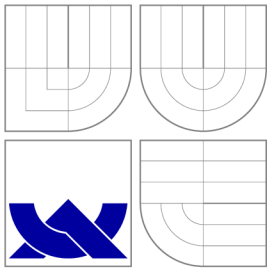
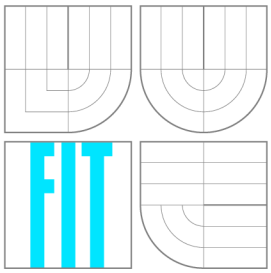


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

# STAVOVÉ ZPRACOVÁNÍ TCP/IP TOKŮ

STATEFULL PROCESSING OF TCP/IP FLOWS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — JAKUB OLBERT
AUTHOR

VEDOUCÍ PRÁCE — Ing. JAN KOŘENEK
SUPERVISOR

BRNO 2010

# Abstrakt

Rychlý vývoj počítačových sítí s sebou přináší potřebu tyto sítě zabezpečit proti stále pokročilejším útokům. Bezpečnostní systémy vyžadují pro svoji činnost pokročilou analýzu síťového provozu, která je prováděna na základě stavového zpracování toků. Zaměřením této bakalářské práce je návrh a simulace systému stavového zpracování toků. Navrhovaný systém využívá specializovaného hardware pro akceleraci zpracování síťového provozu vysokorychlostních páteřních linek. Specifickou vlastností systému je distribuce paměti toků mezi hardware a software. Vytvořený simulační model umožní otestování a optimalizaci systému stavového zpracování toků již ve fázi návrhu a tím usnadní případnou implementaci.



# Abstract

The fast development of computer networks brings the necessity to protect those networks against more and more advanced attacks. The security systems require an advanced analysis for their operation which is carried out based on the stateful processing of flows. This Bachelor Thesis focuses on the proposal and simulation of the stateful flow processing system. The proposed system uses a specialized hardware for network operation processing acceleration of high-speed backbone lines. The specific feature of the system is the flow memory distribution between the hardware and software. The created simulation model will make it possible to test and optimize the stateful flow processing system already in the phase of proposal and thus the possible implementation will be facilitated.



# Klíčová slova

počítačové sítě, stavové zpracování toků, simulace, FPGA

# Keywords

computer networks, statefull processing of flows, simulation, FPGA

# Citace

Jakub Olbert: Stavové zpracování TCP/IP toků, bakalářská práce, Brno, FIT VUT v Brně, 2010

# Stavové zpracování TCP/IP toků

## Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jana Kořenka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jakub Olbert
19. května 2010


## Poděkování

Chci poděkovat vedoucímu své bakalářské práce panu Ing. Janu Kořenkovi za jeho odbornou pomoc a také panu Ing. Martinu Žádníkovi za poskytnutí materiálů nezbytných pro vypracování této práce.

# Obsah

# Kapitola 1

# Úvod

V posledních letech můžeme pozorovat velký rozvoj v oblasti počítačové komunikace a to především díky Internetu. Internet je veřejné sdílené médium, což kromě jeho dostupnosti a univerzálního využití přináší také bezpečnostní rizika. S rostoucím počtem připojených uživatelů a množstvím přenášených dat je monitorování včetně zabezpečení počítačových sítí výpočetně velmi náročné. Zpracování dat, přenášených na multigigabitových rychlostech (10 Gbps a více), pomocí programového vybavení na běžně dostupných architekturách není možné kvůli limitované propustnosti systémové sběrnice a nedostačujícímu výkonu univerzálního procesoru. Řešením tohoto problému je využití specializovaného hardware pro akceleraci jednoduchých a časově náročných výpočtů.

Pro zajištění bezpečnosti počítačové sítě je často nezbytné *stavové* zpracování probíhající komunikace, kdy je na síťový provoz pohlíženo jako na množinu toků, které jsou analyzovány jako celky. Analýza jednotlivých paketů bez uvážení jejich kontextu neposkytuje dostatečné údaje pro správné vyhodnocení možné anomálie (např. síťového útoku). Použití stavového zpracování můžeme nalézt v systémech IDS (Intrusion Detection System), které vyhledávají v paketech vzory značící možný útok. Systém bez stavového zpracování toků nebude schopen odhalit hrozbu bude-li rizikový vzor rozdělen do více paketů, což tento detekční systém činí snadno prolomitelným.

Z předchozích odstavců je patrné, že pro zajištění bezpečnosti vysokorychlostních počítačových sítí je nezbytné použití stavového zpracování toků, které navíc važaduje specializovaný hardware pro akceleraci výpočtů. Vývoj tohoto systému probíhá v rámci projektu Liberouter [5], který je součástí výzkumného záměru CESNET Programovatelný hardware. Cílovou architekturou jsou karty rodiny COMBO [4] jež svým výkonem umožňují zpracování síťového provozu na přenosové rychlosti až 40 Gb/s. Cílem této práce je navrhnout vhodné rozdělení paměti síťových toků (dále *flow cache*) mezi hardware a software tak, aby bylo dosaženo maximální propustnosti a minimálních nároků na zdroje hardware.

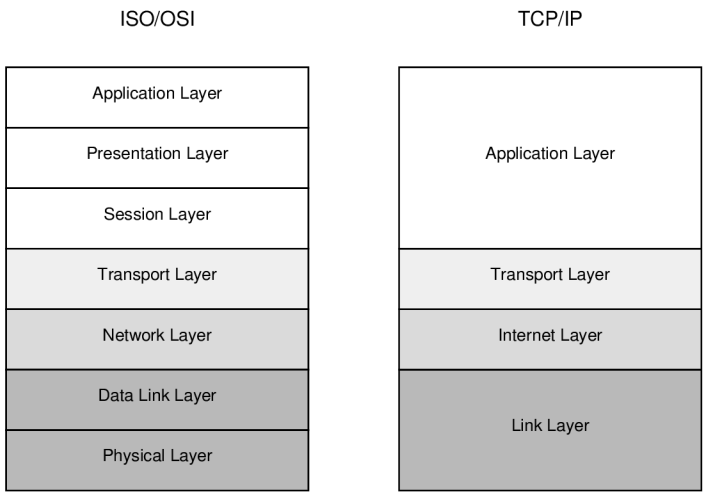
Dokument je logicky rozdělen do kