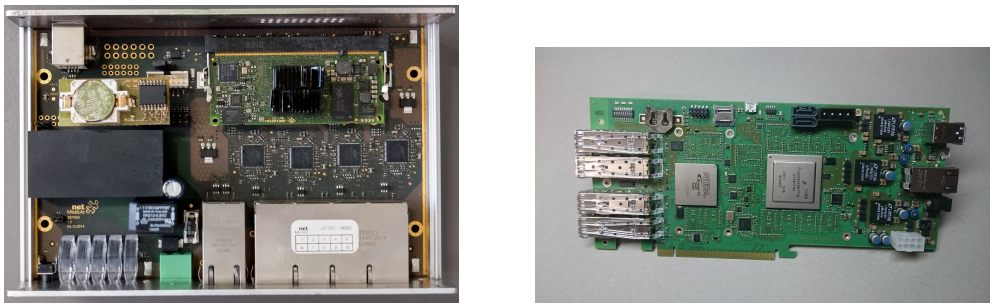
Jádrem staré platformy je aplikace síťového analyzátoru *PaSt*. Pro efektivní využití nové platformy je nutné provést paralelizaci tohoto software. Paralelní implementace této aplikace je spojena se zásahy do *FPGA* firmware i s problémy s atomicitou konfigurace *FPGA*. Způsob paralelizace je popsán v sekci 5.4.

Dalším nutným krokem k dosažení optimálního výkonu na nové platformě je použití knihovny *DPDK* popsané v sekci 3.2. Přechod na tuto knihovnu pro komunikaci se síťovými rozhraními je spojen s kompletním přepsáním řady aplikací i s řadou nedodělků v implementaci této knihovny pro tento procesor. Použitím knihovny *DPDK* a hardwarové synchronizace výpočetních vláken bylo dosaženo velmi výrazného navýšení výkonu, jak je popsáno v sekci 5.10.

Nově vznikl hardware Sprobe10g sondy (obrázek 2.1) s $10Gb/s$ porty postaven na zcela jiných technologiích. *FPGA* je nyní mimo hlavní *SoC* a jeho propojením s hlavním procesorem je zajištěno prostřednictvím ethernetových portů. Procesor *NXP* obsahuje sofistikovanou sběrnici *DPAA2* s akcelerátory pro směrování, filtrování, kompresi a šifrování. Tato sběrnice je popsána v sekci 4.1. Síťové aplikace pro tuto platformu budou postaveny na knihovně *DPDK*. Nový *SoC* nabízí oproti starému masivní paralelní výkon. Firmware *FPGA* je rozšířen pro dosažení požadované propustnosti. Nově přibyly jednotky pro vyvažování zátěže a směrování toků na jednotlivá vlákna aplikací na procesoru NXP.

Nový firmware *FPGA* tedy obsahuje zejména:

- Jednotku pro extrakci hlaviček
- Jednotku pro detekci protokolů.
- Akcelerátor regulárních výrazů založený na Takaguchi NFA.
- Filtr založený na kukaččím heši.
- Jednotky pro směrování toků na konkrétní jádra aplikací běžících na procesoru NXP.



(a) Sprobe1g (b) Sprobe10g

Obrázek 2.1: Základní desky síťových sond

2.1 Záchyt cílového provozu na základě L7 obsahu

Cílem této úlohy je na základě popisů identifikátorů cílového provozu vyexportovat celistvé síťové toky, které odpovídají danému identifikátoru provozu. Identifikátor provozuj je složen z identifikátoru protokolu (například SIP) a a regulárního výrazu, který je vyhledáván v daném toku (např. "sip:\S*@server\.com"). Pro každý protokol definuje stavový automat a pozice v datech ve kterých se následně *RV* vyhledávají. Při nalezení daného *RV* se zpětně exportuje i historie toku. Důvodem je snaha mít celistvý tok, například při zachycení těla emailu je potřeba mít i hlavičku.

Úloha záchytu na základě L7 je tedy složena ze čtyř základních pod-úkolů, které je nutno v rámci úlohy vyřešit.

- Rekonstrukce toků a stavů protokolu
- Vyhledávání regulárních výrazů
- Filtrování
- Bufferování a pozdní export

Rekonstrukce datových toků a následná detekce stavů protokolu je z pohledu teoretického jednoduchá, protože pakety většinou obsahují sekvenční číslo a posloupnost kontrolních sekvencí je jasně definovaná. Úloha sebou však nese řadu implementačních problémů. Na procesorových architekturách s dostatečně velkou RAM je zpravidla možné uchovávat dostatečně dlouhou historii provozu tak, aby bylo možné toky rekonstruovat. Toto však neplatí pro *FPGA*. Navíc v reálném prostředí mohou toky končit nekorektně nebo můžou některé zprávy chybět. Tím dochází k zaplňování *flow-cache* mrtvými záznamy a implementace timeout je komplikovaná, protože vyžaduje zamykání *flow-cache*, které silně degraduje výkon.

Vyhledávání regulárních výrazů je obvykle časově nejnáročnější podúloha. Regulární výrazy se zřídka mění, avšak je potřeba je konfigurovat za běhu.

Akcelerace na paralelních architekturách Paralelní architektury jsou vhodné jen pro některé typy síťových aplikací. Je několik možností, jak paralelní architektury pro urychlení síťových aplikací použít. Typickými topologiemi jsou zřetězená linka nebo hvězda, kde centrální uzel pouze rozděluje vstupní pakety na výpočetní uzly. Topologie zřetězené linky je vhodná, pokud nad každým paketem musí být provedeno několik přibližně stejně složitých operací. Topologie hvězdy je vhodná pokud lze úlohu rozdělit na nezávislé podúlohy, jako je například paketový filtr. Časté jsou také kombinace, kdy část je prováděna paralelně a zbytek zřetězeně [1], [4].

Problém analýzy na L7 vrstvě však nemá vhodné parametry ani pro čistě paralelní architekturu ani pro zřetězenou linku. Vstupní toky nelze jednoduše rozdělit do více nezávislých skupin a ani úlohy nelze zřetězit, protože počet operací pro daný paket záleží na jeho obsahu. Navíc náročnost zpracování je dána obsahem paketu.

Akcelerace na GPU GPU akcelerátorů jsou zástupci silně paralelní architektury s poměrně velkou propustností do centrální paměti. Tyto akcelerátory jsou složeny ze skupin procesorů. GPU jsou vhodné zejména pokud je nutné provést jednu operaci na více datech. V Q1 2018 již existovaly grafické akcelerátory, které měly v FP32 výkon $12TFLOPS$, $12GB$ paměti s propustností $650GB/s$ při spotřebě $250W$. GPU mají tedy ve srovnání s ostatními akcelerátory výrazně větší teoretický výkon. Jíž v roce 2013 existovaly aplikace na zpracování síťového provozu, které GPU využívaly. Bohužel pro síťové aplikace, GPU akcelerátory nejsou soběstačné a vyžadují řízení z centrálního procesoru. Pro zpracování síťového provozu o vysokých rychlostech je tedy potřeba i výkonné CPU. Úzké hrdlo pak vzniká na hlavní sběrnici, a proto je nutné pakety na procesoru ještě předzpracovávat. Dále předpokládejme jako hlavní sběrnici *PCI-E v3.0*, drtivá většina výkonných grafických akcelerátorů v době psání této práce používá právě tuto sběrnici. *PCI-E* je nesdílená sběrnice a přístup k více zařízení na většině procesorů nezpůsobuje výkonnostní problémy. V době psaní této práce existují i technologie pro přímý přistup k periferiím připojených na *PCI-E*. Jednou z takových technologií je například *NVIDIA GPUDirect*.

Při zpracovávání provozu L7 OSI vrstvě velmi často obsah paketu řídí jakým způsobem se bude paket zpracovávat. To je pro GPU velmi nevýhodné, protože jednotlivá vlákna

budou provádět jiný kód a architektura GPU na to není uzpůsobená. Jev, ke kterému na GPU v takových případech dochází, se jmenuje *divergence vláken* a dokáže velmi silně degradovat výkon aplikace.

Pro vestavěné zařízení, podobného formátu jako je zařízení *Sprobe10g* používané v této práci, je však použití GPU nevhodné z důvodu vysoké spotřeby, velikosti i ceny. [8], [18]

Akcelerace na FPGA Technologie FPGA je silně paralelní architektura, kde paralelnost je na úrovni jednotlivých bitů a hradel. Jedná se o rekonfigurovatelný čip, který umožňuje z elementárních bloků vyskládat téměř libovolné propojení a tím sestavit čip na míru. Technologie FPGA umožňuje velmi výrazné zrychlení řady síťových aplikací. FPGA jsou používány v řadě síťových zařízení nejen jako náhražka ASIC. Velmi často bývají vybaveny velmi rychlými sériovými linkami, které je možno nakonfigurovat na přímou komunikaci se síťovým mediem.

Cena FPGA je obvykle velmi vysoká, nejen proto se obvykle používají společně s procesorem, ve formě akcelerační karty nebo na stejné desce. V FPGA lze velmi efektivně provádět základní filtrování hešování a jednoduché analýzy.

Slabým místem FPGA je velikost paměti, která bývá často velmi omezená. Při použití externí paměti pro FPGA bývá často omezující, náročné a drahé.

Pro analýzu na L7 vrstvě je však často potřeba historie toku a obecně velké množství paměti. Současně datové toky a stavové automaty jednotlivých protokolů jsou složité. To znamená, že je nutné používat FPGA pouze jako preprocesor vstupních toků a toky dále zpracovávat až v procesoru.

Akcelerace na síťových procesorech Síťový procesor je procesor jehož periferie a instrukční sada jsou optimalizovány pro zpracování síťového toku. Mezi obvyklé akcelerátory na síťových procesorech patří akcelerace manipulace s frontami a pakety, akcelerace hešování/filtrování a vyhledávání regulárních výrazů. Síťové procesory v dnešní době obsahují větší počet jader a akcelerátor, který distribuuje síťový provoz na jednotlivé vlákna.

Síťové procesory tedy obvykle obsahují hardware, který se obvykle implementuje v *FPGA* a zároveň mají vetší počet jader. Většina síťových procesorů je taktéž vybavena rychlým síťovým rozhraním. U síťových procesorů je obecně snaha udělat je natolik konfigurovatelné, aby je bylo možné použít v široké škále aplikací, avšak často se používají v konjunkci s *FPGA*.

Pattern-match akcelerátory jsou v poslední době navrhovány s podporou stavových pravidel a s možností dynamicky přidávat pravidla do filtrů.

Kapitola 3

Aplikace L7 analyzátoru PaSt

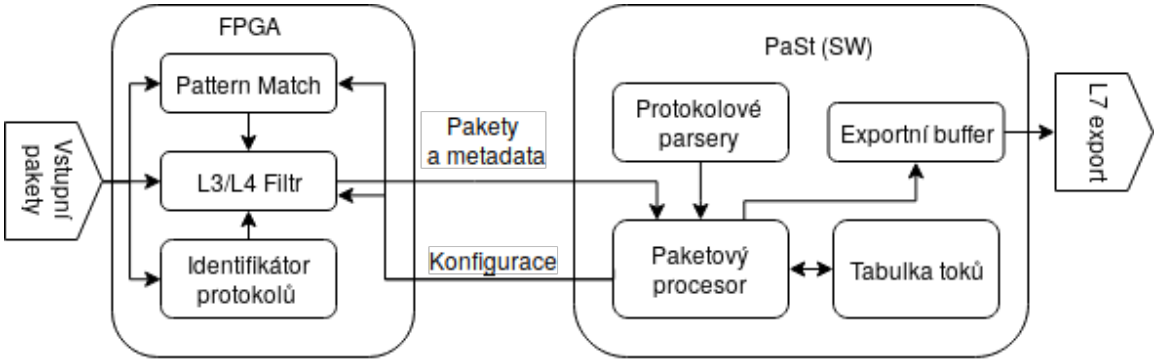
Aplikace *PaSt* (Packet Stack) slouží pro analýzu a cílený odposlech síťového provozu na L7 OSI[1] vrstvě, dále jen záchyt. Aplikace zachytává síťové toky aplikačních protokolů na základě uživatelem specifikovaných L7 identifikátorů. L7 identifikátor tvoří regulární výrazy a L7 protokol. Pro každý protokol však platí jistá specifika. Pro některé protokoly je vymezen nebo je možné vymezit rozsah, ve kterých datech se mají regulární výrazy vyhledávat. Například u protokolu *FTP* lze zachytávat pomocí jména souboru, uživatele atd. Podpora pro jednotlivé protokoly je vysvětlena na konci této kapitoly. Výsledek představují všechna data síťových toků, která byly vyměněny odposlouchávanými účastníky síťového provozu. Respektive všechna data, která byla dostupná. Tyto data jsou pak dále odesílána na ukládací server, kde jsou dostupné pro podrobnější analýzu a dlouhodobé uložení. Tento výstup je přenášen pomocí zabezpečeného *TCP* spojení na ukládací server. Tento server se nazývá *mediační funkce*.

Aplikace *PaSt* je úzce provázaná s hardwarem platformy na které běží. Tato platforma obsahuje FPGA s akcelerátory, díky kterém může tato aplikace pracovat bez zpomalení i na saturované lince. Avšak podmínkou je, aby zachytávaný provoz tvořil jen část provozu. Tento požadavek je v drtivé většině případů splněn. Pro aplikace legálních odposlechů je typické, že zájmový provoz tvoří pouze malou část provozu na lince.

Na software *PaSt* lze se zanedbáním implementačních specifik nahlížet jako na zřetězené paketové procesory. Zjednodušené blokové schéma je zobrazeno na obrázku 3.1. Prvním procesem, který se v rámci aplikace *PaSt* odehrává, je předfiltrovávání. Předfiltrovávání probíhá poloautonomně ve firmware *FPGA*. Firmware v FPGA rozeznává potenciálně zajímavé pakety a přeposílá je do software. Hardware obsahuje filtr, pomocí kterého jsou ignorovány nežádoucí síťové toky. Síťové toky, u kterých je potenciál, že se mohou stát odposlouchávanými se v terminologii aplikace *PaSt* označují jako *Potenciální začátky odposlechů* (dále jen PZO). PZO v první fázi identifikují akcelerátory v *FPGA* a autonomně přidávají dočasná pravidla do hlavního filtru, která zaručují přeposlání paketů daného toku do software. Tyto pravidla musí být ze software v požadované době potvrzena jinak hardware automaticky odkloní datový tok ze software. Jedná se o formu pojistky proti zahlcení, kdy jsou prioritizovány toky, které jsou jistě zajímavé před těmi, které jsou potenciálně zajímavé.

Detekce *PZO* ve firmware *FPGA* pobíhá za pomocí *identifikátoru protokolů* a akcelerátoru pro vyhledávání regulárních výrazů dále označovaný jako *Pattern match*. Všechny akcelerátory staré platformy jsou popsány v sekce 3.1. Detekce může trpět falešnými pozi-

[1] ISO7498



Obrázek 3.1: Blokové schéma síťového analyzátoru aplikačních protokolů PaSt.

tivy. Falešným pozitivům se není možné levně vyhnout s použitím dnešních technologií a poznatků. Nároky na zdroje takovéhoto hardware by byly enormní.

Po vstupu do software probíhá bufferování paketů z potenciálně zajímavých síťových toků aplikačních protokolů. V průběhu bufferování je postupně ověřováno, zdali daný síťový tok má být skutečně odposloucháván. Ověření znamená kontrola výskytu daného L7 identifikátoru. Toto ověření je v současné implementaci nutné, protože implementace vyhledávání regulárních výrazů v hardware přidání falešně pozitivní hlášení. Doba ověření závisí na charakteru síťového toku. Teoreticky omezená není avšak při nedostatku paměti jsou postupně zahazovány buffery nejstarších toků. Daný síťový tok může být také explicitně zahozen například ukončením spojení mezi uživateli. Při zahození je i přenastaven filtr v *FPGA*, který způsobí, že se síťový tok do software nadále nedostává. Při potvrzení výskytu L7 identifikátoru je daný síťový tok exportován a to včetně historie.

Potvrzování odposlechů síťových toků probíhá v několika fázích, které jsou pro každý protokol specifické. Pro každý podporovaný síťový protokol past obsahuje parser, který v datech vyhledává L7 identifikátory. Hlavní částí těchto parserů je vyhledávání regulárních výrazů.

Export probíhá přes zabezpečený kanál na *mediační funkci*. Toto zařízení ukládá zachycené síťové toky a umožňuje jejich pozdější analýzu a dlouhodobé ukládání. *Mediační funkce* také slouží k agregaci dat z více sond. Současná implementace tohoto serveru ukládá data ve standardním formátu souborů *PCAP*. A je tedy možné je pohodlně prohlížet s využitím běžně dostupného software.

Zpracování paketů v hlavní paketové smyčce Softwarová část má na vstupu pakety s meta-informacemi. Meta-informace obsahují časovou značku, příznaky protokolů a pravděpodobného nalezení regulárních výrazů. Pro každý paket je vyhledán nebo vytvořen stav daného toku v *tabulce toků*. Tato tabulka toků obsahuje informace o tocích, odkazy na vyrovnávací paměti toku a podobně. Obsahuje tedy zcela jiné informace než například tabulka toků v *NetFlow*. Zpracování paketů v hlavní paketové smyčce pak probíhá podle pole příznaku příznaků z hardware a stavu toku daný paket. Volitelně jsou nad paketem spuštěny požadované aplikační parsery. Paket může být také uložen do vyrovnávací paměti nebo odeslán do exportního bufferu. V *tabulce toků* jsou uloženy informace jen o odposlouchávaných tocích a potenciálně odposlouchávaných tocích. Podle stavu síťového toku software provádí konfiguraci hardwarových filtrů tak, aby omezil přísun paketů, které nesouvisí s odposlouchávaným provozem.

Síťový tok se stane potenciálním odposlechem, pokud se jedná o protokol, který je odposloucháván a je možné, že odposlouchávaný motiv se v toku objeví. Pro tyto síťové toky neprobíhá žádný export a jejich historie je udržovaná ve vyrovnávací paměti aplikace. Při potvrzení odposlechu jsou zpětně vyexportovány i pakety, které byly ve vyrovnávací paměti.

Hlavní smyčka je nejnáročnější část aplikace *PaSt*. Pro *PZO* platí, že zdaleka nejnáročnější operace je spouštění protokolových parserů, které lze popsat jako vyhledávání regulárních výrazů. Dále i konfigurace akcelerátorů v *FPGA* může být časově náročná operace. Například při nutnosti vyčistit pravidla filtru kvůli nedostatku paměti. Pro potvrzené záchyty už protokolové parsery spouštěny nejsou a nejsložitější operací se stává vyhledávání v tabulce toků. Toto vyhledávání lze popsat jako vyhledávání ve čtyřech hešovacích tabulkách. V ojedinělých případ může dojít i opakovanému vyhledávání. Následné bufferování a export nejsou z pohledu výkonu kritické. V současnosti *PaSt* využívá vlastní paměťový pool, který zabraňuje realokacím bufferů pro příchozí pakety. S použitím knihovny *DPDK* však bude muset být nahrazen.

Aktuálně podporované protokoly:

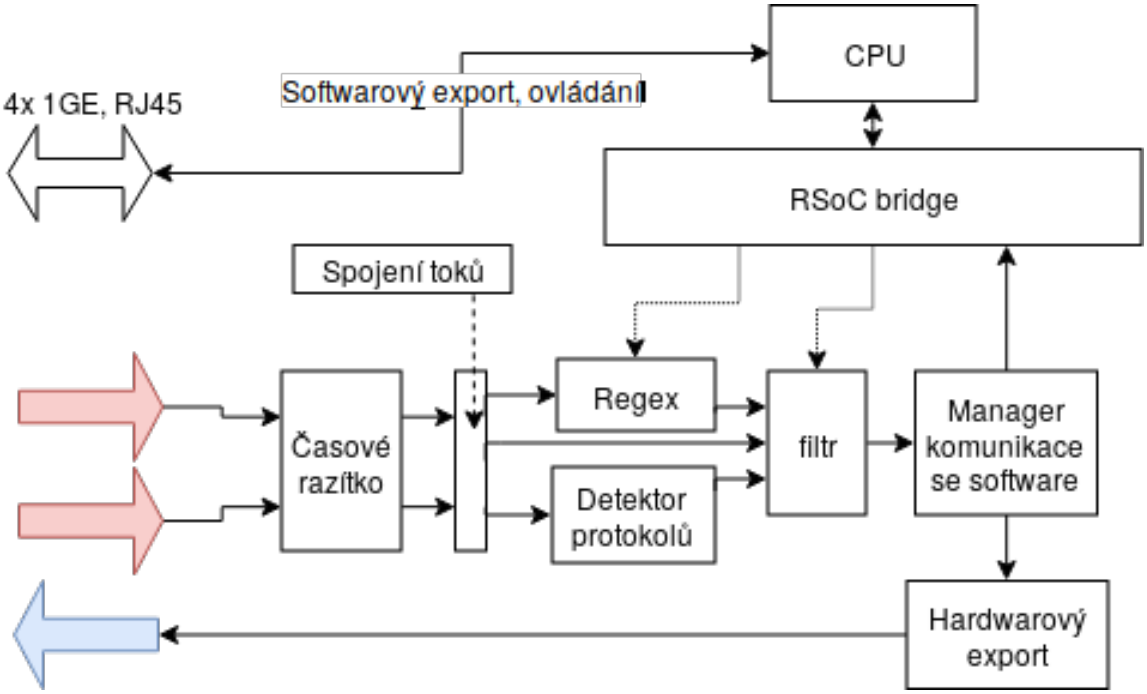
- SMTP - Podporován záchyt od MAIL FROM pro e-mail, ve kterém se objevuje hledaná e-mailová adresa v MAIL FROM, nebo v RCPT TO. Zachytává se až do konce SMTP toku.
- FTP - Záchyt celého sezení FTP (podle přihlašovacího jména), nebo zachytávání přenosu konkrétního souboru příkazy RETR a STOR (pouze informace z řídícího kanálu).
- POP3 - Záchyt celého sezení POP3 (pokud není použito kódování base64), nebo záchyt konkrétního e-mailu při detekování Internet Mail Format (příkaz RETR).
- IMAP - Záchyt celého sezení POP3 (pokud není použito kódování base64), nebo záchyt konkrétního e-mailu při detekování Internet Mail Format.
- SIP - Záchyt sezení od detekovaného identifikátoru (SUBSCRIBE, INVITE, NOTICE) po první výskyt identifikátoru BYE.

3.1 Starý firmware FPGA

Stará platforma měla procesor a FPGA v jednom pouzdře. To umožňovalo rychlý přístup z FPGA do RAM procesoru a rychlou konfiguraci akcelerátorů. Nová platforma, popsaná v sekci 2, je pravým opakem. Nový firmware pro FPGA této platformy je popsán v sekci 5.2.

Původní hardwarová platforma pro aplikaci *PaSt*, zobrazená na obrázku 3.2. Firmware FPGA obsahoval zejména tří akcelerátory. *Akcelerátor regulárních výrazů* (tzv. *Pattern match jednotku*), *L3/L4 filtr* a *identifikátor protokolů*. Účelem firmware *FPGA* bylo odklonit nezajímavé toky a umožnit tak software, aby mohl zpracovávat pouze potenciálně zajímavé toky. Cílem je dosáhnou, co nejvyšší výkonnosti. Hardwarové jednotky jsou ze software aktivně konfigurovány. Celý systém tedy pracuje v konceptu SDM[2].

[2]Software Defined Monitoring



Obrázek 3.2: Blokové schéma platformy Sprobe1g.

L3/L4 filtr je komplex síťových filtrů založených na kukaččím hašování. Umožňuje filtrovat vstupní pakety podle n-tic IP, port, protokol, prefix a další. Filtr navíc umožňuje na základě vstupů z ostatních jednotek autonomně přidávat/mazat pravidla. Pravidla mohou obsahovat čítač, který se dekrementuje s použitím pravidla a při dosažení nuly je pravidlo automaticky odstraněno.

Pattern match je konfigurovatelný akcelerátor vyhledávání regulárních výrazů založená na nedeterministických konečných automatech. Regulární výrazy je možno konfigurovat za běhu. Konfigurace spočívá ve zkompilování regulárního výrazu a jeho namapování do architektury teto jednotky a jeho nahrání do jednotky. Vstupem této jednotky jsou pakety, ve kterých jsou regulární výrazy vyhledávány. Výsledkem je pole příznaků, ve kterém každá položka vyjadřuje příznak pravděpodobného nalezení daného regulárního výrazu. Tato jednotka tedy nedokáže s jistotou určit zdali se regulární výraz v paketu skutečně nachází a místo toho jen indikuje pravděpodobný nález. Jednotka kvůli propustnosti musí zpracovat 8 znaků za takt. Při sestavení deterministického automatu pro regulární výrazy s použitím více znaků za takt roste exponenciálně počet stavů takového automatu. Nárůst je obrovsky i pro nedeterministické automaty a automat je tak dále redukován se snahou zmenšit spotřebu hardwarových zdrojů a současně zachovat co nejpřesnější detekci. Množství zdrojů FPGA je velmi omezené a zmenšení nároků na zdroje je nutné. Proto muselo být použito řešení, které má falešné pozitivy.

Identifikátor protokolů je hardwarová komponenta, která detekuje typ protokolu pro daný paket. Pracuje na bázi vyhledání pevně nastavených řetězců nebo hodnot v paketech.

3.2 DPDK - Data Plane Development Kit

DPDK je framework pro tvorbu síťových aplikací. Tento framework vznikl, aby umožnil zpracováváni vysokorychlostního provozu na procesorech Intel. Brzy na to se však stal multiplatformní. *DPDK* umožňuje psát aplikace, které pracují běžně s 40GE a rychlejšími. Například na síti 10GE při nejmenší velikosti paketů má procesor o taktu 3GHz pouze 201 taktů na zpracování paketu. Současně přepnutí kontextu trvá na většině procesorů více než 1000 taktů. To je jeden z důvodů proč bez použití *DPDK* nebo podobného frameworku není zpracování na rychlosti linky možné na žádném procesoru standardní architektury. Pro dosažení maximálního výkonu *DPDK* využívá nízko-úrovňový přístup k hardware a obchází jádro operačního systému a další pokročilé techniky [6]. Pro pochopení toho jak *DPDK* dosahuje své rychlosti je nutné, aby čtenář znal následující pojmy.

Paměťové prostory v operačních systémech Ve většině dnešních operačních systémů jsou především dva druhy paměťových prostorů. Paměťový prostor jádra tzv. *kernel space*, ve kterým pracuje jádro operačního systému, ovladače a další moduly. A další paměťový prostor ve kterém pracují obvyklé aplikace spuštěné uživateli, tzv. *user space*. Z *user space* není možné přímo přistupovat do *kernel space*. To je to nutná podmínka pro zachování bezpečnosti a paměťové konzistence paměti operačního systému. Z *kernel space* je možné přistupovat do *user space*, ale pokud má data přijímat aplikace v *user space* je potřeba tyto data do *user space* nakopírovat nebo namapovat. Při přechodu mezi *user space* a *kernel space* je nutné přepnutí kontextu, tato operace spočívá v přenastavení aktuálního jádra procesoru tak, aby začalo vykonávat jiné vlákno. Přepnutí kontextu je velmi náročná operace, která navíc často způsobí další výpadky ve vyrovnávacích pamětích cache a podobně.

VFIO je virtuální sběrnice a rozhraní ovladačů, pomocí kterého je možné namapovat fyzický prostor hardwarových prostředků přímo do *user space*. S pomocím *VFIO* je možné do *user space* i delegovat přerušení a další. Ovladače pro *DPDK* využívají většinou právě *VFIO*.

Hardwarové přerušení Hardwarové přerušení vysílají mimo jiné i periferie a příjemcem je zpravidla hlavní procesor, respektive některé z jader. Při nastavení příznaku přerušení je přerušen právě probíhající kód na daném jádře a podle tabulky vektorů přerušení je spuštěna obslužná rutina. Při přijetí přerušení je tedy také přepnut kontext. Dnešní systémy neumožňují uživatelským aplikacím přijímat přerušení a přerušení je nutno delegovat například pomocí *VFIO* anebo je zpracovávat v kernelu. Přerušení dokáží silně degradovat výkon, pokud při přerušení dochází příliš často. Přerušení jsou často optimalizována různými technikami, tak aby se počet přerušení minimalizoval ale i přes to nadměrné používání přerušení silně degraduje procesorový výkon. Na superskalarních architekturách dnešních procesorů přerušení způsobují ještě další ztráty výkonu.

Stránkování, hugepages Virtuální paměť je rozdělena do bloků, které se jmenují stránky. Stránky mají obvyklou velikost $4096B$. O stránkování se stará hardwarová jednotka, která se jmenuje *MMU* (Memory Management Unit), obdobně *IOMMU* pro periferie. Tato jednotka využívá datovou strukturu tabulku stránek, která slouží k překladu virtuálních adres na fyzické, které jsou nutné k přístupu do paměti a podobně. S rostoucím paměťovým prostorem roste velikost tabulky stránek. To v kontextu dnešních architektur znamená to, že roste počet úrovní tabulek stránek. Procházení tabulek je časově náročná operace, proto

MMU obsahují malou a velmi rychlou vyrovnávací paměť *TLB*[3], ve které jsou uloženy překlady pro některé adresy. V aplikacích, které kladou důraz na výkon, je vhodné použít větší stránky a tím zvýšit rychlost vyhledávání *MMU* a i zvýšit pravděpodobnost výskytu v *TLB*. Tyto větší stránky se nazývají *hugepages* a jejich typické velikosti jsou například $2MB$ nebo $1GB$. Velikosti podporovaných stránek závisí na operačním systému a architektuře. V případě DPDK závisí i na podpoře síťové karty.

Zero-copy princip Zero-copy je režim DMA[4]. DMA v tomto režimu kopíruje data přímo z/do bufferu aplikace bez použití další kopie. Při běžném přijetí paketu z běžného rozhraní tomu tak není, protože systém musí překopírovat data z *kernel space* do *user space*. Ovladače v *DPDK* dodržují tento princip, za účelem dosažení maximálního výkonu.

Způsoby, kterými DPDK zvyšuje výkon *DPDK* obchází linuxový *network stack*. Tím je odstraněna nutnost zbytečných kopírování mezi *user-space* a *kernel-space* a přepínání kontextů mezi linuxovým jádrem a aplikací. Ovladače postavené nad *DPDK* omezují používání přerušení a používají zero-copy princip. Paměť v *DPDK* je ve většině případů alokována z hugepage filesystemu. Použití hugepages výrazně snižuje zatížení *MMU* v systému a tím dosahuje dalšího navýšení výkonu.

DPDK vzniklo v rámci Intel Corporation, již delší dobu podporuje i jiné architektury. Podporovány jsou síťové karty a procesory jiných výrobců i architektur (mezi nimi i procesory architekturou ARMv7 a ARMv8 s DPAA a DPAA2 od NXP). *DPDK* není vymezeno pouze na síťové karty, obsahuje například i knihovny pro práci se šifrovacími akcelerátory. Tyto knihovny však lze chápat pouze jako doplňkové.

Použití *DPDK* však s sebou nese i některá úskalí. *DPDK* vyžaduje speciální ovladače a není možné jakkoli použít ovladače ze systému. Při použití DPDK ovladače není možné současně používat síťovou kartu mimo DPDK. Ovladače pro *DPDK* se nazývají *Pool Mode Driver* (dále jen *PMD*). *PMD* je ovladač běžící v *user space*, používá *run-to-completion* model. V *run-to-completion* modelu uživatelská aplikace v nekonečné smyšce zpracovává pakety a kontroluje vstupy. Většina *PMD* je vystavena nad *VFIO*. *VFIO* je sběrnice a rozhraní pro práci s DMA a paměťovými prostory za MMU z *user space*. *DPDK* síťové rozhraní je zcela nekompatibilní s linuxovým rozhraním a aplikace napsané pro linuxové síťové rozhraní není možné s *DPDK* použít.

DPDK je složeno z mnoha knihoven a ovladačů. Mezi esenciální knihovny v *DPDK* patří *librte_eal*, *librte_mempool*, *librte_mbuf* a *librte_ether*.

librte_eal je knihovna, která slouží k abstrakci od operačního systému. Zkratka znamená Environment Abstraction Layer. Obsahuje funkce pro zprávu paměti v hugepage filesystemu, funkce pro zpracování *DPDK* parametrů z příkazové řádky, funkce pro obsluhu přerušení a časovačů a inicializaci celého DPDK a spuštění jednotlivých výpočetních vláken s nastavenou afinitou na daná jádra CPU.

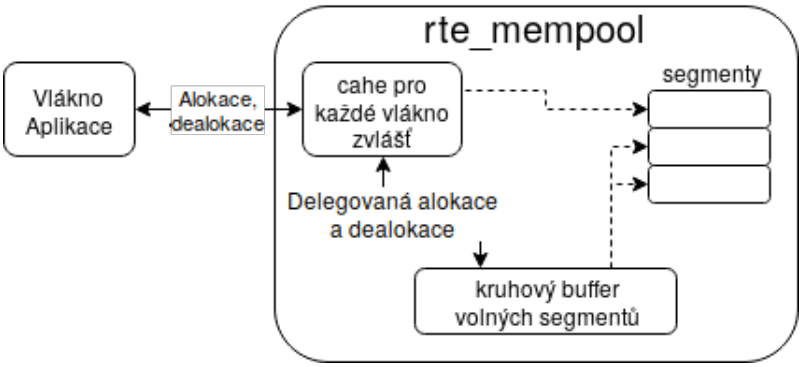
librte_ring je knihovna, která obsahuje implementace kruhových front. Tyto fronty jsou taktéž v *DPDK* hojně používány. Vyskytují se v implementacích paměťových poolu, používají se pro předávání paketů mezi síťovými kartami i mezi jednotlivými jádry. Fronty jsou implementovány jako bezzámkové fronty a podporují i větší počty konzumentů/producentů.

[3]Translation Lookaside Buffer

[4]Direct Memeory Access - hardwarová jednotka pro přesun dat

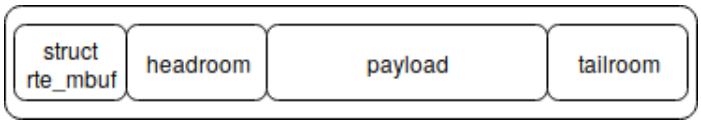
librte_mempool je knihovna, která obsahuje základní implementaci paměťového poolu, který nepřímo používají všechny DPDK aplikace a ovladače. Tato implementace paměťového poolu umožňuje alokace pouze položek o stejné velikosti, tím se předchází nadměrné fragmentaci paměti a výrazně to urychluje alokaci. Mempool implementuje několik metod, které je možné v rámci PMD přetížit. Mezi tyto funkce patří zejména funkce pro alokaci a dealokaci položek z mempoolu a funkce pro manipulaci s paměťovými bloky, které mempool používá.

Schéma 3.3 znázorňuje vnitřní strukturu objektu mempool. Mempool je složen z paměťových bloků označovaných jako segmenty. Tyto segmenty jsou uloženy do kruhového bufferu. Každé jádro, které mempool používá má v mempool alokovanou vyrovnávací paměť cache. S touto pamětí je minimalizována potřeba synchronizace mezi vlákny a segmenty se taktéž s větší pravděpodobností budou vyskytovat v hardwarové vyrovnávací paměti L1 cache daného jádra. Použití cache také minimalizuje potřebu prohledávat kruhovou frontu a vyhledávat v ní volné segmenty.



Obrázek 3.3: Schéma *rte_mempool*.

librte_mbuf je knihovna, která obsahuje implementaci paketového mempool. Tento mempool se jmenuje *rte_pktmbuf_pool* jedná se o implementaci mempoolu, kde položka odpovídá struktuře *rte_mbuf*. Tato struktura v *DPDK* reprezentuje paket nebo jeho část a je tedy klíčovou strukturou, se kterou pracují všechny *DPDK* aplikace. V mempoolu je uložena na začátku segmentu a odkazuje ne začátek datové části paketu a obsahuje základní informace o paketu jako je například jeho velikost nebo následující odkaz na další část paketu pro pakety větší, než je výchozí velikost payloadu nastavená při vytváření mempool. Segment mempoolu, ve kterém je *rte_mbuf* uložen není spojitá a obsahuje mezery, jak je znázorněno na obrázku 3.4. Tyto mezery slouží k zarovnání a pro případné potřeby síťových karet.



Obrázek 3.4: Segment v *rte_pktmbuf_pool*.

librte_ether je knihovna, která obsahuje základní API ethernetového rozhraní. V *DPDK* je ethernetový port reprezentován svým id, toto id je obdoba file deskritoru v Linuxu,

jedná se pouze o celočíselnou hodnotu. Tato hodnota odpovídá indexu portu na dané platformě. Indexy portů začínají od nuly. Aby bylo možné začít přijímat pakety z daného portu je potřeba port nejdříve nakonfigurovat pomocí funkce *rte_eth_dev_configure*, *rte_eth_rx_queue_setup* a dalších. Poté je možné port otevřít pomocí *rte_eth_dev_start* a začít přijímat pakety například pomocí funkce *rte_eth_rx_burst*. Pro odesílání je nutné si alokovat potřebný *rte_mbuf*, který je kontejnerem pro paket. Tato struktura musí být alokována z instance *rte_pktmbuf_pool*. Tento mempool lze vytvořit například pomocí *rte_pktmbuf_pool_create*. Polocí funkce *rte_mempool_get_bulk* lze z instance mempoolu alkokovat potřebné struktury *rte_mbuf*. Po vyplnění dat a velikosti je možné paket odeslat například pomocí *rte_eth_tx_burst*.

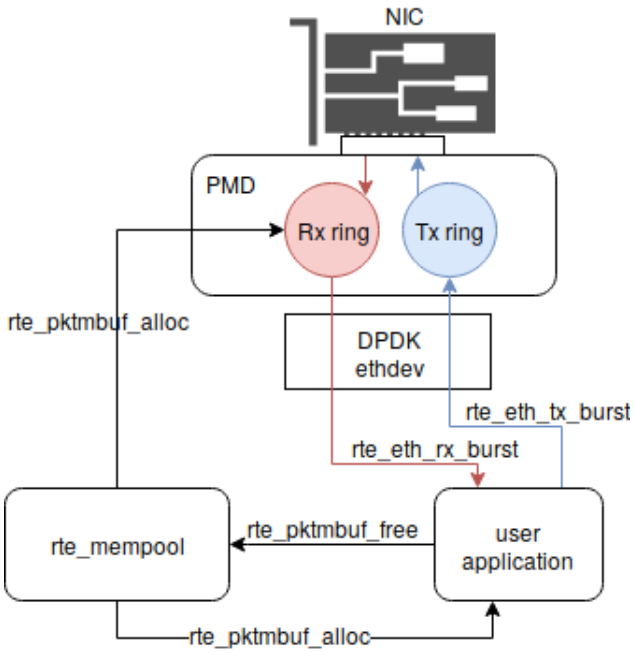
Kromě funkcí pro obvyklé operace s ethernetovými rozhraními obsahuje i pokročilejší funkce pro práci se statistikami a pro konfiguraci pokročilých periferii na síťových kartách jako je například práce s *RSS*[5], které slouží k rozdělování příchozích paketů do jednotlivých front na portech.

Typická aplikace využívající DPDK. Na obrázku 3.5 je schéma typické aplikace postavené nad frameworkem *DPDK*. Všechny části aplikace běží v *user space*. *PMD* přímo ovládá hardware síťové karty a zprostředkovává předávání paketů mezi uživatelskou aplikací a hardwarem síťové karty. Uživatelská aplikace přistupuje k frontám přes rozhraní *DPDK* ethernetového zařízení, které je implementováno v knihovně *rte_ether* popsané v předešlém odstavci. Správu paměťových bufferů obstarává objekt mempool, ke kterému přistupuje uživatelská aplikace i PMD ovladač. Odesílání a příjem paketů je prováděn pomocí funkcí z API ethernetového rozhraní *DPDK* pomocí funkce *rte_eth_rx/tx_burst*. Tyto funkce vždy pracují s dávkou paketů. Aplikace v nekonečné smyčce musí kontrolovat příchozí pakety. Výhodou je, že není potřeba přepínat kontexty procesoru ani využívat přerušení. Procesor tedy může nerušeně a efektivně zpracovávat síťový provoz.

Kompilace DPDK aplikací *DPDK* aplikace jsou obyčejné aplikace napsané v jazyce C/C++, avšak řada věcí při kompilaci je nestandardních. Kromě přepínačů pro aktivaci zřetězení, šifrovacích a hešovacích instrukcí je také potřeba zajistit správné pořadí konstruktorů modulů. Konstruktor modulu je funkce, která se obvykle spouští po spuštění programu ještě před spuštěním funkce main. Prostřednictvím těchto funkci jsou v *DPDK* mimo jiné registrovány ovladače, sběrnice i zařízení. Aby byly uživatelé odstíněni od těchto a dalších starostí, přichází *DPDK* s vlastním buildovacím systémem založeným na nástroji make. Tento přístup však sebou přinesl řadu nekompatibilit s ostatními knihovnami. V *DPDK* verze 18.02 nově přibyla podpora buildovacího systému Meson. Bohužel verze *DPDK*, kterou je nutné použít na této platformě, nemá pro tento buildovací systém podporu. V rámci této práce byl tedy použit nástroj CMake. Více o použité verzi *DPDK* v sekci 5.7.

Přínos *DPDK* je zřejmý, dochází k razantnímu zrychlení síťové aplikace. Pro nově vzniklou platformu jsou dostupné ovladače pro *DPDK*. Nic tedy nebrání tomu, aby aplikace *PaSt* používala tuto knihovnu a z její rychlosti také profitovala. Použití *DPDK* však sebou přinese řadu komplikací. Informace o těchto a dalších komplikacích jsou součástí kapitoly 5.3 a 5.5. Nepříjemné důsledkem je také zvětšení množství kódu aplikace a narůst složitosti její kompilace.

[5] Receive Side Scaling



Obrázek 3.5: Přehled toku paměťových bufferů v obvyklé DPDK aplikaci.

3.3 HWIO - Knihovna pro sdílený přístup k HW

Aplikace *PaSt* pro ovládání akcelerátorů v *FPGA* využívá ovladače postavené nad *C/C++* knihovnou *HWIO* 3.3. Tato knihovna umožňuje sdílený přístup k hardwarovým prostředků z *user-space* prostředí Linuxu. Knihovna *HWIO* na základě řetězce kompatibility (compatibility string) vyhledá v *device-tree*[6] kompatibilní hardwarové prostředky a namapuje je do adresového prostoru aplikace. Pro novou platformu bylo potřeba zajistit přístup k hardwarovým prostředkům jiného stroje. Byla použita architektura server-klient a protokol *TCP*. Sdílený přístup a dynamická alokace zdrojů byla zachována. Nově přibyl dávkový režim umožňující vykonávat více *I/O* operací na jednou, který kompenzuje zpoždění *TCP* připojení.

Předchozí verze podporovala pouze lokální přistup a hardwarovým prostředkům. Sdílení hardwarových komponent bylo zajištěno pomocí namapování stejného adresového prostoru. Podporován byl standardní Linux a Linux s použitím *RSoC frameworku*. *RSoC framework* je knihovna pro komunikaci s hardwarovými komponentami a její součástí je i specifický hardware. Tento framework je použit právě na staré platformě.

Server nově vzniklé implementace *HWIO* je kompatibilní s Linuxem i *RSoC frameworkem*. Klient je zcela nezávislý na použitém modu serveru a je tedy možné abstrahovat od způsobu přístupu k hardware. Současně je možné využívat hardwarové prostředky odkudkoli. Nová implementace umožňuje jednoduchý synchronní zápis a čtení s variabilní délkou dat. Dále je podporováno vyhledávání hardwarových prostředků a jejich dynamická alokace či dealokace. Maximální počet klientů je možné konfigurovat v době překladu. Připraven je také návrh asynchronní verze. Tento návrh však počítá s řadou změn v API knihovny a ponese sebou změnu téměř všech aplikací na platformě *Sprobe*. Z tohoto důvodu je implementace odložena do doby, než se zbytek týmu *Sprobe* seznámí s nově vzniklou platformou

[6] Adresářová struktura, ve kterém sou uloženy informace o hardwarové platformy.

Sprobe10g. Asynchronní přístup přinese i řadu dalších komplikací, ale především umožní razantně zvýšit výkon některých aplikací. Sekce 5.11 obsahuje návrhy na vylepšení této knihovny, které v současné době není možné a nebo rozumné provádět. Realizací těchto vylepšení výkonnost této knihovny dále řádově naroste.

Kapitola 4

Síťový SoC NXP LS2088

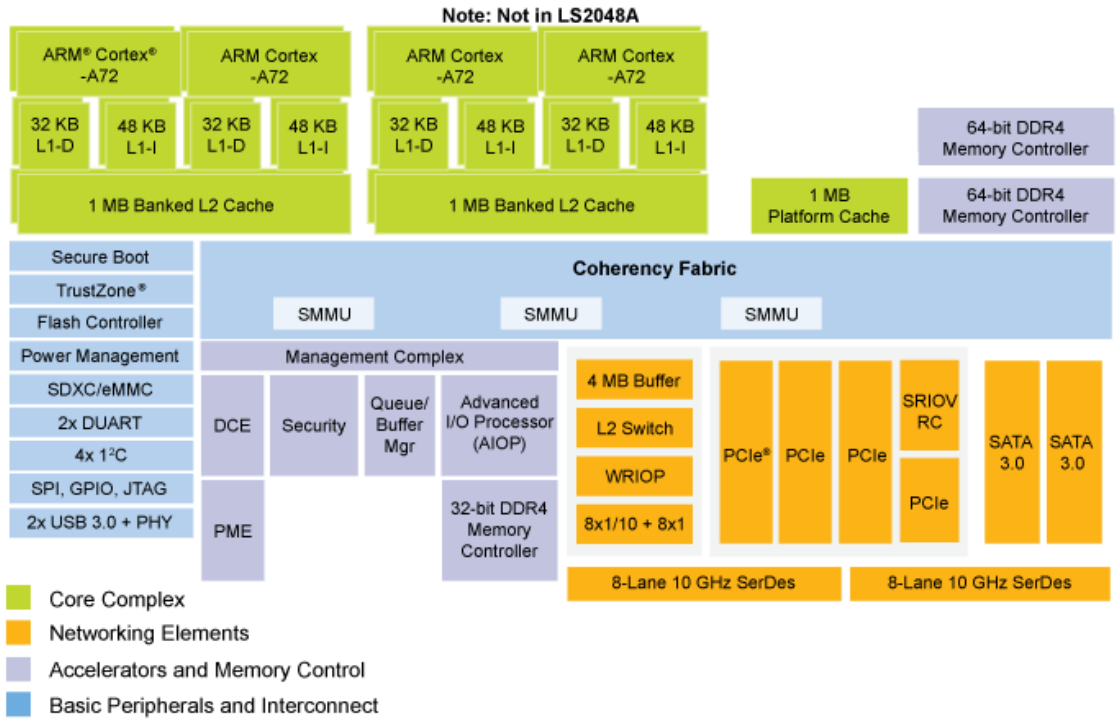
Síťový SoC NXP LS2088 tvoří především 8 jader ARM Cortex-A72 a skupina síťových akcelerátorů připojení pomocí *DPAA2* sběrnice.

Začátek této kapitoly popisuje systém procesorů, paměťových řadičů, periferie a cache procesoru NXP LS2088. V podsekci 4.1 je popsána sběrnice *DPAA2*, ke které jsou připojen síťové akcelerátory tohoto procesoru. Sekce 4.3 a 4.4 popisují strukturu, obsah a bootovací proces firmware pro tento procesor. V sekci 5.5 jsou rozebrány problémy implementace DPAA2 v době psaní této práce, tedy ještě před oficiálním datem vydáním tohoto procesoru.

LS2088 je 8-jádrový ARM Cortex-A72 s akcelerátorem pro zpracování síťového provozu a L2 switchem. Je optimalizován na nasazení v routerech a dalších síťových prvcích s přepínací kapacitou 80 Gb/s. Zjednodušené schéma je znázorněno na obrázku 4.1. Síťové akcelerátory jsou v procesoru připojeny na sběrnici DPAA2 (více v sekci 4.1). Pro síťová rozhraní existuje ovladač kompatibilní s knihovnou DPDK (vice v sekci 4.1.10). Seriové IO tohoto procesoru podporuje několik protokolů a je možno měnit funkci jednotlivých sériových linek podobně jako třeba u FPGA (více v sekci 4.3). Specifikace tohoto procesoru:

- CPU 8× Cortex-A72 2.0GHz
- 4 MB L2 cache
- 2× 64-bit DDR4 řadič
- 16 × 10GHz SerDes, které je možné nakonfigurovat jako 10GE/SATA 3.0/PCIe 2.0.
- 2× USB 3.0, 4× I2C, SPI, GPIO, 2× DUART SDXC a další

Procesorová jádra jsou sdruženy do clusterů po dvou se sdílenou L2 cache. L1 cache je privátní. L2 a L3 cache je exkluzivní. Všechny paměti cache jsou chráněny pomocí ECC, ale L1-instrukční jen částečně. L1-datová cache je dvoucestná, L2 16-cestná. L3 16-cestná prokládaná. Všechny akcelerátory jsou připojeny pomocí koherentního interconnectu *CCN-504*. Tento interconnect má kruhovou topologii. Sběrnice je taktována na 1.6GHz a je 128b široká. Sběrnice umožňuje obousměrný transport po kruhu a podporuje *snoop filtering*. *Snoop filtering* je technologie, která slouží k filtrování transakcí zajišťujících koherenci, výrazně tak šetří koncové body sběrnice, které nemusí zpracovávat transakce pro zajištění koherence, pokud daný úsek paměti nepoužívají. V hlavní interconnectu je integrovaná L3 cache.



Obrázek 4.1: Schéma NXP LS2088 (převzato z [9]).

Paměťový subsystém SoC obsahuje celkem 3 DDR4 paměťové řadiče. Hlavní dva jsou určeny pro CPU[1], třetí je určen pro akcelerátory na *DPAA2* [2]. SoC obsahuje navíc speciální paměťový buffer přímo na čipu založený na SRAM. Tento buffer je určen pro akcelerátory. Jeho velikost je $4MB$, rychlost čtení a zápisu je v součtu $230GB/s$. Paměť je chráněna pomocí ECC. Paměť je využívána podobně jako pro DDR řadič pro *DPAA2*. Jednotky pro zprávu bufferů mohou tuto paměť využít pro paketové fronty. Tyto paketové fronty se v terminologii nazývají PEB.

CPU má hardwarovou podporu pro virtualizaci a technologii *ARM TrustZone*. Paměťový prostor je v tabulce 4.1.

Podstatnou částí SoC NXP LS2088 je sběrnice DPAA2 a akcelerátory na ní. Tato sběrnice je popsána v sekci 4.1. Informace v této kapitole byly sepsány podle referenčního manuálu k procesoru [10] a k jeho SDK [15].

4.1 Akcelerátory a DPAA2

DPAA2 (Datapath Acceleration Architecture 2) je sběrnice a architektura navržená pro akceleraci síťových aplikací. Tato sběrnice je v době psaní této práce pouze uvnitř SoC NXP řady LayerScape, konkrétně v SoC LS2085/8. Tato práce se zabývá pouze výkonnějším modelem LS2088. Na sběrnici *DPAA2* jsou připojeny všechny ethernetové porty, síťový filtr, procesorové pole, šifrovací a komprimační akcelerátor a mnohé další. Ke každé komponentě

[1] $2 \times 64 - bit$, ECC, $34.1GB/s$, $2.133GT/s$, koherentní přístup, až $128GB$

[2] $32bit$, ECC, $6.4GB/s$, $1.67GT/s$, nekoherentní přístup, až 64GB

Fyzická adresa	Typ paměti	Velikost
0x00_0000_0000 - 0x00_000F_FFFF	Secure Boot ROM	1MB
0x00_0100_0000 - 0x00_0FFF_FFFF	CCSRBAR	240MB
0x00_1000_0000 - 0x00_1000_FFFF	OCRAM0	64KB
0x00_1001_0000 - 0x00_1001_FFFF	OCRAM1	64KB
0x00_2000_0000 - 0x00_20FF_FFFF	DCSR	16MB
0x00_7E80_0000 - 0x00_7E80_FFFF	IFC - NAND Flash	64KB
0x00_7FB0_0000 - 0x00_7FB0_0FFF	IFC - CPLD	4KB
0x00_8000_0000 - 0x00_FFFF_FFFF	DRAM1	2GB
0x05_0000_0000 - 0x05_07FF_FFFF	QMAN S/W Portal	128MB
0x05_0800_0000 - 0x05_0FFF_FFFF	BMAN S/W Portal	128MB
0x08_8000_0000 - 0x09_FFFF_FFFF	DRAM2	6GB
0x40_0000_0000 - 0x47_FFFF_FFFF	PCI Express1	32G
0x48_0000_0000 - 0x4F_FFFF_FFFF	PCI Express2	32G
0x50_0000_0000 - 0x57_FFFF_FFFF	PCI Express3	32G

Tabulka 4.1: Paměťový prostor procesoru NXP LS2088

je připojen specifický počet kontrolních a datových kanálů, které slouží k ovládání a umožňují paketovou komunikaci [15] mezi jednotlivými akcelerátory a i částmi software. Zapojení této sběrnice je dynamické. Jedná se tedy o formu virtuální rekonfigurovatelné sítě. Obecně lze říct, že možnosti propojení jsou omezené pouze počtem hardwarových prostředků. Konkrétní akcelerátory jsou popsány v podsekci 4.1.1.

Pro snadnější komunikaci se softwarem *DPAA2* sběrnice pracuje s objekty. Objekt je virtuální reprezentací akcelerátorů. Mapování objektů na hardware je specifické pro každou třídu objektů. Objekt může být složen z více částí některých akcelerátorů. Ve většině případů však objekt odpovídá některé funkci akcelerátorů jako je například síťový port pro software (*DPNI*), L2 switch (*DPSW*) nebo propojení (*DPCON*). Tyto objekty jsou popsány v podsekci 4.1.8.

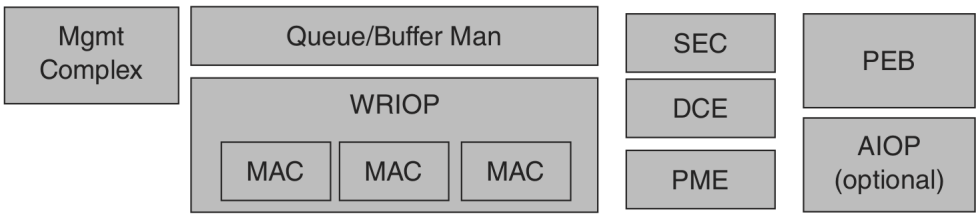
4.1.1 DPAA2 akcelerátory

Akcelerátory v procesoru připojeny na sběrnici *DPAA2* (viz schema 4.1). Tato sběrnice je ze softwarového pohledu podobna například PCIe. Podporuje dynamické připojování (hotplug) a přemapovávání. Z pohledu HW se však jedná o komplexní systém. Konfigurace všech akcelerátorů probíhají prostřednictvím *MC portálů*. *MC* je jediný akcelerátor, který je přímo přístupný z adresového prostoru procesoru. Datové cesty na sběrnici DPAA2 jsou odděleny a lze s nimi téměř libovolně připojovat akcelerátory včetně řetězení a napojení na IO. Na sběrnici vyúsťují všechny ethernetové porty a přeposílání paketů do hlavní RAM zajišťuje farma DMA.

4.1.2 MC, BMan a QMan a další

Zkratkami MC, BMan a QMan jsou označovány akcelerátory Management Complex, Buffer Manager, Queue Manager v tomto pořadí.

Management Complex je hardwarový akcelerátor, který zjednodušuje přístup k zařízením připojených na *DPAA2*. Tento komplex obsahuje konfigurační porty, které lze připojit na jednotlivé akcelerátory a tím je ovládat. *MC* je nepostradatelný pro funkci *DPAA2*.



Obrázek 4.2: DPAA2 HW akcelerátory (převzato z [9]).

Tento akcelerátor vyžaduje firmware, který je nutno zavést v hlavním zavaděči před bootováním Linuxu. Tento firmware se skládá z mikrokódu pro *MC* a konfiguračních souborů *DPAA2* (*DPL*, *DPB*), firmware je popsán v sekci 4.3.

Buffer manager je akcelerátor, který slouží ke správě paměťových bloků pro buffery na úrovni hardware. Tento akcelerátor zajišťuje koherenci. *BMan* dokáže alokovat buffery ve více paměťových prostorech. Pod pojmem normální buffer se skrývá buffer, který je uložen v DDR připojené k DPAA2. Na platformě použité při psaní této práce je však ještě jeden typ paměťového bufferu, který se jmenuje *PEB (Performance Enhanced Buffer)*. *PEM* je umístěn ve speciální $4MB$ paměti a čipu. Výkonnostní parametry této paměti jsou pro čip použitý v této práci uvedeny v sekci 4.

Queue manager zpravuje bezzámkové fronty, které jsou naalokovány z paměťových bufferů, které zpravuje *BMan*. Tyto fronty jsou využity pro komunikaci se software i pro buffery mezi akcelerátory. Výchozí velikost fronty, která využívá *PEB* je $10KB$. Díky koherentnímu propojení s cache procesoru jsou data udržována koherentní mezi procesory a akcelerátory.

Management komplex obsahuje také statistický engine, který obsahuje čítače statistik pro *DPAA2*.

4.1.3 WRIOP - Wire Rate I/O processor

WRIOP je programovatelný HW akcelerátor, který umožňuje jednoduchou klasifikaci, třídění paketů a úpravy síťového provozu. Podporuje kontrolu a generování TCP/IP hlaviček, statistiky, QoS. Umožňuje aplikovat jednoduchá pravidla na síťové toky. Tyto jednoduché pravidla zahrnují zahazovaní paketů, vybírání a inteligentní distribuci, to vše na základě hlavičky paketu. Akcelerátor umí pracovat s některými standardními protokoly a navíc umožňuje uživateli definovat 3 další 32b položky hlavičky. Na základě hodnot z hlavičky probíhá klasifikace, kde pro každou klasifikační třídu je možné definovat akci, která má být s paketem provedena. Akce mohou být kromě založení a přesměrování také uplatňování ořezávání provozu (traffic policing) a další kontroly toku. Akcelerátoru umožňuje definovat až 256 profilů (RFC4115, RFC2968) kontrol toků, které je možné aplikovat pro libovolnou klasifikační třídu.

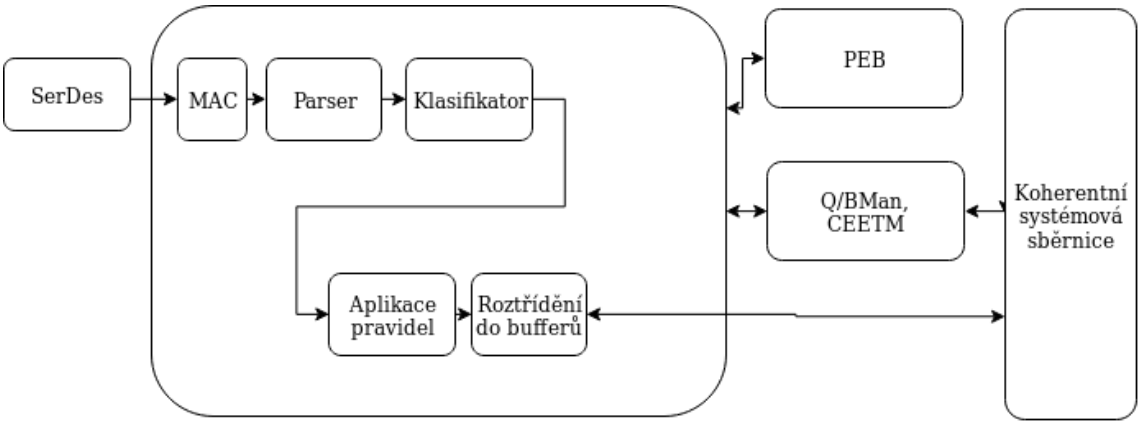
Části *WROIP* se v SW jeví jako *DPNI*, *DPMAC*, *DPSW*, *DPIO* a *DPCI*.

Schema zobrazeno na obrázku 4.3. Parser je chopen extrahovat libovolné položky z prvních $256B$ paketu. Klasifikátor vybírá na základě položek z parseru akci. Obsahuje hardwarový *ACL (Access Control List)*, který je implementován jako *TCAM (Ternary Content Addressable Memory)*.

Akcelerátor dokáže přeposlat až $88GPkt/s$. Akcelerátor obsahuje celkem 16 fyzických portů z nichž 8 lze použít pro 10G Ethenete a zbylých 8 pro 1G/2.5G Ethernet. Všechny

port podporují *IEEE 1588v2*[3] a *802.1AE* (*MACsec*). *WRIOP* navíc obsahuje ještě další dva recyklační porty.

WROIP může být společně s manažerem front nakonfigurován tak, aby plnil úlohu L2 switche, takto vzniklý objekt se pak v terminologii *DPAA2* označuje jako *DPSW*.



Obrázek 4.3: Schéma WROIP.

DPSW Akcelerátor L2 switche může mít až 16 fyzických portů a až 2 recyklační porty. Recyklační porty slouží k vytvoření virtuálních propojení, zpravidla používané pro propojení mezi virtuálními stroji.

Switch podporuje pravidla se specifikací ACL[4], *VLAN*, portu a *MAC adresy*. Ethernetový multicast je implementován s ohledem na přemostění *VLAN* dle *802.1Q*. QoS a rozložení provozu je implementováno podle *802.1Q*.

Každý port podporuje až 8 front, pro výběr paketů z front použít *round-robin* anebo vážený *round-robin*. Podporovány jsou také *MSTP*, *RSTP* a *GMRP* podle *802.1Q*. Switch podporuje hardwarové učení adres. Tuto funkcionalitu je možné vypnout a použít čistě jen statickou konfiguraci. Switch podporuje také stárnutí pravidel. Broadcast a multicast jsou taktéž podporovány, stejně jako ořezávání vstupních toků.

L2 switch je možné naalokovat pomocí nástroje restool popsaného v sekci 4.2 například příkazem 4.1. Při alokaci je možné specifikovat počet portů, nakonfigurovat učení switche a nastavit velikosti a počet front a statistik. Maximální podporovaná *MTU*[5] je 10KB. Maximální počet záznamů L2 směrování je 1024. Maximální počet VLAN je 16 přičemž jedna je používána hardware a je možné konfigurovat jen zbylých 15 [11].

```
1 restool dpsw create --num-ifs=4 --max-vlans=8 --max-fdb-mc-groups=300 --options=
      DPSW_OPT_TC_DIS,DPSW_OPT_FLOODING_DIS
```

Algoritmus 4.1: Příkaz pro vytvoření L2 switche na DPAA2

4.1.4 DCE - Decompression/Compression engine accelerator

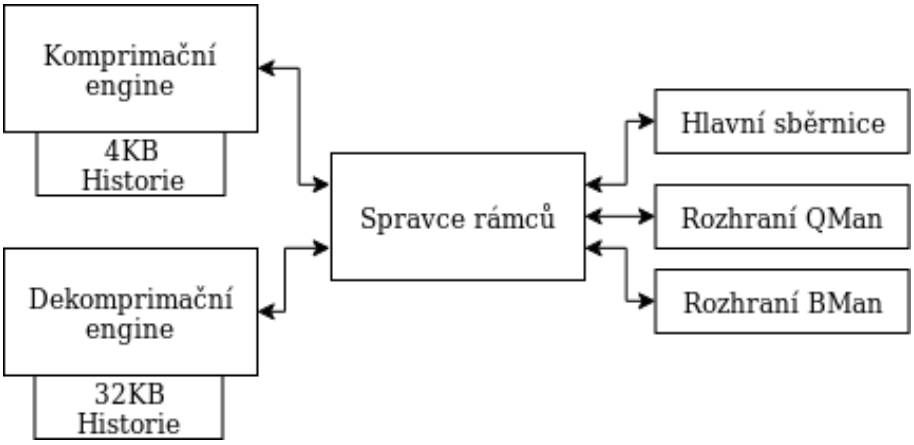
Jedná se 2. generaci tohoto akcelerátoru od NXP. Podporované formáty:

- Deflate podle RFC1951

[3]PTP - Precision Time Protocol
[4]Access Control Lists
[5]Maximum Transmission Unit - Maximální velikost paketu

- GZIP podle RFC1952
- Zlib podle RFC1950

Akcelerátor dokáže zpracovat až 12 Gb/s komprese/dekomprese nebo 20 Gb/s pří kombinované zátěži, avšak propustnost záleží na použitém algoritmu, na datech i nastavení. Na *DPAA2* sběrnici je akcelerátor ukryt za *DPDCEI* objektem. V době psaní této práce je akcelerátor možné používat pouze s ovladači v Linux kernel. User-space ovladače jsou plánovány. Hrubé schéma je zobrazeno na obrázku 4.4. Kompresní a dekompresní jednotka jsou odděleny ale DMA je sdíleno.



Obrázek 4.4: Schéma DCE.

DCE je možné naalokovat pomoc nástroje *restool* popsaného v sekci 4.2 příkazem 4.2. Kompresní algoritmus lze prozatím specifikovat pouze voláním nízkoúrovňového rozhraní *DPAA2*.

```
1 restool dpdcei create --engine=DPDCEI_ENGINE_COMPRESSION --priority=1
```

Algoritmus 4.2: Příkaz pro alokaci portálu kompresního akcelerátoru.

4.1.5 PME - Pattern Matcher Engine

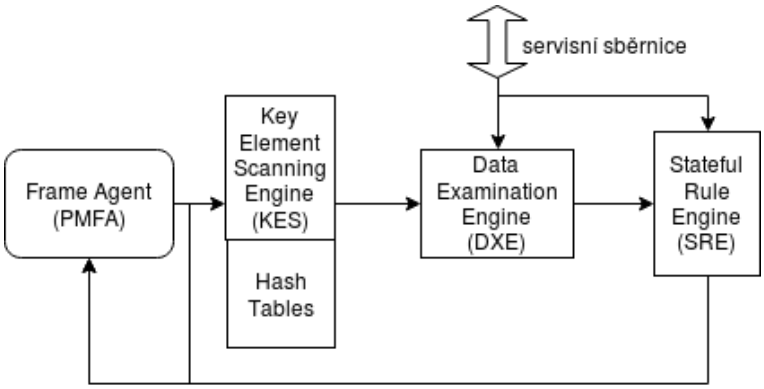
PME je akcelerátor pro vyhledávání regulárních výrazů. Jedná se již o 3. generaci tohoto akcelerátoru od NXP. Vstupní regulární výraz je zadáván ve stejném formátu jako ve skriptovacím jazyce Perl. Akcelerátor je založen na kombinovaném hašování a nedeterministických konečných automatech. Podporovány jsou i libovolné znaky, opakování, rozsahy a ukotvení. Pro každé nalezení výrazu je možné uživatelsky zadefinovat událost. Tato událost může být využita k detekci vzorů, ohodnocení shod anebo sledování stavu protokolů. Akcelerátor je složen z několika částí, jak je vidět na obrázku 4.5. *PME* nejčastěji slouží k implementaci aplikačních filtrů.

Ovladač *PME* pro předchozí verze hardware obsahuje nástroj *fsl-pme*. Tento nástroj slouží k nastavování akcí pro jednotlivé regulární výrazy a k celkové konfiguraci akcelerátoru. Součástí balíčku je i kompilátor regulárních výrazů. Pro předešlé verze hardware je tento software sice dodáván i se zdrojovými kódy, ale klíčové knihovny jsou již zkompilované a to pro architekturu PowerPC, současně je software licencován pod licencí, která neumožňuje vytvořit si vlastní port, i kdyby to bylo možné.

Pravidla jsou specifikována ve dvou souborech. První obsahuje regulární výrazy a jejich identifikátory. Druhy obsahuje stavová pravidla. Ukázka formátu zadávání stavových pravidel je uvedena v 4.3.

```
STATEFUL_RULE: waitTime
  RESET_STATE:
    EVENT "invite"
      SRV[1] = 0
      SRV[2] = 0
      next_state ringing

  STATE ringing:
    EVENT "ring"
      SRV[1] = SRV[1] + 1

    EVENT "accept"
      report {
        0xa0000001
        SRV[1:2]
      }
      next_state RESET_STATE
```

Algoritmus 4.3: Ukázka definice stavových pravidel pro PME.



Obrázek 4.5: Schéma PME.

4.1.6 SEC - Security Engine Complex

SEC je komplex více hardwarových akcelerátorů pro protokoly zabezpečení, výpočet hešovacích funkcí a podobně. Jedná se již o 5. generaci tohoto akcelerátoru. Nejdůležitější částí tohoto komplexu je *CAAM (Cryptographic Acceleration and Assurance Module)*. [17]

Následuje výčet podporovaných funkcí a protokolů, které akcelerátor podporuje.

- AES (128, 192, 256-bit), EIA, 23.8Gbps AES-256
- DES/3DES, 14.3Gbps 3DES
- SHA-1,2,256,384,512 digest, 34.1Gbps SHA-256
- MD5 128-bit digest

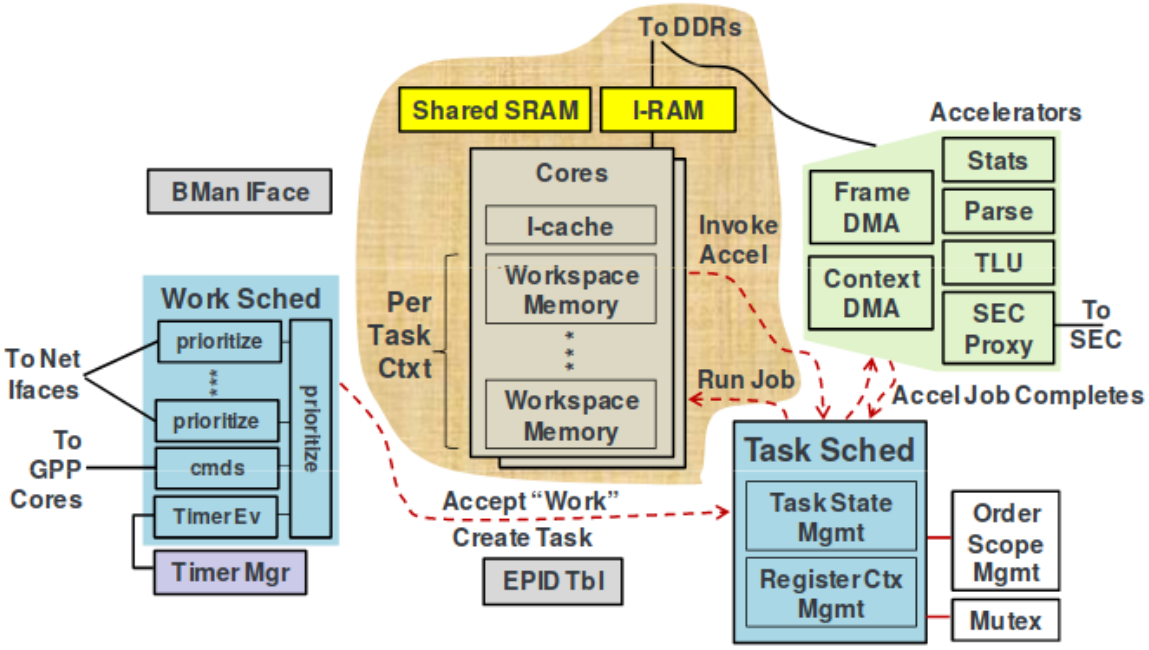
- off-load hlaviček pro IPSec, SSL/TLS, 3G RLC, PDCP, SRTP, Wifi, MACSEC, 23.8 Gbps IPSec (AES-HMAC-SHA-2)
- NIST certifikovaný generátor náhodných čísel
- akcelerace pro SNOW, ZUC, Kasumi pro sítě 3G
- CRC32, CRC32C

Akcelerátor je možno používat z linuxového prostředí explicitně jako další endpoint na *DPAA2*. Ovladač akcelerátoru však umožňuje i přístup na vyšších úrovních. Pokud kterákoliv aplikace na běžící na platformě použije například *IPSec*, akcelerátor je využit automaticky a generátor náhodných čísel tohoto akcelerátoru je použit jako výchozí v Linuxu. Pro platformu je dodávána modifikovaná knihovna *OpenSSL*, která využívá tento akcelerátor.

4.1.7 AIOP - Advanced IO Processors

AIOP je pole procesorů, spolu s dalšími akcelerátory Toto pole obsahuje 16 PowerPC e200 procesorových jader taktovaných na $800MHz$. Každé jádro je schopno zpracovávat až 16 vláken. Celkově je tedy možné mít až 256 běžících vláken. Jádra disponují hardwarovým plánovačem úloh. Přepínání kontextů je vysoce optimalizované a má jen minimální dopad na výkon. Procesorové pole je programovatelné v jazyce C, avšak s použitím knihovny dodávané výrobcem.

Zjednodušené schéma zobrazeno na obrázku 4.6.



Obrázek 4.6: Schéma AIOP (převzato z [15]).

AIOP obsahuje i další akcelerátory pro klasifikaci, statistiky, přesné časovače, farmu DMA a krypto engine. Pole procesorů může taktéž dostat přístup ke všem ostatním akcelerátorům. Architektura je vysoce optimalizovaná pro práci s pakety a synchronní programovací model umožňuje dosáhnout deterministického výkonu.

Pro jednoduché aplikace zvládne zpracovat až 30 MPkt/s, pro složitější aplikace jako *SEC* zvládne 17 MPkt/s.

Zkompilovat aplikaci pro toto pole procesoru je v době psání této práce možné pomocí *CodeWarrior Development Studio for Advanced Packet Processing*, což je IDE, které vyvíjí NXP. Součástí IDE je i simulátor.

AIOP může využívat několika pamětí, kromě *DDR* připojení k *DPAA2* má přístup také do prostoru *PEB* bufferů a má i speciální *ShRAM* na čipu, která je taktéž optimalizovaná na rychlost zápisu a čtení na úkor velikosti.

Pro ovládání *AIOP* z linuxového prostředí slouží nástroj *gpp-aioptool*, který je součástí softwarového balíku dodávaného výrobcem. Nástroj pracuje pouze na sběrnici *DPAA2*, která používá *VFIO* ovladač.

Spustit program na AIOP je možné příkazem uvedeným v algoritmu 4.4

```
1 aiop_tool load -g <container name> -f <path to file> -d -v
```

Algoritmus 4.4: Příkaz pro spuštění programu na AIOP.

Tato sekce vznikla jako výtah z dokumentů [14], [12] a [13].

4.1.8 DPAA2 objekty

DPAA2 objekty jsou abstrakcí nad hardwarem *DPAA2*. K alokaci těchto objektů slouží nástroj *restool* popsaný v sekci 4.2 anebo je možné objekty naalokovat staticky pomocí *DPL* firmwaru z u-boot jak je popsáno v sekci 4.3.

Názvy většiny objektů začínají písmeny DP, tato zkratka znamená Data Path.

Aplikace běžící na *GPP* mají přístup pouze k MC a QBMan portálům. Firmware *MC*, konfigurační registry a datové spoje uvnitř *DPAA2* jsou procesoru skryty. Software běžící na *GPP* a *AIOP* přistupuje k hardware pouze prostřednictvím dynamicky rezervovaných rozhraní, která reprezentují rozhraní *DPAA2* objektů.

DPRC - Resource Container je kontejner objektů na *DPAA2* sběrnici. Tento kontejner se jeví v OS jako samostatná sběrnice a podobně jako *PCI-E* je *plug-and-play*. *DPRC* si uchovává informace o přiřazených objektech a je možné zobrazit seznam těchto objektů například pomocí nástroje *restool* popsaného v sekci 4.2. Pomocí kterého je možné s objekty dynamicky manipulovat. *DPRC* má *IOMMU* mapovatelný region podobně jako *MC*, pro zadávání příkazů. Při připojení zařízení a dalších událostech, vysílá přerušení na hlavní procesor. Díky *IOMMU* je možno přímo přistupovat k hardwarovým prostředkům akcelerátorů z uživatelského procesu. Tento přístup je tímto způsobem také zabezpečen. Software nemůže přistupovat k akcelerátorům, které si nejsou alokovány, protože nemají nastavené mapování v *IOMMU*. To mimo jiné znamená, že aplikace si nemohou vzájemně ničit konfiguraci akcelerátorů a akcelerátory mezi kontejnery jsou naprosto oddělené, i když se jedná o jeden fyzický hardware. Každý akcelerátor totiž má určitý počet bran, které tvoří jistý kontext konfigurace.

Objekty *DPRC* lze libovolně zanořovat, při dealokaci tohoto objektu je automaticky uvolněn i jeho obsah.

DPNI - Network Interface obsahuje *RX*/*TX* fronty, konfigurační rozhraní síťového rozhraní a RX bufferů. Při alokaci tohoto objektu lze specifikovat i typ a parametry použitých bufferů a způsob rozdělování toků do jednotlivých front. Rozhraní tohoto objektu tvoří fronty směrem do software, konfigurační rozhraní a jeden datový port. Pro software

se tento objekt jeví jako síťové rozhraní. Dostupný je linuxový ovladač a ovladač založený na *VFIO*.

DPMAC - Ethernet MAC reprezentuje ethernetový *MAC* na sběrnici *DPAA2*, který je komponenta pro komunikaci pomocí protokolu *Ethernet*. Tato komponenta podporuje jen základní příkazy jako je zapnout/vypnout a nastavit přerušení. Rozhraní tohoto objektu tvoří jeden konfigurační port a jeden datový port. V software se tento objekt jeví jako ethernetový rozhraní a lze jej ovládat například pomocí linuxového nástroje *ip* nebo *ifconfig*.

DPSW - L2 switch je hardwarový *L2 switch* s N porty. Tento objekt je po hardwarové stránce složen z částí *WROIP* a *QMan*. Rozhraní tohoto objektu tvoří specifikovaný počet datových portů a konfigurační rozhraní. Tento switch má ovladač v softwarovém balíku, který dodává výrobce. Tento ovladač umožňuje ovládat tento switch pomocí standardních linuxových nástrojů jako je například *brctl* a *ip*. Tento ovladač však nefunguje s *VFIO* a tedy není pohodlně použitelný s DPDK, které je v této práci použito. Tento problém však lze obejít alokováním switche v samostatném *DPRC* kontejneru.

DPDMUX - Demultiplexor je komponenta, která umožňuje rozdělit jeden vstupní tok na více výstupních. Kromě těchto rozhraní demultiplexor obsahuje i konfigurační rozhraní. Tento objekt je zjednodušenou formou switche. Pro tuto komponentu je dodáván ovladač s podporou *EVB (Edge Virtual Bridge)*. Tento objekt je s použitím *EVB* ovladače konfigurovat standardními linuxovými nástroji jako je nástroj *ip* a *bridge*.

DPLAG - Link Agregator je objekt který slouží ke slučování několika vstupních vstupních spojů do jediného. Při slučování je možno uplatňovat policy a další modifikace síťových toků.

DPCON - Connection slouží k přeposílání potvrzení o přijetí paketu protějškem, obvykle pro každé CPU zvlášť. Na procesoru *NXP LS2088* lze alokovat až 512 těchto spojů. Rozhraní tohoto objektu je tvořeno výstupním kanálem pro *DPIO* a dvěma nebo osmi sloty pro *DPIO* fronty.

DPIO - I/O je komponenta, která zastřešuje kanál *DMA*. Umožňuje software číst a zapisovat z RX/TX front. Jedná se tedy o akcelerátor pro přístup k daným frontám. Rozhraní tohoto objektu tvoří konfigurační a notifikační rozhraní. Objekt navíc obsahuje rozhraní pro komunikaci s *GPP*. Tento objekt je specifický tím, že software s ním komunikuje na přímo, ke zbylým akcelerátorům je možné přistupovat pouze prostřednictvím *MC* portálů.

DPBP - Buffer Pool reprezentuje hardwarový pool bufferů v QBMan akcelerátoru. Na sběrnici *DPAA2* tento objekt zastřešuje dvě implementace paměťového bufferu. *PEB* je buffer přímo na čipu (na LS2088 celkem 4MB). Druhá možnost je implementace v externí DDR (na LS2088 $6.4GB/s$ propustnosti sdíleno s *AIOP*). Ve většině případů je aktivně používán jen při startu ovladače, kde jsou z něho alokovány paměťové buffery, ale při nedostatku paměti vysílá i přerušení na procesor.

DPMCP - Management Complex Port je objekt, který rezervuje konfigurační rozhraní na *MC* akcelerátoru. Tento objekt je potřeba naalokovat téměř pro všechny objekty, aby je bylo možné ovládat.

DPAIOP - Advanced I/O Processor zastřešuje *AIOP* akcelerátor. *AIOP* je pole procesorů popsané v podsekci 4.1.7. S tímto objektem je spojen objekt *DPCI*. *DPCI - Communication Interface* je objekt, který zprostředkovává komunikaci mezi *GPP* a *AIOP*. Slouží zejména ke konfiguraci a předávání kontrolních informací.

DPSECI - Security engine interface je Objekt, který reprezentuje jeden datový kanál šifrovacího akcelerátoru.

DPDCEI - Data Compression Engine Interface je Objekt, který reprezentuje jeden datový kanál komprimačního akcelerátoru.

DPRTC - Data Path Real Time Controller je Objekt pro řízení přesných časovačů na *DPAA2*.

Další vývoj objektů DPAA2 - Dle tvrzení NXP [15] pravděpodobně přibudou i další objekty postupně jak budou zprovozňovány další části *DPAA2*.

4.1.9 DPAA2 možnosti propojení a konfigurace

DPAA2 sběrnice umožňuje dané objekty téměř libovolně zapojit. Je však nutno dodržet závislosti komponent a řadu jejich omezení. Např. síťové rozhraní DPNI má závislosti vypsány v tabulce 4.2.

Objekt	**Privátní pro DPNI**	**Závislosti**
DPBP	Ano	1 pro DPNI
DPMCP	Ano	1 pro DPNI, 1 pro DPMAC
DPCON	Ano	1 pro každou frontu na každém DPNI do každého GPP
DPIO	Ne	1 pro každý GPP
DPMAC	Ne	*N* - specifikovano uživatelem dle zapojení

Tabulka 4.2: Závislosti DPNI objektu.

DPNI musí mít připojen vstup na jeden z následujících objektů: *DPNI*, *DPMAC*, *DPSW*, *DPMUX*, *DPLAG*. Dále musí být splněny závislosti dle tabulky 4.2.

Každý objekt na DPAA2 sběrnici má velké množství konfigurovatelných parametrů. Tyto parametry lze konfigurovat různými způsoby (Device Tree, skripty v softwarovém balíku QorIQ/DPDK, ...) následující text předpokládá použití nástroje *restool* popsaného v sekci 4.2.

Následující přehled obsahuje ty nejdůležitější parametry klíčových komponent.

Pro síťové rozhraní *DPNI* je pomocí *restool*, popsaného v sekci 4.2, možné nastavit následující parametry:

1. MAC adresu,
2. TX shaping,

3. počet front - pro přijímání/odesílání paketů,

4. počet počet položek v tabulce tříd provozu/QoS/VLAN - Tyto parametry slouží k nastavení rozdělování paketů do jednotlivých front.

Po připojení DPDK aplikace na tento port je možné nastavit *RSS (Receive Side Scaling)* funkci, která řídí rozdělování příchozích paketů do jednotlivých front.

Pro L2 switch *DPSW* je pomocí *restool*, popsaného v sekci 4.2, možno nastavit:

1. Počet portů,

2. způsob učení MAC adres,

3. velikost tabulky pro VLAN,

4. rychlost stárnutí záznamů v tabulkách.

Softwarový balík obsahuje ovladač *fsl-dpaa2/ethsw*, který umožňuje ovládat switch standardními linuxovými nástroji jako je např. *brctl*.

4.1.10 DPAA2 SW

Procesory NXP rodiny *Layer Scape* využívají softwarovou platformu *QorIQ*. Softwarový balík pro tento procesor je distribuován v dvou různých formách. Jako předkonfigurovaný obraz systému postavený na *Ubuntu Linux* pod názvem *LSSDK*. Anebo ve formě vrstvy pro buildovací systém *Yocto* jako *Yocto SDK*. Oba softwarové balíky obsahují nakonfigurovaný Linux, *U-Boot* a ovladače a další knihovny jako například *DPDK*. Pro účely *Sprobe* je vhodnější použít *Yocto SDK*, protože se jedná o novou platformu a je potřeba změnit *U-boot* i části Linuxu a binární firmwary v obraze systému. Pro tento účel je *Yocto SDK* přímo vytvořeno. Pro sběrnici *DPAA2* je v software balíku od NXP dodáván ovladač *mc-bus*. Tento ovladač po přidání do kernelu zaregistruje sběrnici *fsl-mc* a zaregistruje *call-back* funkce. *mc-bus* implementuje API pro DPAA2 ovladače a manipulaci s DPAA2 objekty.

Části linuxového ovladače:

- Ovladač *DPRC* (dprc-driver) je napojen na *DPRC* objekty a za běhu spravuje DPAA2 sběrnici, zapojuje/odpojuje zařízení na ní.

- *Allocator*, objekty jako *DPMCP* 4.1.8 a *DPBP* je možné zobecnit a vzájemně nahrazovat, pro tento účel slouží tento ovladač.

- *DPIO driver* je napojen na *DPIO* objektů. Poskytuje API pro příjem a odesílání ethernetových rámců. Mezi klíčové funkce patří signalizace dostupnosti, operace nad hardwarovými *RX*/*TX* frontami a management hardwarového poolu bufferů.

  Tento ovladač registruje ethernetové, kryptografické, kompresní a podobné zařízení dle konfigurace.

- *DPAA2 Ethernet* je ovladač, který v Linuxu reprezentuje *DPNI* jako ethernetové zařízení a umožňuje jeho napojení na linuxový *network statck*. Tento ovladač se omezuje na použití jediné fronty na *DPNI* a jeho výkon pro malé pakety je výrazně omezen.

- *DPAA2 MAC driver* je ovladač pro řízení ethernetového *MAC*, jedná se o proxy, která zviditelní *MDIO* sběrnici připojenou na *MC* ostatním ovladačům v systému, které jej pak můžou konfigurovat. Tedy vlastni ovladač filtru je další samostatný ovladač.

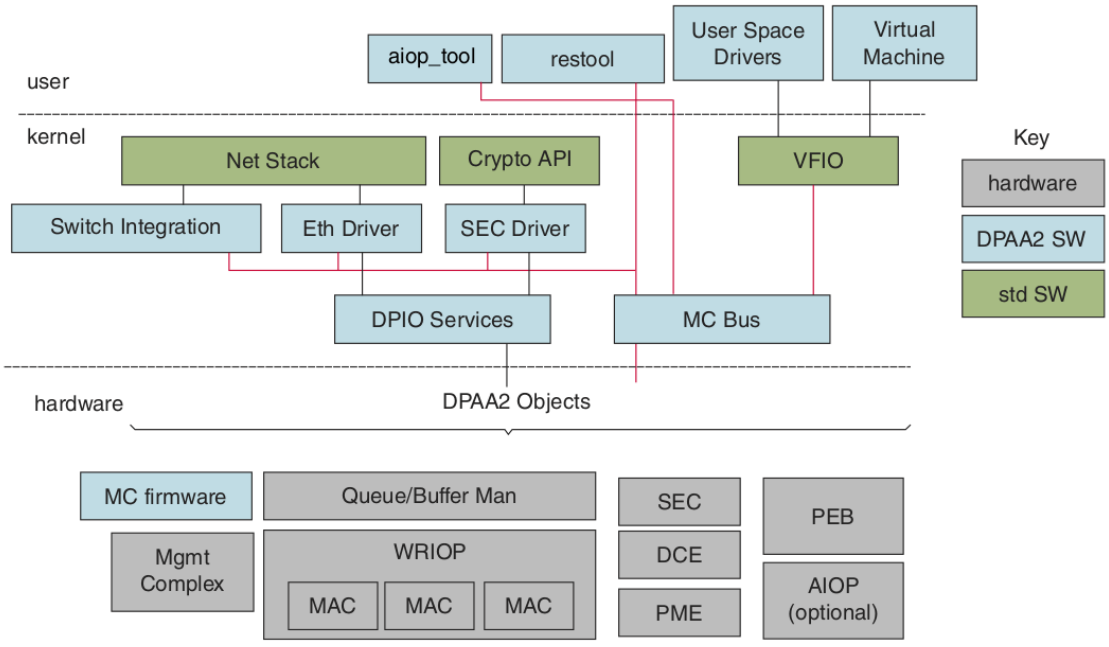
Linuxové nástroje

- *restool* - Nástroj pro dynamickou alokaci a základní konfiguraci objektů na *DPAA2*. Popsáno v sekci 4.2.
- *bridge* - Standardní linuxový nástroj pro konfiguraci síťových mostů [7].
- *ip* - Standardní linuxový nástroj pro konfiguraci síťových rozhraní. Pomocí něj je možné konfigurovat síťové rozhraní na *DPAA2*, které jsou naalokovány pro Linux. Případně i porty *DPSW*, *DPDMUX* a podobných.

Příkazy v U-boot

- *fsl_mc start mc [FW_addr] [DPC_addr]* - zapnutí *MC*
- *fsl_mc apply DPL [DPL_addr]* - konfigurace *DPAA2* podle DPL souboru
- *fsl_mc lazyapply DPL [DPL_addr]* - pozdní konfigurace *DPAA2* podle DPL souboru
- *fsl_mc start aiop [FW_addr]* - zapnutí *AIOP*

Switch, kryptografický a komprimační akcelerátor mají vlastní ovladač. Pro síťová rozhraní jsou dostupné dva ovladače. První pro standardní linuxový a druhý s použití *VFIO*, který používá DPDK. Všechny ovladače jsou napojeny na *MC Bus*, který reprezentuje konfigurační sběrnici *DPAA2*. Datové přenosy obstarávají *DPIO services*, které ovládají přímo DMA v *DPIO* objektech na *DPAA2* sběrnici. Celkový pohled je zobrazen na obrázku 4.7.



Obrázek 4.7: Schéma softwarových vrstev *DPAA2* na *LS2088* (převzato z [9]).

4.2 Nástroj restool

Nástroj restool slouží ke konfiguraci *DPPA2 sběrnice* z linuxového prostředí je možno použít nástroj *restool*. Tento nástroj umožňuje alokaci, propojování a základní konfiguraci všech objektů na sběrnici. Nástroj navíc dokáže generovat statickou konfiguraci pro sběrnici *DPAA2*. Konfigurace pomocí tohoto nástroje je v této práci vysvětlena v sekcích popisujících jednotlivé objekty na *DPAA2*.

S pomocí tohoto nástroje je také možné zobrazit aktuální konfiguraci anebo uložit aktuální konfiguraci do DTS souboru a vyrobit z něj statickou konfiguraci *DPAA2*, kterou je možné ze zavaděče nahrát před startem systému. Příkaz pro získání statické konfigurace *DPAA2* je napsán v kódu 4.5. Nástroj *dtc* je linuxový device-tree kompilátor. Tento formát je nezávislý na platformě a lze tedy použít jakoukoli verzi tohoto nástroje na jakékoli platformě. Takto vygenerovaná konfigurace *DPAA2* je otisk aktuální konfigurace, je tedy nutné nejprve platformu nakonfigurovat do požadované konfigurace.

```
1 restool dprc generate-dpl > dppa2_config.dts
dtc -O dtb -o dppa2_config.dtb dppa2_config.dts
```

Algoritmus 4.5: Příkazy pro vygenerování statické konfigurace DPAA2.

4.3 Firmware procesoru

Firmware procesoru obsahuje kromě kernelu, file-systému a device-tree i řadu dalších datových bloků:

- *Reset Configuration Word/Pre-boot Loader (RCW/PBL)*
- *Data path configuration file (DPC)* - obsahuje nastavení *MC (Management Complex firmware)*, defaultní MAC adresy a další defaultní hodnoty pro objekty na *DPAA2* sběrnici.
- *DPL (Data path layout file)* - obsahuje prvotní zapojení DPAA2 sběrnice
- *U-Boot* - second stage bootloader
- *PPA (Primary Protected Application)* - Speciální firmware běžící v ARMv8A-EL3 privilegovaném režimu. Řídí mimo jiné power management a spouští další jádra CPU.
- *Device Tree* - Soubor obsahující informace o platformě. Stejná struktura jako u většiny procesorů rodiny ARM [3].
- kernel - Jádro operačního systému.
- file system

Výchozí adresy, na kterých lze části firmware najít v paměti FLASH a SD kartě jsou vypsány v tabulce 4.4. LS2088 používá rozdělení FLASH na jednotlivé banky (adresy v tabulce 4.3), kde firmware může být nahrán do některé z nich. FLASH paměť muže na LS2088 mít až 16 bank, přepínání mezi nimi je možné prostřednictvím pinů procesoru. RCW/PBI a *second-stage bootloader* jsou jediné dvě nepřesunutelné položky. Všechny ostatní jsou načítány U-Bootem v rámci bootu a jejich adresy jsou uloženy v jeho konfiguraci (environment).

Banka	Velikost	Adresa
VBank0/Bank 0	64MB	0x580000000
VBank1/Bank 1	64MB	0x584000000

Tabulka 4.3: Adresy bank paměti FLASH [15].

Položka	Max velikost	FLASH offset	SD Card start block
RCW+PBI	1MB	0x00000000	0x00008
Boot firmware (U-Boot, UEFI)	2MB	0x00100000	0x00800
Boot firmware Environment	1MB	0x00300000	0x01800
PPA firmware	2MB	0x00400000	0x02000
Secure boot headers	3MB	0x00600000	0x03000
DPAA1 FMAN ucode	256KB	0x00900000	0x04800
QE / uQE firmware	256KB	0x00940000	0x04A00
Ethernet PHY firmware	256KB	0x00980000	0x04C00
Scripts	256KB	0x009C0000	0x04E00
DPAA2 MC	3MB	0x00A00000	0x05000
DPAA2 DPL	1MB	0x00D00000	0x06800
DPAA2 DPC	1MB	0x00E00000	0x07000
Device tree	1MB	0x00F00000	0x07800
kernel	16MB	0x01000000	0x08000
ramdisk rootfs	32MB	0x01100000	0x08800

Tabulka 4.4: Výchozí adresy částí firmware v paměti FLASH [15].

Po zavedení *Second-stage bootloaderu* je již možné bootovat i z ostatních medii jako *SATA*, *TFTP* anebo *USB memory device*[15].

Veškerý firmware pro DPAA2 je uzavřený kód a je distribuován jako binární soubor v softwarovém balíku do výrobce.

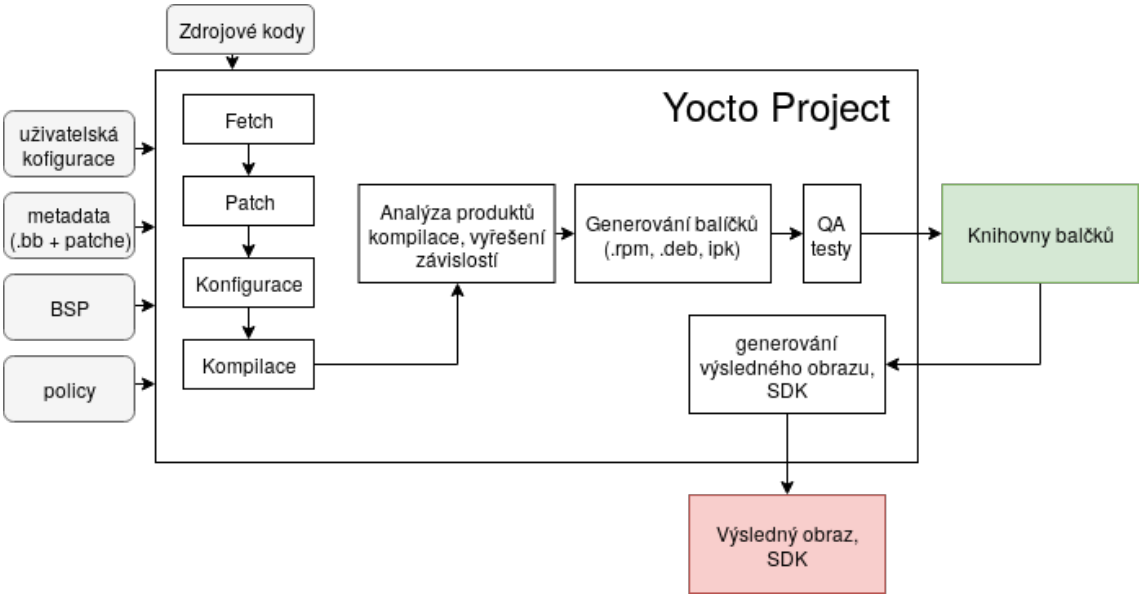
RCW a *PBI* lze vygenerovat pomocí *CodeWarrior Network Applications*. Toto IDE[6] je rovněž vyvíjeno společností NXP. Zavaděč *U-boot*, Linux, *MC firmware* a *PPA* firmware je součástí *Yocto SDK*. Soubory DPL a DPC je možné vygenerovat s pomocí nástroje *restool* popsaného v sekci 4.2

Yocto SDK Softwarový balík k procesoru je mimo jiné dodáván i jako *vrstva* pro buildovací systém *Yocto*. *Vrstva* je forma skupiny balíčků.

Základní části buildovacího systému *Yocto* jsou zobrazeny na schématu 4.8. Hlavním vstupem jsou Yocto balíčky tzv. recepty (recipes) soubory s koncovkou *.bb* a jejich modifikace uložené v souborech s koncovkou *.bbappend*. V těchto balíčcích jsou uloženy informace o zdrojových souborech a o operacích, které mají být provedeny. Speciálním balíčkem je *BSP*[7]. Tento balíček obsahuje základní návody pro sestavení obrazu pro cílovou platformu. Tyto vstupy jsou ve schématu vybarveny šedě. *Yocto* je založeno na balíčcích, na schéma je

[6]Integrated Development Environment

[7]Board Support Package



Obrázek 4.8: Schéma buildovacího systému Yocto.

znázorněn základní životní cyklus balíčku, který začíná stáhnutím potřebných zdrojových souborů tzv. *fetch*, dále pokračuje aplikováním patchů konfigurací a kompilací. Balíčky jsou následně zabaleny do své obvyklé formy, dle druhu cílového Linuxu a jsou prováděny kontroly kompatibility. Následně je z balíčků seskládán výsledný *obraz*.

Problematika buildovacího systému *Yocto* je rozsáhlá a mimo zaměření této práce. Bližší informace k *Yocto* je možné nalézt v knize [5] nebo v oficiální dokumentaci [19]. Výsledný software této práce je ve formě další vrstvy pro tento buildovací systém.

Výrobce procesoru NXP dodává *BSP* a základní balíčky s Linuxem, ovladači a dalšími knihovnami a nástroji. Tyto balíčky dohromady tvoří softwarovou platformu *NXP QorIQ*, která je dodávána ve formě vrstvy pro tento buildovací systém pojmenovaná jako *Yocto SDK*. Tato platforma obsahuje software pro síťové procesory od NXP.

Výsledkem buildování *Yocto* je kompletní obraz systému pro procesor s předinstalovanými aplikaci. Obraz systému také obsahuje zavaděč a další firmware.

4.4 Bootovací proces

Bootovací proces LS2088 je obdobný jako u ostatních ARMv8a SoC. Navíc však obsahuje nastavování *DPAA2* sběrnice a nastavení sériových linek.

Výběr bootovacího media je zajištěn pomocí konfigurace na dedikovaných pinech procesoru. Bootovací proces má následující fáze:

1. Po restartu SoC se spustí *first-stage bootloader*, který je uložen na čipu. Načte RCW/-PBL a na jeho základě se provede nízko úrovňová konfigurace SoC.

2. *First-stage bootloader* načte *second-stage bootloader* (*U-Boot* nebo *UEFI*) a předámu řízení.

3. *Second-stage bootloader* načte PPA a ostatní firmware zařízení a inicializuje je.

4. *Second-stage bootloader* načte *device tree* a doplní do něj informace o *DPAA2*, načte kernel a předá mu řízení.

Procesor umožňuje i další způsoby bootování, jejich popis je např. v dokumentu[15].

Kapitola 5

Portace software na novou platformu Sprobe10g

Portace probíhala v několika fázích. V průběhu portace se průběžně měnil softwarový balík dodávaný výrobcem procesoru. Vývoj tedy neprobíhal zcela lineárně. Se zanedbáním nepodstatných detailů lze portaci shrnout v následujícím odstavci.

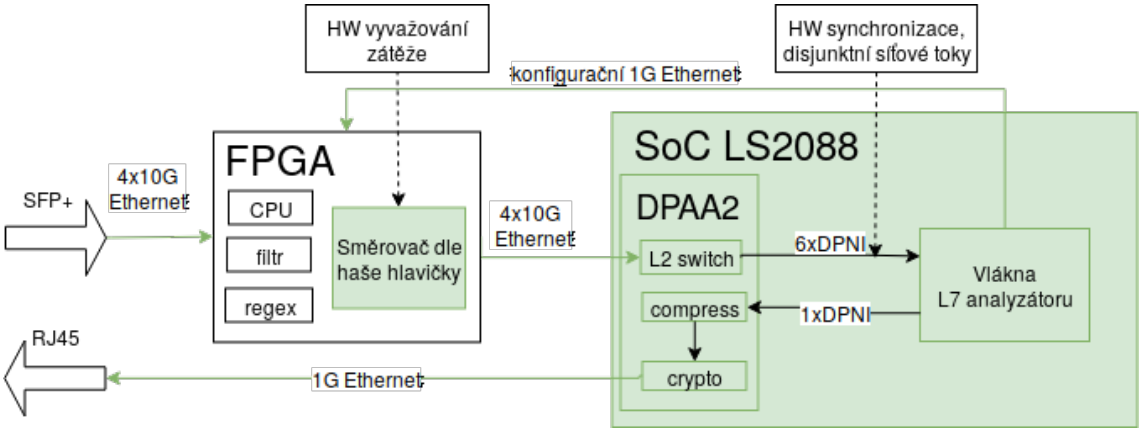
V první části portace byly přemostěny přístupy k hardware přes síťový protokol *TCP*. Následně vznikla naivní portace aplikace *PaSt* na *x86_64*, která využívala knihovny *DPDK* místo původního frameworku pro kopírování dat mezi hardwarem a softwarem (*RSoC framework*). V další fázi probíhalo oživování nové platformy a prvotní experimenty s *DPAA2*. Po částečném zprovoznění *DPAA2* byl vytvořen pokročilejší port programu *PaSt* s podporou procesoru *NXP LS2088* (popsáno v sekci 5.3 a 5.4). Dále byl naportován zbylý software a platforma byla usilovně testována a opravována. Výsledkem je vrstva pro buildovací systém *Yocto*, která obsahuje mimo jiné knihovny a aplikace pro obsluhu hardware v FPGA a program *PaSt*. Vznikl také zcela nový firmware FPGA, ten ale až na výjimky není předmětem této práce. Port software tedy zahrnoval i podporu pro změněné a nově přidané akcelerátory v FPGA i pro některé akcelerátory v *SoC NXP*.

Konfigurace nově vzniklé hardwarové platformy se ustálila ve stavu, který zachycuje schéma na obrázku 5.1. Zeleně jsou vyznačeny části, které na staré platformě nemají obdobu. Konfigurace akcelerátorů v FPGA je nově prováděna přes konfigurační $1GE$ spoj. Konfigurace hardware je prováděna za pomocí knihovny *HWIO* popsané v sekci 3.3. Firmware hardware je blíže popsán v podsekci 5.2. Zapojení sběrnice *DPAA2* je blíže popsáno v sekci 5.4 společně s paralelizací aplikace *PaSt*.

5.1 Analýza složitosti programu PaSt

Tato sekce se zabývá rozborem programu PaSt po stránce teoretické složitosti i praktické výpočetní náročnosti. Cyklomatická[1] složitost programu *PaSt* je vysoká. Vysoký je taktéž počet úrovní abstrakce. Z tohoto důvodu jsou některé popisy dědičností zkráceny a pseudokódy výrazně zjednodušeny, aby bylo možné pozorovat hlavní kód. Fakta zmíněná v této sekci také platí pro nově vytvořenou (*Sprobe10G*) i starou verzi (*Sprobe1G*) tohoto software, není-li uvedeno jinak.

[1] Cyklomatická neboli podmínková složitost je číslo vyjadřující složitost programu. Měří počet možných cest skrz zdrojový kód.



Obrázek 5.1: Hlavní datové cesty v Sprobe10g.

Aplikace *PaSt* po spuštění provádí inicializace hardwaru. Tyto inicializace trvají zanedbatelnou dobu. Nově nejdelší čas při startu zabírá inicializace *EAL* z knihovny *DPDK*. Tento čas je však stále kratší než 3 sekundy a vzhledem k použití této aplikace můžeme inicializaci a její složitost zcela zanedbat.

Po inicializaci program přechází do hlavní smyčky, ve které neustále čte pakety ze vstupních rozhraní a zpracovává je.

Stará implementace využívala specifický *zero-copy* protokol pro přenos paketů do software. Nově je použito rozhraní DPDK. Každý příchozí paket nebyl v době zpracování v paměti cache procesoru a způsoboval cahe miss. Zpracování paketu je však složitý kód. Díky tomu přednačtení (pre-fetch) následujících paketů do paměťi cache nemělo velký význam, protože na staré platformě byla paměť cache velmi malá. Na nové platformě tyto přednačtení provádí ovladač. Jejich vliv je však sporadický a závisí na charakteru provozu i aktuálním zatížení aplikace.

Zpracování paketu v hlavní smyčce spočívá v provedení kroku stavového automatu pro daný síťový tok a otestování pole příznaků generované akcelerátory v FPGA. Stav síťového toku je uložen v tabulce toků.

Hardwarem nastavované příznaky značí, že paket s vysokou pravděpodobností obsahuje dané regulární výrazy, které se vyhledávají v aplikačních datech. V rámci hlavní smyčky jsou zkontrolovány výskyty regulárních výrazů v paketu. Tento úkol provádí aplikační parsery. S pomocí nově vygenerovaných příznaků po jednotlivé regulární výrazy a metadaty je proveden další krok stavového automatu pro daný síťový tok.

Hlavní tabulka toků Stavy síťových toků jsou uloženy ve struktuře, která se nazývá tabulka toků v kódu odpovídá třídě *flow_table*. Tato tabulka je složená z několika tabulek, které s slouží k ukládání hardwarového a softwarového stavů filtrů a aktuálního stavu síťových toků. Tento složitý komplex tabulek by bylo vhodné rozdělit s využitím hardwarových prostředků nové platformy. Zpětná kompatibilita je však požadována. Zjednodušená struktura této tabulky je zapsána v pseudokodu 5.1. Tabulka toků je složena z několika hešovacích tabulek. Firmware v FPGA, který PaSt používá obsahuje složený filtr. Tento složený filtr umožňuje filtrování pomocí trojic nebo pětic. Trojice mají tvar (IP adresa, UDP/TCP port, protokol) pětice navíc mají IP a port pro oba směry komunikace. Pro tyto různé identifikátory toků má tabulka toků dvě podtabulky (*pft_3tuple* a *pft_5tuple*). Tabulky jsou pak ve dvou kopiích, kde jedna je je určena pro používání v software a je

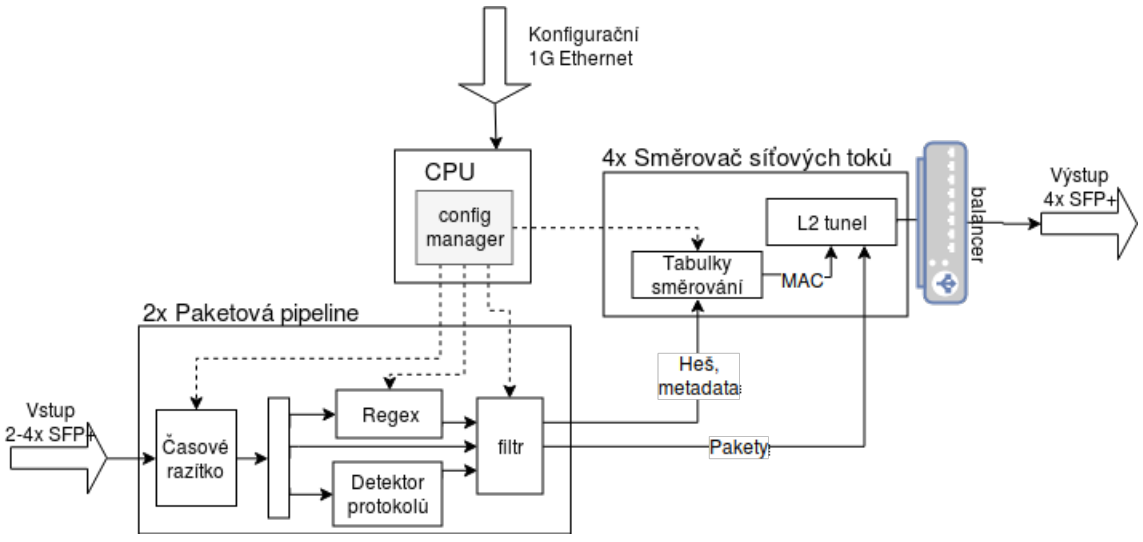
pojmenována *pft_sw*. Tato tabulka obsahuje aktuální stav všech toků, které se dostaly do software. Druhá tabulka *pft_hw* je obrazem aktuálního stavu hardware a slouží primárně k rozhodování o čistění záznamů v hardwarových filtrech v FPGA.

```
class flow_table {
  flow_table_coordinator pft_3tuple;
  flow_table_coordinator pft_5tuple;
};

class flow_table_coordinator {
  partial_flow_table pft_hw;
  partial_flow_table pft_sw;
};

class partial_flow_table {
  std::unordered_map<l4_tuple, flow_data, l4_tuple_hash> flow_map;
};
```

Algoritmus 5.1: Struktura tabulek toků.

5.2 Nový fimware FPGA

Firmware FPGA slouží k předfiltrování vstupních toků a částečně také k vyhledávání regulárních výrazů, rozdělení datových toků na jednotlivé výpočetní vlákna a vyvažování zátěže. Starý firmware *FPGA* je popsán v sekci 3.1. Zjednodušené schéma nového je znázorněno na obrázku 5.2. Firmware *FPGA* vznikl na základě požadavků této práce nikoli v rámci této práce, proto je zde pouze zmíněn.



Obrázek 5.2: Zjednodušené schéma FPGA firmware pro Sprobe10g.

Princip předfiltrování Při vstupu paketu do *FPGA* jsou z hlaviček paketu vyextrahovány metadata a je pro ně vyhledán záznam ve filtru. Pokud pro daný paket není záznam ve filtru, přepošle se do vlákna programu *PaSt* podle heše hlavičky. Vyhledávání ve filtru je invariantní ke směru paketu v datovém toku. Tím je zaručeno, že všechny pakety da-

ného síťového toku jsou klasifikovány stejným pravidlem. Dalším důsledkem je, že pakety síťového toku zpracovává vždy stejné výpočetní vlákno.

Pokud filtr obsahuje záznam, který odpovídá paketu, je provedena operace definovaná v záznamu filtru. Touto operací může být zahození a nebo přeposlání na konkrétní vlákno podle pravidla filtru.

Směrování a vyvažování zátěže je popsáno v sekci 5.4.

5.3 Portace programu PaSt

Cílem portace L7 analyzátoru *PaSt* je především dosažení řádově vyššího výkonu oproti staré platformě. Toho je dosaženo především použitím knihovny *DPDK* a paralelizací s hardwarovou synchronizací výpočetních vláken. Knihovna DPDK je popsána v sekci 3.2 a způsob paralelizace v sekci 5.4.

Původní aplikace popsána v sekci 3 byla pevně spojena se starou platformou. Pro síťovou komunikaci používala vlastní *zero-copy* protokol[2]. Pro konfiguraci hardware byl taktéž použit vlastní protokol popsaný v sekci 3.3. Stará verze byla striktně jedno-vláknová aplikace. Stará hlavní smyčka obsahovala bufferování paketů a manipulace s tabulkou toků způsobem, který byl neslučitelný s vícevláknovou implementací. Bylo také zapotřebí přemostit přístup k akcelerátorům v *FPGA*.

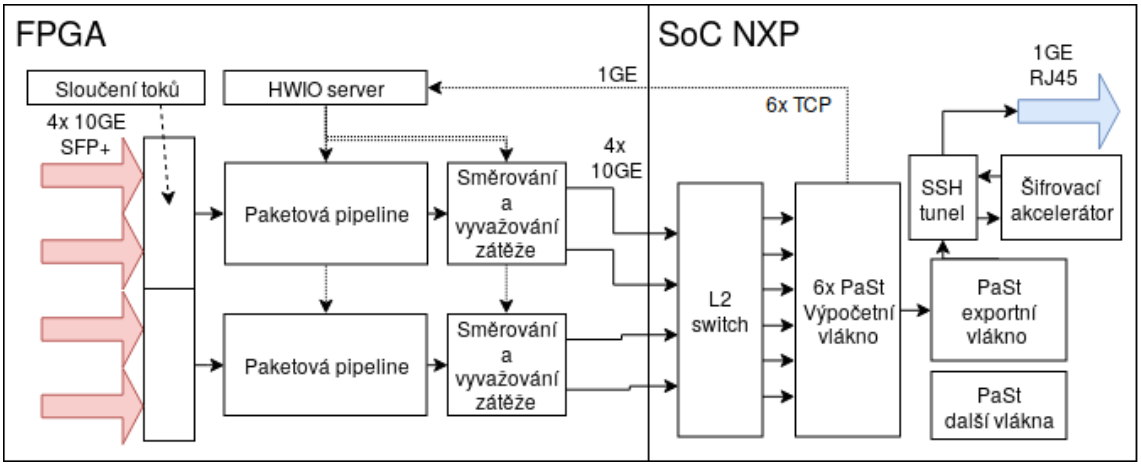
Portace probíhala v několika fázích. Jako první byla přepsána vrstva obstarávající konfiguraci hardware. Byla použita architektura server-klient, kde server běží na *SoC* v *FPGA* a deleguje čtení a zápis do hardwarových komponent v *FPGA*, více v sekci 3.3.

Síťová vrstva byla přepsána na standardní síťové API *DPDK*. Použitím této knihovny je na této platformě možné téměř dosáhnout rychlosti propustnosti linky. Avšak výkon je ovlivněn mnoha faktory viz sekce 5.9. To umožnilo, aby *PaSt* bylo možné zkompilovat pro libovolnou platformu s podporou knihovny *DPDK*. Kvůli podpoře *DPDK* musel být zcela změněn přístup ke správě paměti a paměťových bufferů pro datové toky. Změna zprávy paměťových bufferů byla již implementována v rámci přechodu na zero-copy protokol a změny v rámci této práce byly jen minoritní. Současně muselo být dočasně vyřešeno rozdělování zátěže v software. Tatko vznikl první test konceptu nové platformy.

Následně byla sestavena vrstva pro buildovací systém *Yocto*. Byla vytvořena konfigurace pro SoC NXP. Byl upraven firmware *FPGA* (mimo tuto práci). A tím vznikl první naivní port aplikace *PaSt* pro procesor NXP LS2088. V dalších fázích probíhalo ladění parametrů *DPAA2*, *DPDK* i samotné aplikace *PaSt*. Během těchto fází byla kromě výkonu velmi výrazně zvýšena i uživatelská přívětivost zejména po stránce konfigurace procesoru NXP. Vznikl nejen hlavní firmware ale i firmware pro ladění, který obsahuje většinu běžně používaných ladících nástrojů a obsahuje konfigurace, které umožňují pohodlné ladění aplikací na této platformě. Dále vznikly i návody jak s platformou zacházet.

Výsledkem je modifikovaná aplikace *PaSt*, která aktivně využívá kromě akcelerátorů v *FPGA* také akcelerátor *WROIP* v SoC NXP. Tento akcelerátor slouží mimo jiné k jednoduché klasifikaci a směrování paketů na virtuální rekonfigurovatelné síti uvnitř čipu. Nepřímo je použit i bezpečnostní komplex *SEC*. Akcelerátor vyhledávání regulárních výrazů *PME* by bylo velmi výhodné použít, ale v současné době pro něj neexistují ovladače a výrobce na nich začne pracovat nejdříve v Q3 2019, tedy za více než rok po odevzdání této práce. Kompresní akcelerátor je v době psaní této práce možné pouze demonstračně

[2] *RSoC framework*



Obrázek 5.3: Finální blokové schéma nové platformy.

spustit, navíc pouze z prostředí jádra systému. Problém je s nakonfigurováním IOMMU na této platformě a podpora pro tento akcelerátor je také na počátku vývoje. Proto akcelerátor nemůže být využit, ale jeho využití je připraveno.

Schéma na obrázku 5.3 zobrazuje výsledné blokové schéma nové platformy. Šifrovací akcelerátor je použit prostřednictvím *SSH* tunel, systém je na platformě nakonfigurován tak, aby jej použil automaticky. Paralelizace, použití L2 switche, a dalších prostředků *DPAA2* a FPGA je vysvětleno v sekci 5.4.

Nová implementace programu *PaSt* byla ověřena základními testy z již existující testovací sady, která byla upravena pro použití s *DPDK* a paralelní verzí nové aplikace.

5.4 Paralelizace programu PaSt

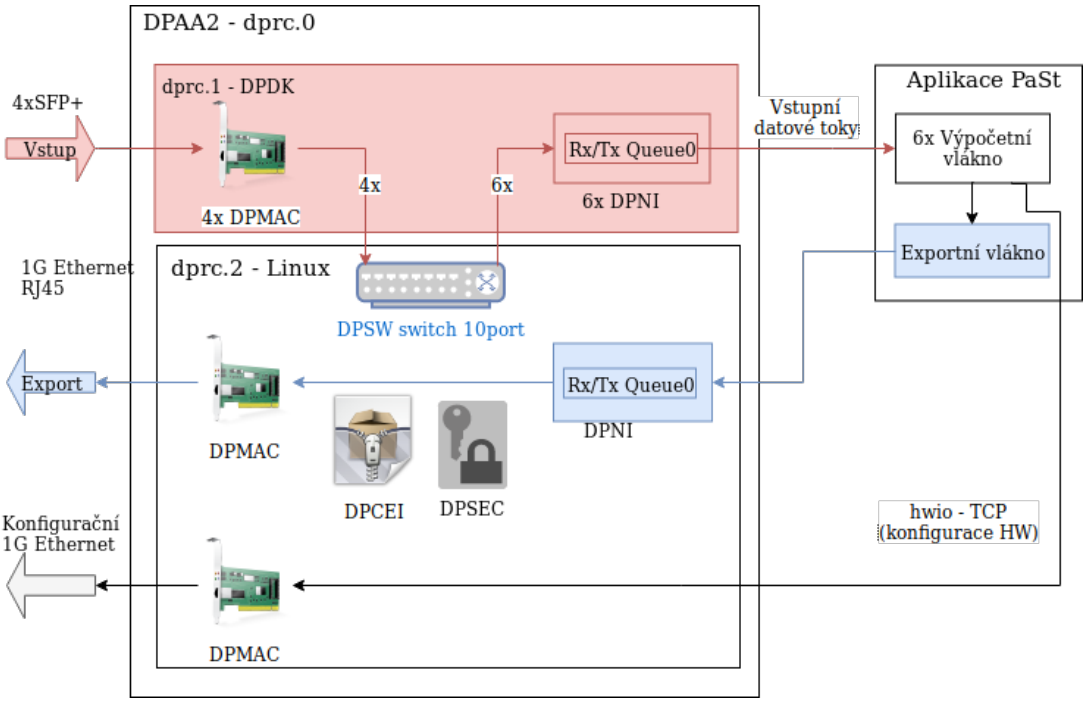
Aplikace *PaSt* obsahuje hlavní smyčku znázorněnou v pseudokódu 5.2. *Tabulka toků* musí být sdílena pro všechna vlákna. Toto je zásadní problém pro paralelizaci. Sdílení *tabulky toků* by velmi limito výkon paralelní implementace. Tento problém lze však obejít. Pokud by bylo zaručeno, že tabulky budou obsahovat pouze jiné záznamy nebyla by potřeba jejich synchronizace. Díky *FPGA* a 4.1 je možné rozdělit datové toky do téměř oddělených skupin. To umožňuje značně zrychlit paralelní implementaci tohoto software a vyřešit většinu synchronizačních problému na úrovni hardware. Problém však nastává u protokolů, které využívají více připojení v rámci jedné relace. Například protokol *FTP* používá komunikační a datový kanál, kde port datového kanálu je dynamický přidělován a informace o něm jsou předávány pomocí komunikačního kanálů. Proto filtr v hardware musí podporovat ještě možnost explicitně směrovat daný síťový tok na konkrétní výpočetní vlákno.

```
while True:
    paket = network.recv()
    meta = flow_table.get(packet.meta)

    for parser in parsers:
        if parser.packet_has_flag(packet, meta):
            parser.proces(packet, meta)

    if meta.export or meta.buffer or meta.drop or meta.filter:
        handle_flags(packet, meta)
```

Algoritmus 5.2: Hlavní smyčka PaSt (pseudokód).

S ohledem na výše uvedené požadavky byla navržena a implementována architektura zobrazená na obrázku 5.1. V *FPGA* je každému paketu předřazena další ethernetová hlavička. Cílová adresa je určena z heše hlaviček, které jednoznačně určují příslušnost k danému síťovému toku. Cílové výpočetní jádro pro daný heš je možné softwarově přetížit pomocí speciální položky v záznamu filtru.

V testech se nejlépe osvědčila konfigurace *DPAA2* znázorněná na obrázku 5.4 a na schéma 5.3 je znázorněno její použití v platformě jako celku. Podpůrné objekty (*DPBP*, *DPMC*, *DPCON*, *DPIO* a další) nejsou zakresleny pro zpřehlednění. Červeně jsou znázorněny cesty pro příjem paketů, modře exportní část. Šifrovací a komprimační akcelerátor jsou zde alokovány. Komprimační akcelerátor prozatím ve finální verzi není aktivní. Důvodem je nedokončená implementace ovladačů ze strany výrobce. Více o problémech akcelerátorů je možné nalézt na konci předešlé kapitoly 5.3. L2 switch *DPSW* je instanciován v kontejneru *dprc.2*, který je spravován čistě ovladači v Linuxu. To umožňuje použít linuxové ovladače i pro tento switch a konfigurovat ho pomocí běžných nástrojů v Linuxu. Stejně tak síťové rozhraní pro export a šifrovací akcelerátor je možné používat běžným způsobem. Naproti tomu kontejner *dprc.1* pro *DPDK* využívá ovladač postavený na *VFIO*. Tento ovladač umožňuje použití síťových rozhraní v *DPDK* a znemožňuje použít standardní ovladače.



Obrázek 5.4: Zjednodušené schéma konfigurace DPAA2 pro Sprobe10g

V procesoru jsou všechny vstupní ethernetové porty připojeny do L2 switche prostřednictvím *DPAA2* sběrnice. Tento L2 switch je staticky nakonfigurován tak, aby podle cílové MAC adresy dopravil paket na dané výpočetní vlákno aplikace *PaSt*. *DPAA2* umožňuje vytvořit až 63 virtuálních síťových portů *DPNI*. Pro každé vlákno aplikace *PaSt* je vytvořen samostatný port, který je připojen na hlavní L2 switch a ten svou přirozenou funkcí prování distribuci na jednotlivá vlákna aplikace *PaSt*. Výpočetních vláken je prakticky ome-

zen pouze na 6. Jádro 0 nelze použít kvůli častým přerušením a jádro jedna není výhodné využít.

Každé vlákno má vlastní připojení na konfigurační server hardware v FPGA a sdílení hardware je řešeno na úrovni tohoto serveru. Každé vlákno spravuje oddělené síťové toky a to znamená, že konfigurace nekolidují. Teoreticky však může docházet k problémům s konfigurací filtru při jeho přetížení. Tento problém bude vyřešen, až se změnou API knihovny pro přístup k hardware popsané v sekci 3.3.

Export dat je prostřednictvím bezzámkových front předáván na exportní vlákno a z něj je přes zabezpečeny tunel exportován na *mediační funkci*. Exportní vlákno používá samostatné síťové rozhraní připojené pouze na samostatný 1GE rozhraní.

Výkon této platformy je zhodnocen v sekci 5.10. Platforma jako celek dosahuje dobrého výkonu, ale do budoucna bude možné získat ještě další výkon použitím asynchronního přístupu k hardware a použitím dalších akcelerátorů. V současné době toto není možné z již zmiňovaných důvodů. Implementace sebou nese i synchronizační rizika, které bude možné jednoduše vyřešit s novou verzí *HWIO* popsanou v sekci 3.3.

5.5 Známé problémy DPAA2

DPAA2 architektura je v době psaní této práce ještě nevydanou novinkou. Při psaní této práce byl použit před produkční vzorek ještě nevydaného procesoru LS2088AE Rev1.0.

Chyby V současné verzi verzi Yocto SDK se vyskytují zejména dva hlavní problémy:

- *Yocto SDK v17.08* obsahuje *u-boot* nekompatibilní s *MC firmware*.
- *DPDK* aplikace nelze ukončit pokud po *DPAA2* stále přicházejí pakety. Program uvázne v rutině pro vyprazdňování bufferů. Pokud se aplikace ukončí předčasně je často potřeba restartovat celou platformu, protože došlo k poškození DPAA2 paměťových bufferů.

Neúplné implementace Řada funkcí *DPDK* není implementována anebo nefunguje správně více v dokumentaci [15]. Tato práce by využila zejména *PME* jednotku pro vyhledávání regulárních výrazů anebo *RSS* na síťových portech pro rozdělování do přijímacích front podle MAC adresy.

DPAA2 sběrnice nehlásí přehledně chyby zapojení a možnosti ladění chyb jsou velmi omezené. Vývojář je mnohdy odkázán na metodu pokus-omyl. Velmi problematická je také lokalizace problému. Při ladění problému je často nutné brát v úvahu veškerou konfiguraci. Ve verzi *yocto SDK v17.11* došlo k výrazným zlepšením, problém však přetrvává (v předchozích verzích hlášení zcela chyběly). Z tohoto důvodu byly v rámci této práce vyvinuty skripty, které konfigurují platformu. Tyto skripty kontrolují uživatelem specifikovanou konfiguraci a před konfigurací zcela reinicializují platformu. Při výskytu problému je tedy možné pomocí skriptů celou platformu vrátit do původního stavu.

Dalším problémem je, že buffery *PEB* je možné přetížit a tím poškodit. Problém jde navíc vyřešit pouze restartem [15] protože došlo ke zničení správy paměti na sběrnici *DPAA2*.

V *yocto SDK v17.03* dealokace řady objektů (zejména *DPSW*) na *DPAA2* nefunguje správně. Ve verzi *v17.11* však došlo k odstranění většiny problémů.

Ovladač pro *PME* na *DPAA2* prozatím není vydán vůbec.

Výkonnostní problémy Při využití jádra 0 pro I/O operacemi silně kolísá propustnost. Jev je způsoben zapojením řadiče přerušení, problém se ještě zhorší při použití USB.

Některé PCI-e karty můžou způsobovat ztráty výkonu. Kvůli systému přerušení.

Při nepřipojení některého z portů víceportových komponent, jako je například *DPSW* nebo *DPDMUX*, dochází k postupné degradaci propustnosti, až k úplnému zničení DPAA2 sběrnice. Všechny naalokované porty musí být korektně připojeny.

Vytvořit požadované propojení na *DPAA2* sběrnici obvykle není složité. Problém je však s propustností. Výkon *DPAA2* sběrnice ovlivňuje velké množství parametrů, krom počtů síťových rozhraní, počtů front a jejich velikostí také počty a konfigurace všech objektů na sběrnici. DPDK aplikace, které zpracovávají pakety delší dobu je poměrně komplikované odladit, tak aby jejich výkon byl aspoň na hranici funkčnosti při obvyklém provozu.

Nekompatibility Konkrétní verze firmware *MC* je navázán na konkrétní verzi *yocto SDK* a tedy verzi *u-boot*, Linux a ovladačů.

5.6 Konfigurace směrování na DPAA2

Architektura *DPAA2* popsaná v sekci 4.1 obsahuje *virtuální rekonfigurovatelnou síť*. Nová platforma vyžaduje rozdělování síťových toků na jednotlivé výpočetní jádra. S pomocí *FPGA* je možno předřadit každému paketu další ethernetovou hlavičku, kterou je možné použít ke směrování paketu. Na *virtuální rekonfigurovatelné síti* uvnitř procesoru *NXP* lze vytvořit L2 switch, ve kterém je možné nastavit statická pravidla pro směrování. Současně je možné vytvořit dostatečný počet síťových rozhraní, ze kterých mohou číst jednotlivá výpočetní vlákna aplikace.

Platforma umožňuje směrovat pakety zejména pomocí ethernetových MAC adres anebo čísla VLAN. Některé VLAN jsou na této platformě využívány samotným hardwarem a pro to je lepší využít směrování pomocí MAC adres a předcházet tak nežádoucímu chování.

Použitá konfigurace sběrnice *DPAA2* je popsána v sekci 5.4. Tato konfigurace obsahuje L2 switch *DPSW* (dále jen switch). Který je použit ke směrování. Tento switch je v po alokování nakonfigurován tak, že se MAC adresy na jednotlivých portech učí. Toto učení musí být vypnuto a musí být nakonfigurováno směrování založené na statických pravidlech.

Switch je naalokován v *DPRC* kontejneru, který je zpravován linuxovými ovladači. To mimo jiné umožňuje použít linuxové ovladače i pro tento switch. Při použití těchto ovladačů je možné ovládat switch prostřednictvím standardních linuxových nástrojů. Použité konfigurační příkazy jsou zapsány v kódu 5.3. sw0pX reprezentuje název portu switche. Tento název lze zjistit například pomocí příkazu *ip -s link*, tento příkaz zobrazuje také statistiky pro dané porty. První řádek v kódu 5.3 vypíná učení switche a přepíná switch do modu, kdy je směrováno jen pomocí statických záznamů. Druhý řádek nastavuje MTU. Toto nastavení je nutné, protože přidáním další hlavičky do paketu v FPGA se paket zvětší a ve výchozím nastavení by tedy nebyl přijat. Třetí řádek přidává statický záznam pro směrování podle MAC adresy, třetí řádek podle VLAN.

```
bridge link set dev sw0pX learning off
ip link set dev sw0pX mtu 2048
bridge fdb add 00:00:00:00:00:01 dev sw0pX
bridge vlan add vid 2 dev sw0pX
```

Algoritmus 5.3: Příkazy pro konfiguraci směrování switche.

Obrázek 5.5: Hlavní datové cesty v Sprobe10g.

Pro nově vzniklou platformu bylo vytvořeno několik skriptů, které konfigurují celou platformu včetně hlavního switche. Tyto skripty jsou součástí vrstvy pro buildovací systém *Yocto*, který je také výsledkem této práce.

5.7 Testovací prostředí

Cílová platforma, kde má být výsledek této práce provozován vznikne až v Q3 2018, tedy není dostupná v době psaní této práce. Cílovou platformu lze však pro potřeby odladění aplikace sestavit z dostupných vývojových desek a tím platformu emulovat. Způsob zapojení a verze použitých dílů a software jsou sepsány právě v této sekci.

Část s *FPGA* je realizována na vývojové kartě *ReFLEX CES R329 v1.0* s modulem, který je osazen FPGA Intel Arria10 10AS066H2F34I1SG. Toto FPGA obsahuje dvoujádrový procesor ARM A9 taktovaný na 1GHz s 4GB 32bit DDR4@2400MT/s. FPGA obsahuje 660KLE. Karta je vybavena několika ethernetovými porty: $2 \times 10GE$ SFP+, $1 \times 40GE$ QSFP+, $2 \times 1GE$ RJ45.

Procesorová část je realizována pomocí vývojové desky *NXP LS2088ARDB rev F* [16]. Tato deska obsahuje první před-produkční verzi procesoru *NXP LS2088* spolu s $16GB$ DDR4 pro procesor a $4GB$ DDR4 pro *DPAA2*. Deska má celkem 4 SFP+ porty a další 4 RJ45 ethernetové porty připojené do DPAA2. Konkrétně se jedná o SoC *LS2088AE Rev1.0*. Z procesoru je vyvedeno i *PCI-E*, ve kterém je další $1GE$ síťová karta.

Pro testovací účely je platforma zapojená následovně. Pomocí $1GE$ je spojen *HPS* a hlavní CPU podobně jako na cílové platformě. Software pro záchyt běží přímo na desce s NXP a nebo dalším serveru. Data se však ve většině případů neukládají. Důvodem je, nedostupnost dostatečně rychlých úložišť. Podrobnosti jsou uvedené u testů, ve kterých na tom záleží. Testovací platforma je zobrazena na fotografii 5.5. Oproti finální platformě popsané v sekci 2 má toto zapojení pouze spoje s FPGA vyvedeno přes optické kabely. Tato změna však nemá na parametry měřené v této práci žádný vliv. $40GE$ modul je s rozpletem využíván jako $4 \times 10GE$.

Pro generování síťového provozu je požíváno zařízení *Spirent SPT-2000A* s rozšiřovací kartou *hyperMETRICS CV 2 10G sfp+*. Toto zařízení umožňuje na dvou SFP+ portech generovat síťový provoz o rychlosti linky. Toto zařízení má podporu pouze pro jednoduché protokoly a tvorba paketů L7 protokolů je velmi omezená. Pro generování složitějších vzorů

byl použit server s procesorem *Intel Core i3-6300* ($2 \times 3.80GHz$, $4MB$ cache, $14nm$ výrobní proces) se síťovými kartami *Intel x520* ($2 \times 10GSFP+$, PCI-E 8x).

Ve všech testech SoC NXP běžen na taktech uvedených v tabulce 5.1. Tyto takty jsou nejnižší možné. Současně není známo na jaké frekvenci poběží nová platforma.

CPU	1600 MHz
Sběrncice	600 MHz
DDR	1600 MT/s
DP-DDR	1600 MT/s

Tabulka 5.1: Použité frekvence procesoru NXP.

5.8 Výkonnostní parametry knihovny HWIO

Knihovna *HWIO*, popsaná v sekci 3.3, je použita ke přístupu ke vzdáleným hardwarovým prostředkům v FPGA. *HWIO* ma architekturu klient-server, kde server běží na SoC v FPGA a klienty jsou aplikace na hlavním procesoru. Pro přenos požadavků na zápis a čtení je použit jednoduchý protokol nad TCP. Maximální propustnost TCP je často limitována velikostí posuvného okénka, pro hlavní aplikaci portovanou v rámci této práce je však důležitá latence komunikace nikoli propustnost, protože objemné datové toky tečou jinými cestami a *HWIO* je použito jen pro řídící a signalizační kanály.

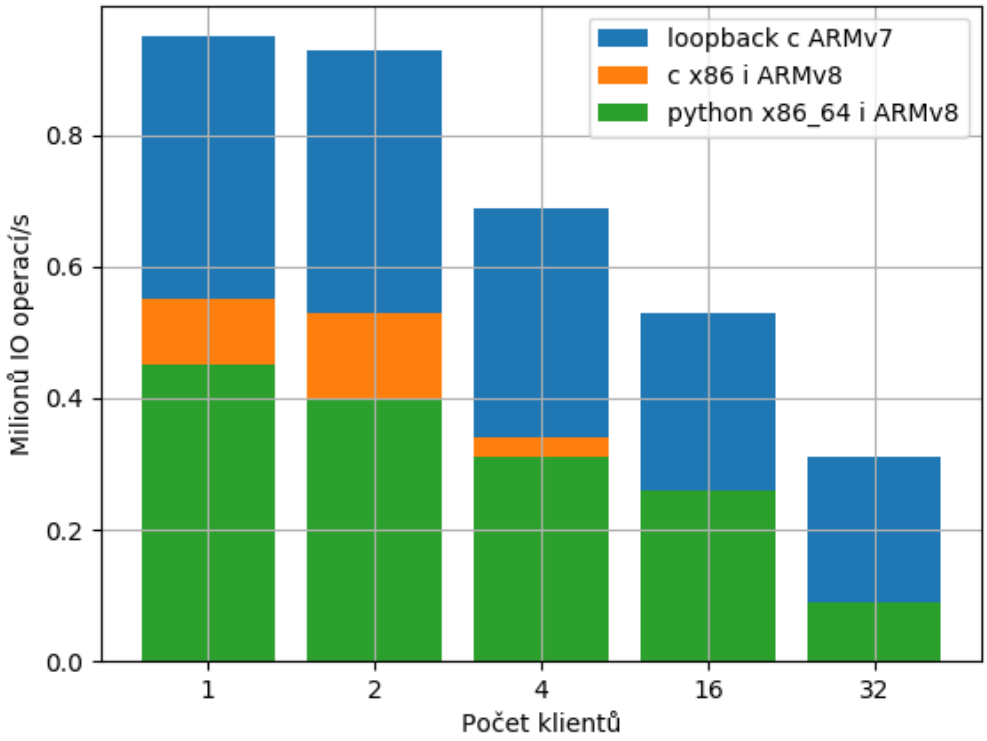
Graf 5.6 zobrazuje závislost počtu IO operací na počtu připojených klientů připojených přes různé rozhraní. Grafy jsou vykresleny pro server běžící na SoC v FPGA. Klientská testovací aplikace běží na více platformách. Testovací aplikace má dvě implementace. Implementaci v jazyce Python3 a jazyce c/C++. Latence přístupu do akcelerátorů je obvyklá latence na SoC, která se pohybuje okolo 50 taktu při 156.25MHz pro čtení a 70 taktů pro zápis. Tyto latenci mají však jen minimální vliv ve srovnání s transportní vrstvou TCP. Pro lepší představu propustnosti je graf vždy přepočítán i na propustnost.

5.9 Výkonnostní parametry NXP LS2088

V úvodu této sekce je srovnání výkonnosti procesorů nové a staré platformy v syntetických testech. Druhá polovina této sekce je zaměřena na této sekce popisuje zásady, které je nutno dodržet, aby byl dosažen požadovaný výkon. Tyto tvrzení jsou opřeny o dostupné zdroje informací a o provedená měření. Tyto zásady nemusí bezmezně platit pro budoucí verze procesorů a software. Během psaní této práce byly vydány takovéto aktualizace a následně byly údaje aktualizovány. Předpokládá se, že další verze software a hardware mohou trpět jinými problémy, a proto tato kapitola obsahuje zejména doporučení, které jsou obecně platné pro všechny revize procesorů a software. Tedy, konkrétní revize navíc obsahují problémy, které pro které je možné nalézt řešení v errata daných procesorů.

V tabulce 5.2 je srovnání procesoru NXP a Xilinx Zynq při různých frekvencích. Xilinx Zynq byl hlavním procesorem staré platformy a *NXP LS2088* bude hlavním procesorem té nové. Síťový procesor *NXP LS2088* obsahuje 8 jader *ARM A72 (64b)*, *SoC Xilinx Zynq 7020* obsahuje 2 jádra *ARM A9 (32b)* a mobilní procesor *Intel I7-4700HQ* obsahuje 4 jádra *x86_64*.

Z tabulky je patrné, že výkon starého procesoru je oproti novému téměř zanedbatelný. Procesor NXP má téměř 20× větší výkon, a to i v pro něj nejméně výhodném testu.



Obrázek 5.6: Graf závislosti počtu IO operací na počtu klientů, vztaženo na jediného klienta.

Vzhledem k okolnostem nemá smysl srovnávat tyto dva procesory, protože jsou ve zcela jiné výkonnostní kategorii. Současně tyto výsledky považujme za orientační, protože výkon ovlivňuje i použitý kompilátor, jádro systému, přepínače, taktování sběrnic a podobně. [3] Avšak pozoruhodná je závislost výkonu procesoru NXP na frekvenci. Pokud se nepodaří vyrobit novou platformu dostatečně kvalitně a bude muset fungovat na nižších taktech. To bude mít za následek velké ztráty na výkonu i při nízkém zmenšení taktu.

Procesor *NXP LS2088* a jeho akcelerátory mají nominálně velmi vysoký výkon. *DPAA2* architekturu lze přirovnat k rozsáhlé síti, jejíž propojení a nastavení jednotlivých prvků lze téměř libovolně nastavit. Jelikož je procesor ještě v před-produkčním stádiu vývoje řada věcí ještě není zcela funkční a odladěná. Chyby v konfiguraci často způsobují postupný úpadek výkonu. To je způsobeno zaplňováním některé z front nebo kvůli docházení paměťových bufferů.

Řadě problému si lze předejít dodržením následujících zásad:

Nikdy nealokovat nepoužité porty/komponenty. Současný firmware nedokáže dealokovat paměťové buffery z míst z nezapojených portů na *DPAA2*. Například při alokaci switche, kde jeden port zůstane nezapojen se postupně spotřebují všechny paměťové buffery a platforma přestane fungovat.

[3] Software pro různé platformy byl ve značně rozdílných verzích (GCC 5.4, 7.3) ve všech případech byla použita optimalizace kompilátoru na rychlost $-O3$.

CPU	Frekvence CPU	Operační paměť	DMIPS	Core Mark
NXP LS2088	2 GHz	DDR4 2133 MT/s	97600	81600
NXP LS2088	1.8 GHz	DDR4 2100 MT/s	77720	64983
NXP LS2088	1.6 GHz	DDR4 1600 MT/s	43088	32245
Xilinx Zynq 7020	800 MHz	DDR3 533 MT/s	1815	4737
Xilinx Zynq 7020	666 MHz	DDR3 533 MT/s	1532	4013
Intel I7-4700HQ	2.4 GHz	DDR3 1600 MT/s	22659	17498

Tabulka 5.2: Srovnání výkonnosti procesorů.

Dodržovat správný počet front. Podobně jako u portů i jednotlivé fronty dokáží způsobovat ztráty paměťových bufferů.

Dodržovat správný počet pomocných objektů jako je DPBP/DPCON/DPIO a dalších. Při nedostatku pomocných objektů vznikají nedeterministicky výkonnostní i funkční problémy. Správné počty jsou uvedeny v dokumentu [15], ale počet je dán značným množstvím parametrů. Obvykle lze správnou konfiguraci nalézt několika pokusy s různými počty těchto objektů a sledováním gradientu propustnosti/chybovosti/zahozených paketů.

Nepoužívat jádro 0. Jádro 0 obsluhuje většinu přerušení při jeho zatížení dochází k masivnímu úpadku výkonu. Je doporučeno i izolovat Linux kernel od tohoto jádra pomocí *isolcpus=<core list>* bootovacího argumentu. Pro tento procesor je doporučeno použít *isolcpus=0*. Problém nultého jádra lze jasně vidět z grafů 5.7, 5.9, 5.8. Všechna další měření jádro 0 již nepoužívají.

Použít správné DMA pro správné jádro. Uživatel zdánlivě neřídí přiřazení DMA pro jednotlivé jádra se však stále děje. Při přeposílání paketů napříč jádry se při mempool (viz sekce 3.2) operacích s paketem. Při přepnutí DMA začne DMA nahrávat nově příchozí pakety do špatného potenciálně špatného jádra. To způsobí cache miss při přístupu k paketu na původním jádře a na druhé jádře dochází k jevu, který se nazývá *cache trashing*, který taktéž způsobuje nadměrné výskyty *cache miss*. Tento problém způsobuje propad propustnosti zpravidla pod $100Mb/s$ při $128B$ paketech, avšak zpomalení je značně nedeterministické. Velmi vhodné je vyhnout se případům kdy se pakety ve formě struktur mbuf (viz 3.2) pohybují mimo jádro na které byly původně přijaty nebo naalokovány.

Použití 1G hugepages. Pro provozování DPDK aplikací je použití hugepages nutnost. Při použití 1G hugepages však téměř odpadá potřeba prohledávání tabulky stránek. Protože pravděpodobnost, že adresa bude v *TLB*[4] je vysoká, u *TLB* v *IOMMU*[5] je téměř jistá. Tento problém však způsobuje jen zanedbatelný propad výkonnosti.

Použít správnou velikost dávky. Při použití DPDK je nutné použít velikost dávky nejméně 16 paketů, jinak *DPDK* funkce *rte_eth_rx_burst* nikdy žádné pakety nevrátí. A program jen nekonečně cyklí v hlavní smyčce.

[4]Translation Lookaside Buffer - rychlá vyrovnávací paměť v MMU pro urychlení překladu virtuální adresy na fyzickou

[5]MMU pro periferie/akcelerátory

Proměnné prostředí DPDK aplikací Aplikace s *DPDK* na této platformě lze navíc konfigurovat pomocí proměnných prostředí. Tato skutečnost není uvedena v oficiálních dokumentacích. Pro ladění výkonu jsou tyto parametry velmi důležité. Jejich výčet je následující.

- *DPRC* - Jediný dokumentový parametr, který slouží k nastavení hlavního *DPRC* se který má aplikace použít. Do proměnné musí být nastaveno jméno *DPRC* jako je například *dprc.1*.
- *DPAA2_RX_TAILDROP_SIZE* - Maximální počet byte ve vstupní frontě než vstupní rozhraní začne zahazovat příchozí pakety. Při hodnotě 0 je zahazování zcela vypnuto. Výchozí hodnota je 65536.
- *DPAA2_PORTAL_INTR_THRESHOLD* - Pomocí této proměnné prostředí je možné nastavit počet paketů po kterém má hardware vyslat přerušení a informovat tak o nově přijatých paketech.
- *DPAA2_PORTAL_INTR_TIMEOUT* - Pomocí této proměnné je možné nakonfigurovat čas, po kterém má hardware informovat software, o tom že žádné pakety na daný port nepřicházejí.
- *DPAA2_NO_PREFETCH_RX* - Definováním této proměnné se při volání funkci *DPDK* pro příjem paketů se neprování prefetch. Je tak možné dosáhnout lepší latence nebo vyhnout se cache poisoning.
- *DPAA2_HOST_START_CPU* - Pomocí této proměnné lze specifikovat ID CPU. Tato proměnná slouží ke konfiguraci ve virtualizovaném prostředí. ID CPU je číslo fyzického CPU na kterém jsou spuštěny vlákna virtualizovaného stroje.
- *DPAA_FMAN_UCODE_SUPPORT*
- *DPAA2_TX_CGR_OFF* - Tento přepínač slouží pro vypnutí kongrese na výstupu síťového rozhraní.

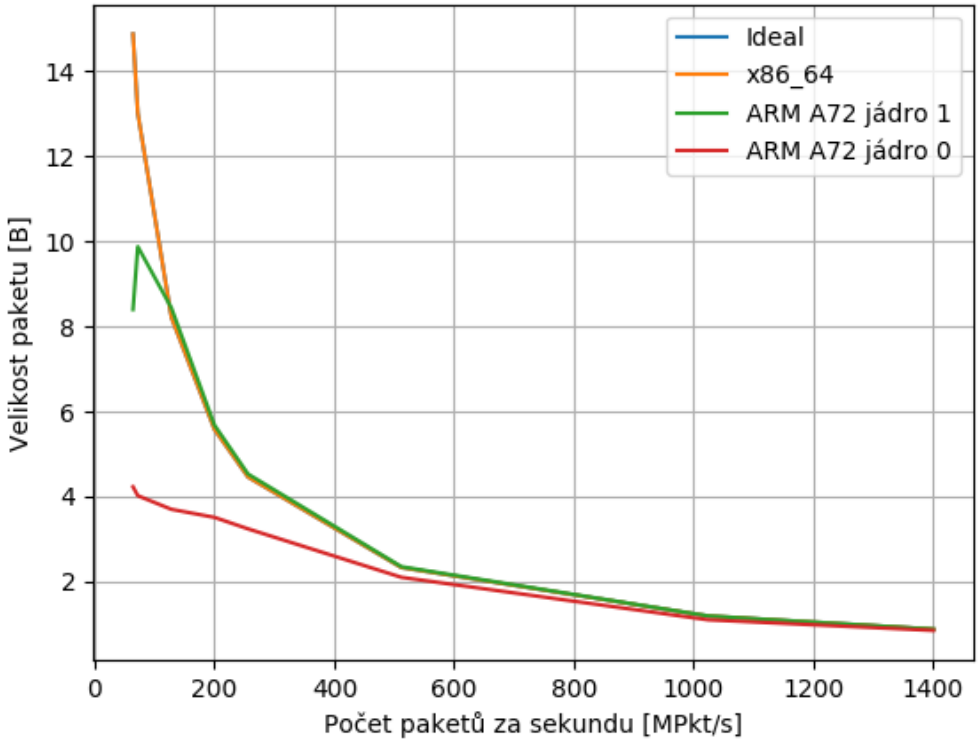
Pro měření propustnosti síťových rozhraní na procesoru byl použit nástroj *testpmd*. Tento nástroj je součástí frameworku *testpmd* a slouží právě pro měření propustnosti. Všechna měření byla prováděna s osmi paměťovými stránkami o velikosti 1GB. Kde není uvedeno jinak je použita jedna fronta na port, ovladače však obsahují chybu, která způsobí, že se nakonec použijí dvě fronty. Ve verzi Yocto SDK 2.0-1703 konfigurace s jedinou frontou procesor rovnou zasekla. Pokud není uvedeno jinak testy probíhaly s následujícími velikostmi paketů: 64, 72, 128, 200, 256, 512, 1024, 1400.

Konfigurací loopback je označováno zapojení kdy na vstupní vlákno do procesoru je přiveden síťový tok o plné rychlosti a rychlost je počítána podle vlákna výstupního. Procesor v tomto režimu pouze přeposílá pakety ze vstupu na výstup. Graf 5.7 zobrazuje závislost počtu přijatých paketů za sekundu. Při menších velikostech paketů vzniká větší režie při jejich zpracování a proto se reálná křivka pohybuje pod ideální. Graf zobrazuje výsledky pro jeden vstupní port přiveden na jádro 0 nebo 1 a pro referenční stroj s procesorem Intel.

Graf 5.8 zobrazuje rychlost odesílání paketů v závislosti na velikosti paketu. Hlavním limitem je znovu počet paketů. Protože data nemusí být neustále načítány do cache výsledná rychlost je větší. Jádro 0 v tomto případě není omezeno přerušeními.

Při propojení optických portů na úrovní *DPAA2*, tedy bez zásahu procesoru, dosahuje propustnost rychlosti linky.

Graf 5.9 znázorňuje to stejné v zapojení looback. Při tomto zpracování provozu na procesoru dochází už na jednom portu při nejmenších paketech až k 50% zahazování provozu. Toto zahazování je však způsobeno omezením sběrnice. Problém lze obejít použitím více front a nebo použitím více síťových rozhraní. Propustnost 50% tedy není limitem platformy, ale její konfigurace.



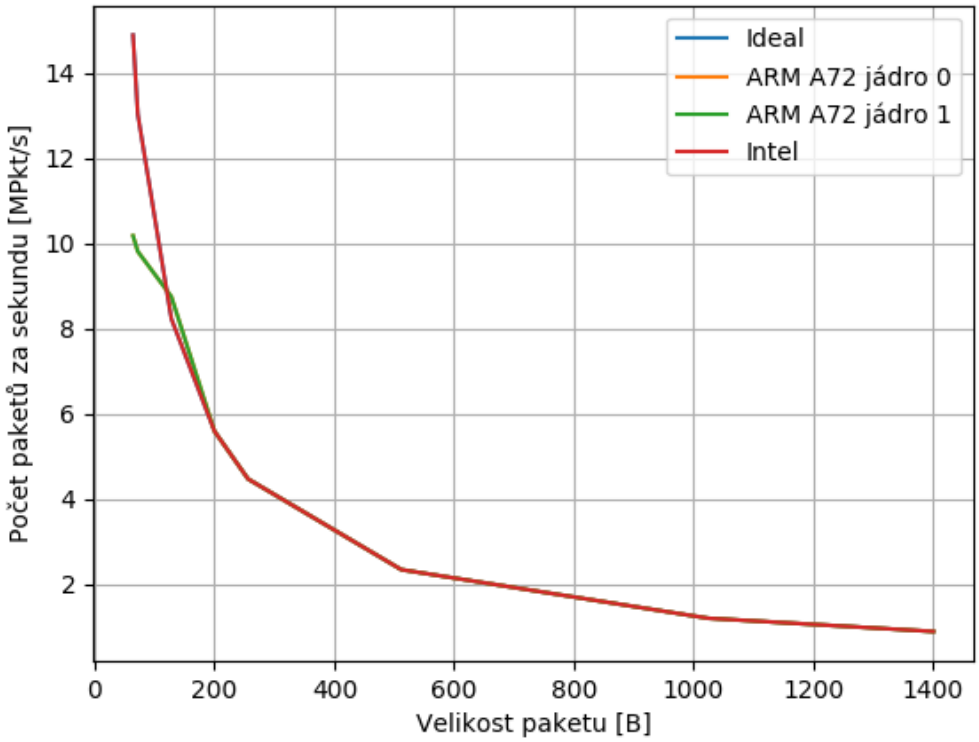
Obrázek 5.7: Propustnost v závislosti na velikosti paketu, konfigurace jeden port přijímání. Křivka pro *ideál* a *x86_64* se překrývá.

Graf 5.10 znázorňuje výkonost procesoru v zapojení looback. Je vidět, že se použitím více procesorů propustnost naopak klesla. Tento jev je způsoben tím, že fronty síťových portů na *DPAA2* nejsou využity a zdá se, že zahazování paketů má poměrně vysokou režii. Problém lze částečně vyřešit použitím více front.

Graf 5.11 odkrývají skutečnost, že kopírování paketů na L2 switchi je pro malé pakety pomalé. Při nejmenších paketech dochází k zahazování až v 60% případech.

Graf 5.12 zobrazuje propustnost pro správně nakonfigurovaný switch do software. Se správnou konfigurací swtich nezpůsobuje žádné zpomalení. L2 switch ale prozatím neukazuje v kombinaci s *DPDK* správně zahozené pakety na vstupních portech, protože *DPDK* ovladač nepočítá správně statistiky v konfiguraci se switchem. Tedy ukazuje jen část zahozených paketů, na switchi ani vstupním *MAC* přitom nejsou zahozeny téměř žádné pakety. Switch má nominální přepínací kapacitu $88Gb/s$ a na současné platformě neexistuje žádná konfigurace, která by trpěla na nestatečnou rychlost switche.

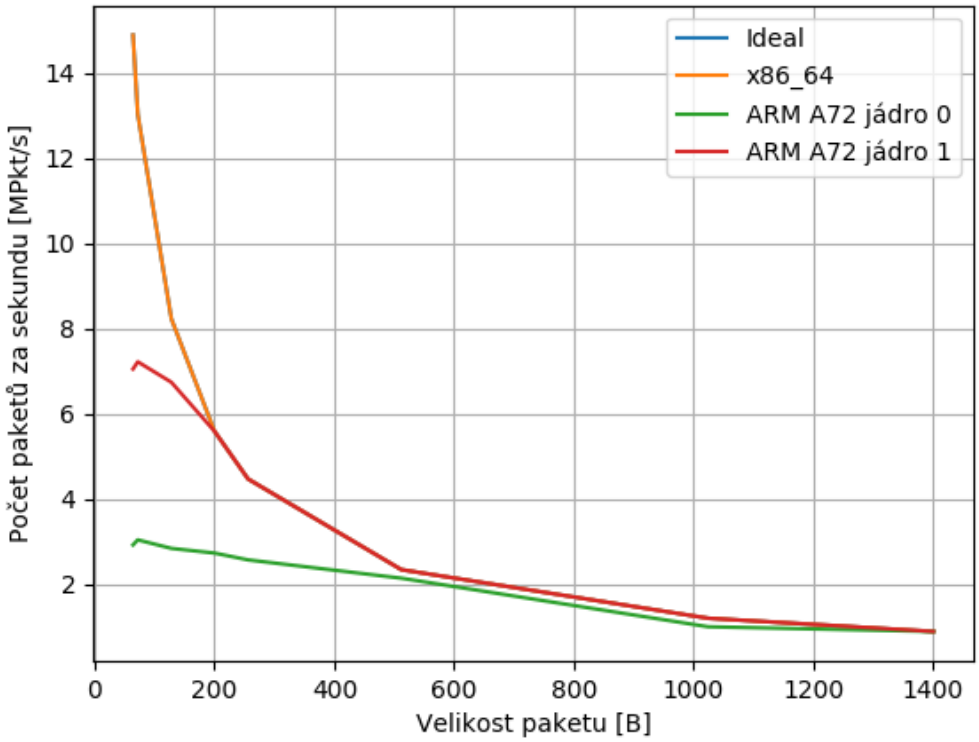
Tabulka 5.3 obsahuje hodnoty propustnosti pro jednotlivé konfigurace velikostí front a limitů zahazování. Parametr *rxd* vyjadřuje počet deskriptorů v přijímací frontě. Parametr



Obrázek 5.8: Propustnost v závislosti na velikosti paketu, konfigurace jeden port odesílání. Křivka pro *ideál* a *x86_64* se překrývá, křivka pro ARMv8 jádro 0 a 1 je totožná.

mbufs vyjadřuje celkový počet struktur mbuf dostupných v centrální instanci *rte_mempool*. Pomocí proměnné prostředí *DPAA2_RX_TAILDROP_SIZE* lze konfigurovat zahazování na vstupních frontách, tento parametr je v bytech. Při hodnotě 0 je zahazování zcela vypnuto. V jiném případě pokud velikost vstupní fronty překročí tuto hodnotu pakety začnou být zahazovány. Tabulka je zapsána pro 1 jádro a jedno síťové rozhraní podobný trend jde však pozorovat nezávisle na počtu rozhraní a jader. Rychlost zpracování paketů jistě souvisí s přítomností paketu v paměti cache. Při pohledu na tabulku však je patrný ještě další problém. Při přetížení fronty se fronta ještě víc zpomalí. Je tedy žádoucí, aby zahazování bylo aktivní a bylo nastaveno správně pro danou aplikaci. Pro aplikaci *PaSt* by bylo velmi výhodné zcela se vyhnout zahazování, to však není možné.

Na sběrnici *DPAA2* se poměrně často objevují věci, které nelze vysvětlit se znalostmi z dostupných dokumentů. Jedním z nich je chování při nerovnoměrných zátěžích. Ukázkovým příkladem jsou měření v tabulce 5.4. Platforma je pro toto měření nakonfigurována v následující topologii. Čtyři vstupní SFP+ porty jsou připojeny do switche na DPAA2. Na tento switch je pak připojeno dalších 6 síťových rozhraní. Podobně jako na finální platformě pro PaSt. Do vstupního portu 0 a 1 tečou celkem čtyři datové toky, které switch přeposílá na jádra 2, 3, 4 a 5. Do portů 0 a 1 tečou dva toky, které jsou směrovány na jádra 0 a 1. Dle předpokladu jádra 0 a 1 přijímají nejvíce paketů. Avšak zvětší-li se velikost pro zahazování první dvě jádra začnou přijímat ještě více paketů na úkol druhých dvou jader, avšak téměř bez ovlivnění posledních dvou jader. Navíc se zvyšující se velikostí paketu se rozdíl mezi

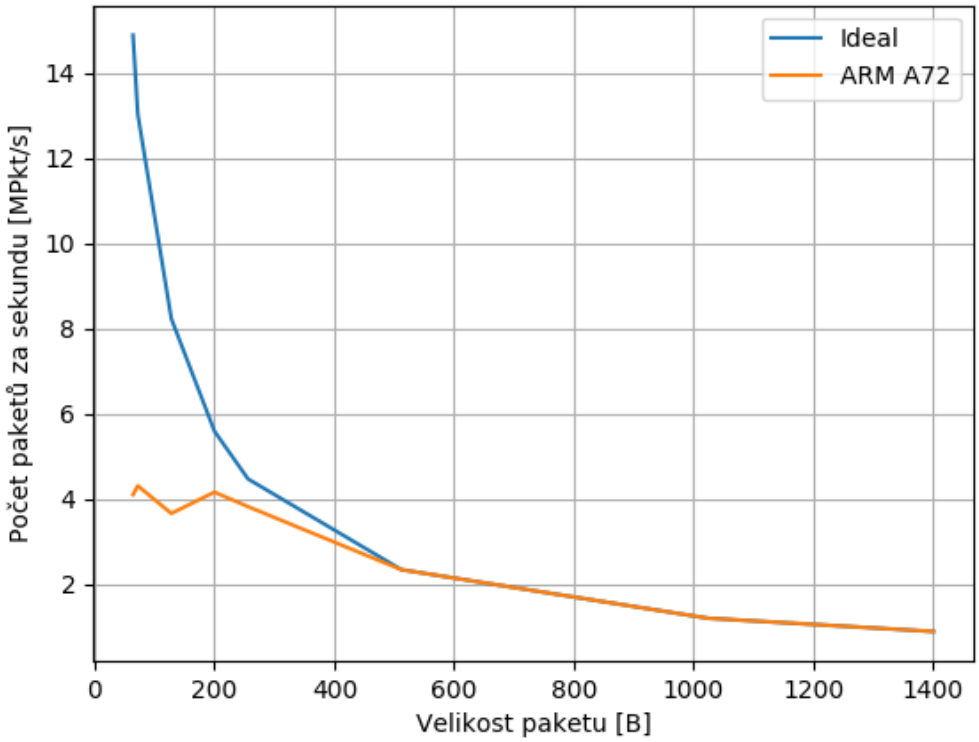


Obrázek 5.9: Propustnost v závislosti na velikosti paketu, konfigurace jeden port loopback. Křivka pro *ideál* a *x86_64* se překrývá.

jádry 2, 3 a 4, 5 znovu smaže. Problém ukazuje na špatné nastavení priorit někde uvnitř konfigurace *DPAA2* avšak všechny jsou stejné. Přenastavením priorit si lze vynutit prioritizováním některého ze spojů a tedy některého z jader. Rovnoměrné rozložení je však není lehce dosažitelné pro všechny topologie a všechny poměry zátěže.

Crypto engine Pro to, aby byl crypto engine na této platformě aktivní je potřeba použít následující moduly jádra *caamalg* nebo *caamalg_qi*, *caamhash*, *caam_pkc*, *cryptodev*. Následující měření byla prováděny pomocí nástroje *openssl* tento nástroj je součástí stejnojmenné známé knihovny pro šifrovaní[6]. Testovací nástroj v použité verzi ještě neobsahoval možnosti konfigurace délky běhu. Tato možnost byla doimplementována podle novějších verzí. Naměřené výsledky pro procesorová jádra odpovídají očekáváním. Procesorové jádro procesoru NXP je při menších blocích pomalejší kvůli tomu, že nelze efektivně používat sběrnici procesoru. Avšak u akcelerátoru *DPSEC* dochází v testech k propadu téměř o tři řády. Tento propad je způsoben opětovnou reinicializací a pro menší velikosti bloku, test neodpovídá výkonu při reálné zátěži. Chování v reálné zátěži není možné měřit, protože není dostupné zařízení, které by mělo dostatečný výkon.

[6]Konkrétně byla použita verze *OpenSSL 1.0.2l 25 May 2017*



Obrázek 5.10: Propustnost v závislosti na velikosti paketu, pro 4 porty najednou v konfiguraci loopback. Průměr na port.

5.10 Výkonnostní parametry programu PaSt

Program *PaSt* pro analýzu síťového provozu na L7 vrstvě byl úzce svázán s původní platformou a je popsán v sekci 3. Tento software v nekonečné smyčce čte pakety ze vstupních portů bufferuje je prování na nich vyhledávání regulárních výrazů případně další parsování. Pro zvýšení výkonu jsou použity hardwarové akcelerátory v FPGA a nově in v *DPAA2*.

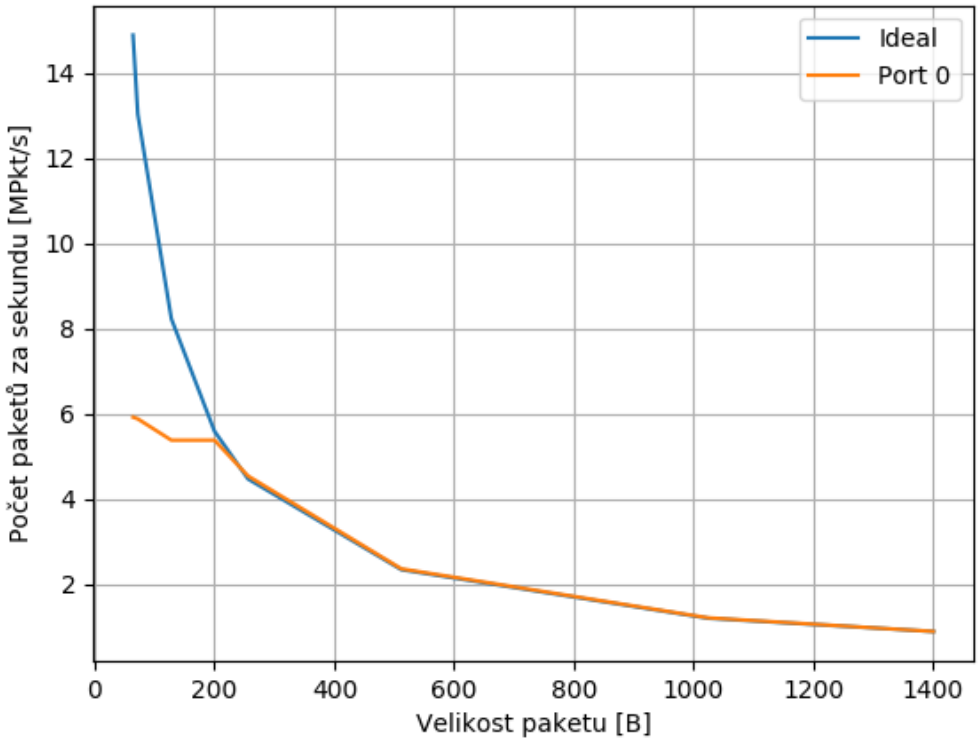
Program past využívá pooling, tedy v nekonečné smyčce kontroluje vstupní porty zdali jsou na nich dostupné příchozí pakety. Tento fakt komplikuje testování výkonnosti nástroji jako je například *perf*, který mimo jiné dokáže zobrazit čas strávený ve volání jednotlivých funkcích.

Současně vstupní datové toky určují chování programu PaSt. Záleží nejen na samotných paketech a jejich pořadí, ale i na předchozích nesouvisejících paketech, které se do hardwaru mohly dostat. To se týká především zaplněnosti filtru v hardware. Závisí také na konfiguraci tohoto software, přidání dalšího odposlouchávacího pravidla zesložiťuje regulární výrazy pro vyhledávání.

S ohledem na komplikace byla pře začátkem psaní této práce provedena analýza jejíž výsledkem bylo objevení problémů v alokaci paměti a vyhledávání v hlavní flow tabulce, tyto problémy byly odstraněny.

Na začátku této práce tedy software obsahoval zejména tyto kritická místa.

- Vyhledávání v hlavní tabulce toků



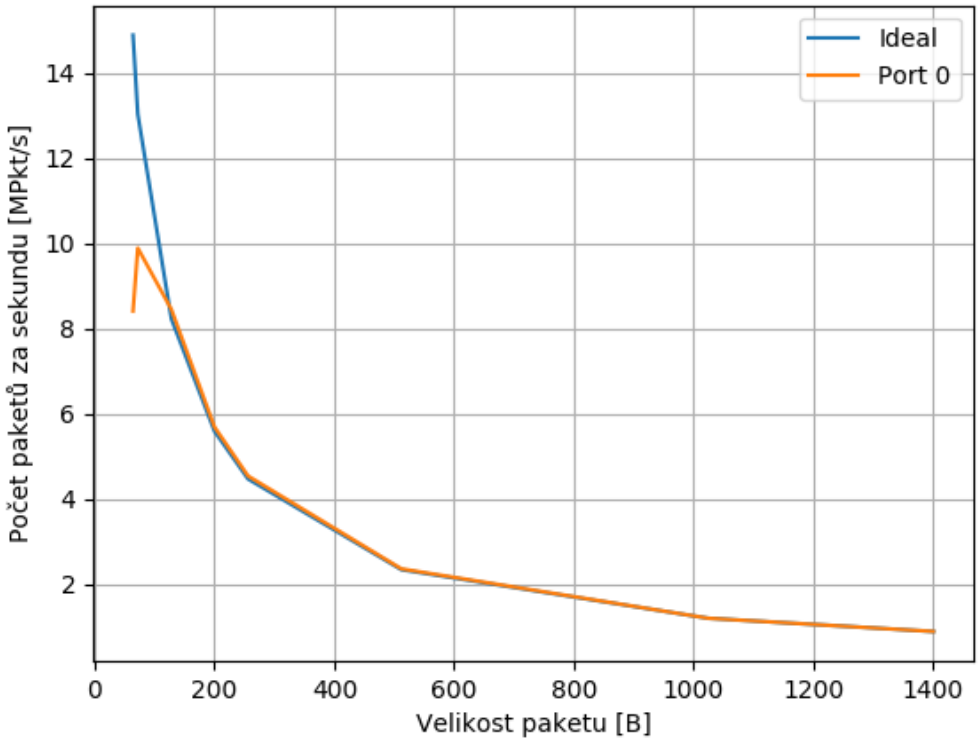
Obrázek 5.11: Graf závislosti propustnosti na velikosti paketu DPSW (L2 switch) bez konfigurace se dvěma porty.

- Vyhledávání regulárních výrazů
- Čtení ze vstupních portů
- Konfigurace hardware

Významnost těchto kritických míst nelze jednoduše seřadit. Zpomalení kvůli těmto kritickým místům se projevuje v závislosti na konfiguraci i aktuálním stavu hardware a software, závisí také na počtu a povaze vstupních toků.

Vyhledávání v hlavní tabulce toků je prováděno pro každý paket, který dorazil do software. V rámci tohoto vyhledávání je na procesoru spočítán heš, který slouží k vyhledávání v hlavní tabulce toků, která je realizována pomocí hešovací tabulky. Heš je počítán z hlavičky paketu. Výpočet tohoto heše prodlužuje dobu zpracování paketu o méně než 5% v případě nejkratších paketů, které reprezentují nehorší případ. Heš by bylo možné spočítat v FPGA. Vzhledem k potenciálnímu zrychlení se tak prozatím neděje.

Výkonnost platformy má své specifika a závisí na mnoha faktorech. Proto se tato práce zaměřuje především na nejhorší případy. Pro tyto nejhorší případy existují vzorové datové toky. Tyto datové toky jsou uloženy v souborech ve formátu PCAP. Proto, aby bylo možné platformy aspoň částečně porovnat, byly komě dalších testů použity právě tyto datové toky. Pro jedno-gigabitovou platformu jsou výsledky uvedeny v tabulkách 5.6 a 5.7 pro nově vzniklou platformu pak v tabulce 5.8. Tato práce využila starší soubory PCAP, protože umožňovaly lepší přehled o rozložení datových toků než testy pozdější. Současně výkon



Obrázek 5.12: Graf závislosti propustnosti na velikosti paketu DPSW (L2 switch) bez konfigurace s jedním SFP+ portem a jedním síťovým rozhraním.

platformy do dnešní doby narostl a proto jsou tu pro srovnání i aktuální výsledky pro podobný test.

V měřeních, která využívaly soubory ve formátu PCAP byl pro generování datových toků použit standardní linuxový nástroj *tcpreplay*, byl nakonfigurován tak, aby výstupní tok dosahoval maximální rychlosti. Byl tedy použit příkaz 5.4.

```
tcpreplay −t −−preload−pcap −i <interface> −l 0 −T rdtsc <pcap file>
```

Algoritmus 5.4: Použití nástroje tcpreplay.

Mediační funkce běžela na stejném procesoru jako aplikace a bylo vypnuto ukládání do souborů. Vliv zpoždění exportního kanálu je tedy téměř eliminován.

Měření pro platformu Sprobe1g nebyla prováděna v rámci této práce. Měření byla prováděna v iteracích. V každé iteraci byla sonda restartována, aby došlo ke smazání pravidel ve filtru a následně byla navyšována rychlost na vstupních linkách dokud nezačala sonda zahazovat vstupní pakety.

Vhledem k tomu, že logika parsování datových toků a četnost komunikace s filtrem zůstala stejná nejsou uvedeny všechny protokoly, protože jejich výkon jen narostl ve stejném poměru a místo toho je dán prostor rozboru problémů omezujících výkon současné implementace. Výkon pro novou platformu ve výchozím nastavení je tabulce 5.8. Propustnost je však při tomto měření omezena zejména dvěma faktory. První je, že zahazování na *DPAA2* není spolehlivé a dochází k němu mnohdy náhodně. Přičemž tento test kontroluje

rxd	mbufs	RX_TAILDROP [B]	MPkt/s
2048	4096	16384	8.77
4096	8192	16384	8.76
8192	32768	16384	8.76
32768	262144	16384	8.79
2048	4096	65536	8.78
4096	8192	65536	8.76
8192	32768	65536	8.76
32768	262144	65536	8.76
2048	4096	4194304	8.78
4096	8192	4194304	8.83
8192	32768	4194304	6.77
32768	262144	4194304	5.74
2048	4096	0	8.84
4096	8192	0	6.62
8192	32768	0	6.77
32768	262144	0	5.69

Tabulka 5.3: Vliv nastavení zahazování a velikostí front na propustnost.

RX_TAILDROP [B]	Velikost paketu [B]	Rychlost portů
32768	64	3.13/3.12/2.84/2.84/2.92/2.98
65536	64	4.96/4.91/2.49/2.50/3.03/3.00
65536	128	5.71/5.14/2.86/2.95/2.33/2.41
65536	256	2.83/2.90/1.93/1.95/1.94/1.95
65536	1500	0.82/0.82/0.41/0.41/0.41/0.41

Tabulka 5.4: Anomálie v chování L2 switche.

právě zahazování. Druhým problém je implementace knihovny pro pro přístup do *FPGA* a knihoven, které ji používají. Tyto implementace nejsou optimální pro novou platformu a jejich problémy a řešení jsou zmíněny v kapitolách 3.3 a 5.11. Přestože v současnosti není rozumné tyto změny provádět v budoucnosti to možné bude a rychlost by se mohla přiblížit syntetickému testu v tabulce 5.9. Tento syntetický test nekomunikuje s filtrem, což bude výkonnostně téměř stejné s asynchronní implementací *HWIO*.

5.11 Budoucí vývoj platformy

Tato kapitola se zabývá budoucími možnostmi platformy a dalšími kroky, které by bylo vhodné provést. Řada z nich povede na posun výkonu o řády řadu z nich nebude možné provést, dokud NXP nezajistí potřebnou podporu ve formě ovladačů. Následující odstavce popisují možná vylepšení. Tyto vylepšení jsou seřazeny podle toho jak je možné je provést. Většina je na sobě závislých.

Ovládání filtru pomocí vzdáleného volání procedur Akcelerátor L3/L4 filtru v *FPGA* je ovládán pomocí několika registrů. Při přidávání pravidla do filtru jsou hodnoty zapsány do konfiguračních registrů následně je v cyklu kontrolován příznak dokončení. Tento pří-

	Šifra	16 B	64 B	256 B	1024 B	8192 B
CPU NXP LS2088	aes-256-cbc	399.58	691.19	845.51	885.31	909.83
i7-4800MQ	aes-256-cbc	544.50	568.53	567.62	582.52	574.66
i3-6300	aes-256-cbc	762.74	1020.83	1042.50	1046.44	1046.80
DPSEC	aes-256-cbc	49.58	154.91	987.17	4977.15	21442.29
DPSEC	aes-128-cbc	36.14	194.70	861.26	7515.22	22078.14

Tabulka 5.5: Srovnání výkonnosti *DPSEC* pro jednotlivé velikosti bloku, hodnoty uvedeny v MB/s.

Protokol	Testovací sada	Rychlost exportu [Mb/s]
SMTP	gurpartap	13
SMTP	clara	10-14
IMAP	L7_IMAP	0.32-0.42
POP3	L7_POP3	4.9-6.5
FTP	LOGINS L7_FTP	13.8-14.9
FTP	FILENAMES L7_FTP	15.7-16.1

Tabulka 5.6: Výkonnostní parametry v nejhorších případech pro platformu *Sprobe1G* v 2016 Q4 (Převzato z wiki stránek Sprobe).

stup je pro novou verzi knihovny *HWIO* velmi pomalý. Bylo by vhodné zavolat obslužnou rutinu na serveru. Současná verze knihovny *HWIO* to však nepodporuje. Vzdálené volání procedur v knihovně *HWIO* lehce implementovatelné jako další příkaz, ale všechny aplikace a knihovny, které ji využívají musí být upraveny, aby této nové funkce využívaly. Vzniká také problém v cyklické závislosti mezi aplikacemi a *HWIO*. Tento problém lze řešit pomocí pluginů pro *HWIO* server. Plugin bude sdílená knihovna pouze s funkcemi pro ovládání daného hardware. *API* hlavní knihovny může být rozděleno na *API* pro pluginy, které bude poskytovat lokální přístup k hardware a *API* pro běžné aplikace, které bude poskytovat přístup k hardwarovým prostředkům dle konfigurace knihovny. Každý plugin musí explicitně označit své funkce API a zaregistrovat je do HWIO. K registrování těchto funkcí může být použito například c konstruktoru modulu `__attribute__((constructor))`. Pomocí tohoto rozšíření jazyka C, které je dostupné v překladačích GCC, Clang i dalších je možné zavolat funkci ještě před spuštěním funkce main. Použití pluginu pak může být vyřešeno přednačtením dané sdílené knihovny nebo ještě lépe umístěním do adresáře ze kterého budou všechny knihovny přednačteny *HWIO serverem*.

Asynchronní přístup k hardware v FPGA Přístup k řadě komponent v FPGA je možno provádět v FPGA. Problémů je však několik. Dále předpokládejme implementaci asynchronní komunikace s pomocí C++ knihovny *asio* anebo s *Boost.Asio*. Tyto knihovny zpracovávají asynchronní události s pomocí dalšího vlákna. V konjunkci s *DPDK* kde výpočetní vlákna neustále vytěžují procesory může dojít ke zpomalení. Není však důvod proč by toto zpomalení mělo být vysoké. Problematický je také nárůst latence. Hardware filtru také není na tento typ komunikace připraven.

Optimalizace spotřeby zařízení Aplikace používající *DPDK* obvykle v nekonečné smyčce kontrolují příchozí pakety. Tento přístup není vhodný u vestavěného zařízení s po-

Protokol	**Poznámka**	**Rychlost vstupu [Mb/s]**
SMTP		213
POP3	pouze email	214
POP3	celé sezení	278
IMAP	pouze email	284
IMAP	celé sezení	323
FTP	celé sezení	92
FTP	pouze soubor	16

Tabulka 5.7: Výkonnostní parametry při 10% zájmového provozu a 1KB aplikačních dat pro platformu *Sprobe1G* v 2018 Q1 (Převzato z wiki stránek Sprobe).

Počet jader	Protokol	testovací sada	Rychlost exportu [Mb/s]
1	SMTP	gurpartap	75
2	SMTP	gurpartap	163
6	SMTP	gurpartap	359

Tabulka 5.8: Výkonnostní parametry v nejhorších případech pro novou platformu *Sprobe10G*.

měrně vysokou spotřebou. Existuje několik způsobů jak vyřešit tento problém. Výrobce může přidat API, s pomocí kterého bude možné výpočetní vlákna uspat pokud nebudou přicházet žádné pakety na daný port. Hardware danou funkcionalitu podporuje přerušení ze síťových rozhraní jsou přemostěna s pomocí knihovny *VFIO* do *DPDK* aplikace. Při startu DPDK aplikace jsou tyto přerušení zakázány. Pro jiné části DPDK se však přerušení běžně používají.

Pokud tato část API nebyla nikdy implementována je možné ji implementovat ve vlastní režii. Nebo jde toto chování virtualizovat s pomocí řídícího vlákna a hardware. Výpočetní vlákna mohou být přidávána dle potřeby s přibývajícími síťovými toky a zatížením jednotlivých jader. Toto řešení však sebou nese komplikace se zastavováním výpočetních vláken kde obsah všech tabulek musí být sloučen.

Pokud by bylo zapotřebí ještě agresivnější zprávy spotřeby je možné při nečinnosti vypnout některé sériové linky mezi FPGA a SoC NXP. Do *FPGA* by bylo nutno přidat další směrovací logiku která by umožnila vypínat jednotlivé sériové linky a přesměrovat provoz do zbývajících.

Použití PME Pro analýzu aplikačních protokolů se výborně hodí akcelerátor *PME* popsaný v sekci 4.1.5. S použitím tohoto akcelerátoru bude možné odstranit vyhledávání regulárních výrazů ze software. To v důsledku způsobí to, že procesory budou provádět správu tabulek toků a export datových toků. Zatížení *GPP* tedy klesne o více než 90%. Ovladačem a obslužným software k tomuto akcelerátoru se výrobce začne zabývat nejdříve v Q2 2019.

Použití AIOP Po tom co bude funkční akcelerátor *PME* bude možné efektivně využít i *AIOP* akcelerátor popsaný v sekci 4.1.7. Toto pole procesorů má speciální akcelerátory na zprávu tabulek toků a má dostatečný výkon pro aktualizace stavů toků podle příznaků z *PME*. AIOP může k paketům přibalit hlavičky se stavem toku podle, kterých může *GPP*

Počet jader	Protokol	Testovací sada	Rychlost exportu [Mb/s]
1	SMTP	gurpartap	163
2	SMTP	gurpartap	315
6	SMTP	gurpartap	570

Tabulka 5.9: Výkonnostní parametry v nejhorších případech pro novou platformu *Sprobe10G*, s použitím statické konfigurace filtru.

provádět export a filtrování. Tím bude téměř celá úloha, kterou vykonává analyzátor aplikačních protokolů PaSt převedena do hardware a *GPP* bude možno využit k dalším úlohám.

Použití TCP implementace v DPDK *DPDK* neobsahuje implementaci *TCP/IP* protokolu, ale existují implementace třetích stran. Například tvůrci knihovny *mTCP* uvádějí, že je až 25× rychlejší než *TCP stack* z jádra systému Linux. S použitím knihoven pro *TCP* v *DPDK* tedy dojde k výraznému zrychlení exportování zachycených síťových toků. Avšak odstraněním linuxových soketů se znemožní používání standardních knihoven pro šifrování.

Šifrování exportního toku Šifrování TCP toků na této platformě (dále jen šifrování) je možné provádět pomocí knihovny *OpenSSL*, která je upravena výrobcem hardware tak, aby používala šifrovací akcelerátor dostupný na této platformě. Tento akcelerátor je taktéž možné použít v *DPDK* prostřednictvím *cryptodev* rozhraní a i z rozhraní šifrovacích zařízení z Linux. Tento šifrovací akcelerátor je použit prostřednictvím šifrovaného tunelu mezi platformou a cílem exportování síťových toků tzv. *mediační funkcí*. Použití SSH tunelu sebou nese zbytečnou režii navíc a do budoucna by měl být odstraněn. Vzhledem k nízkému procentu zájmového provozu, který je potřeba přes tento tunel přeposílat, nebyl tento problém doposud aktuální.

Kapitola 6

Závěr

Cílem této práce bylo navrhnout platformu pro analýzu síťového provozu s použitím existující aplikace L7 analyzátoru *PaSt*, síťového procesoru *NXP LS2088* a *FPGA*. Výsledkem této práce je funkční optimalizovaná portace existující platformy pro analýzu síťového provozu o nižších rychlostech a zcela jinou architekturou. Jedno-gigabitová i nová platforma jsou popsány v kapitole 2.

Firmware FPGA byl aktualizován mimo zaměření této práce. V rámci této práce byl navržen a implementován systém pro hardwarovou synchronizaci paralelního zpracování pro aplikaci síťového analyzátoru *PaSt*. Tento systém je použitelný i pro další síťové aplikace, využívá *FPGA* a *virtuální rekonfigurovatelnou síť* pro rozdělování zátěže na výpočetní vlákna. Síťový procesor *NXP LS2088* je ve světě vestavěných systémů neobvykle velmi výkonný hardware. Jeho vývoj však ještě není u konce. Možnosti odhalování příčin problémů na této platformě jsou prozatím velmi omezené. Výkon je často velmi nedeterministický (viz konec sekce 4) a implementace *DPDK* má řadu uživatelsky velmi nepříjemných vlastností viz sekce 5.5. Přesto byly objeveny konfigurace, se kterými platforma funguje dobře a dosahuje excelentního výkonu.

V rámci této práce byl proveden port a následná optimalizace existující aplikace *PaSt* pro analýzu síťového provozu na L7 OSI vrstvě primárně určené pro legální odposlechy, viz sekce 3. Nová implementace této aplikace aktivně využívá akcelerátory v *FPGA* i některé v SoC NXP. Portace se však neomezuje pouze na tuto aplikaci. Bylo převedeno více než 10 balíčku z buildovacího systému *Buildroot* do buildovacího systému *Yocto*. Nově vznikly také další balíčky, jiné byly nahrazeny. Výsledkem portace je sada balíčků, ze které je možné sestavit obraz systému pro nově vytvořenou hardwarovou platformu založenou na *FPGA Intel Arria10* a síťovém procesoru *NXP LS2088*.

V rámci této práce byla upravována nejen softwarová část původní platformy Sprobe1G, popsané v sekci 2, ale i samotná platforma síťového procesoru *NXP QorIQ* a to především sběrnice *DPAA2*, popsaná v sekci 4.1. Bylo zapotřebí mimo jiné ladit nízkoúrovňové konfigurace, zavaděč i linuxové jádro.

Před portací byly vyhledány časově kritické operace a byla navržena a později implementována jejich akcelerace. Nově vzniklá platforma je většině případů více než 10× rychlejší. Nominální konektivita narostla 20×. Tato problematika byla diskutována v sekci 5.10.

V posledních fázi probíhalo intenzivní ladění, optimalizace a měření výkonnosti platformy a aplikace pro L7 analýzu *PaSt*. Tento software byl úzce svázán s jedno-gigabitovou platformou. Ve všech částech pro ovládání akcelerátorů došlo ke změnám, protože došlo ke změnám i v akcelerátorech. V původní verzi aplikace *PaSt* byl použit speciální framework

pro kopírování paketů do *user-space*, použití *DPDK* zcela změnilo způsob, kterým aplikace pracuje z pakety.

Platforma nabízí řadu možností, jak výkon dále navyšovat. Řada z těchto vylepšení zmíněných v sekci 5.11 závisí na výrobci procesoru a bude je možné realizovat nejdříve 2019 Q3. Některé optimalizace v konfiguračních rozhraních akcelerátorů v *FPGA* však bude možné provést již velmi brzy. Implementací těchto vylepšení se výkon může v ideálním případě zlepšit až o řád, naznačují to i syntetické testy v sekci 5.10.

Literatura

[1] Bonelli, N.; Giordano, S.; Procissi, G. A pipeline functional language for stateful packet processing. July 2017. 10.1109/NETSOFT.2017.8004226.

[2] Cisco Systems, I. : The Zettabyte Era: Trends and Analysis [online]. `https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.html`, 2017-6-7, [cit. 2018-5-17].

[3] devicetree.org : *Devicetree Specification, Release v0.2 [online]*. 20-12-2017, `https://github.com/devicetree-org/devicetree-specification/releases/tag/v0.2`.

[4] Falamarzi, R.; Bahrambeigy, B.; Ahmadi, M.; aj. High-performance multi/many-core network processing architectures with shared and private queues. May 2015. 10.1109/IKT.2015.7288757.

[5] González, A. *Embedded Linux Projects Using Yocto Project Cookbook*. EBL-Schweitzer, Packt Publishing, 2015. Dostupné z: <`https://books.google.cz/books?id=yNi6BwAAQBAJ`>. ISBN 9781784396343.

[6] Linux foundation : *Dokumentace DPDK [online]*. 2018, `https://dpdk.org/doc`.

[7] Linux foundation : *Manuálové stránky nástroje bridge [online]*. 2018, `http://man7.org/linux/man-pages/man8/bridge.8.html`.

[8] Nottingham, A.; Irwin, B. A high-level architecture for efficient packet trace analysis on GPU co-processors. Aug 2013. ISSN 2330-9881, 10.1109/ISSA.2013.6641052.

[9] NXP Semiconductors : *QorIQ SDK v2.0-1703 Documentation [online]*. 03-2017, `https://www.nxp.com/docs/en/supporting-information/QORIQ-SDK-2.0-IC-REV0.pdf`.

[10] NXP Semiconductors : *QorIQ LS2088A Reference Manual [online]*. 12-3-2018, `https://www.nxp.com/products/processors-and-microcontrollers/arm-based-processors-and-mcus/qoriq-layerscape-arm-processors/qoriq-layerscape-2088a-and-2048a-multicore-communications-processors:LS2088A`.

[11] NXP Semiconductors : *AN11979, DPAA2 Ethernet Switch and Edge Virtual Bridge - Application Note [online]*. 18-6-2017, `https://www.nxp.com/products/processors-and-microcontrollers/arm-based-processors-and-mcus/qoriq-layerscape-arm-processors/qoriq-layerscape-2088a-and-2048a-multicore-communications-processors:LS2088A`.

[12] NXP Semiconductors : *VORTIQA-NSP-ADK: VortiQa® Network and Security Package ADK [online]*. 23-3-2016, https://www.nxp.com/support/developer-resources/run-time-software/vortiqa-software-for-networking/vortiqa-applications-development-kits/vortiqa-network-and-security-package-adk:VORTIQA-NSP-ADK.

[13] NXP Semiconductors : *VortiQa Protocol Offload Package (POP) for QorIQ - Online Documentation [online]*. 23-3-2016, https://www.nxp.com/support/developer-resources/run-time-software/vortiqa-software-for-networking/vortiqa-applications-development-kits/vortiqa-protocol-offload-package-application-development-kit:VORTIQA-POP-ADK.

[14] NXP Semiconductors : *VORTIQA-SSP-ADK: VortiQa® Switch Supplementary Package Application Development Kit [online]*. 23-3-2016, https://www.nxp.com/support/developer-resources/run-time-software/vortiqa-software-for-networking/vortiqa-applications-development-kits/vortiqa-switch-supplementary-package-application-development-kit:VORTIQA-SSP-ADK.

[15] NXP Semiconductors : *Layerscape Software Development Kit 18.03 Documentation [online]*. 3-2018, https://www.nxp.com/support/developer-resources/run-time-software/linux-software-and-development-tools/layerscape-software-development-kit:LAYERSCAPE-SDK.

[16] NXP Semiconductors : *LS2088A-RDB: QorIQ® LS2088A Reference Design Board [online]*. 7-5-2018, https://www.nxp.com/support/developer-resources/software-development-tools/qoriq-developer-resources/qoriq-ls2088a-reference-design-board:LS2088A-RDB.

[17] NXP Semiconductors : *LS2088ASECRM, LS2088A Security (SEC) Reference Manual - Reference Manual [online]*. 7-5-2018, https://www.nxp.com/products/processors-and-microcontrollers/arm-based-processors-and-mcus/qoriq-layerscape-arm-processors/qoriq-layerscape-2088a-and-2048a-multicore-communications-processors:LS2088A?tab=Documentation_Tab.

[18] Velea, R.; Dragan, S. CPU/GPU Hybrid Detection for Malware Signatures. Sept 2017. 10.1109/COMAPP.2017.8079736.

[19] Yocto Project : *Yocto Project Reference Manual [online]*. 2018, https://www.yoctoproject.org/docs/.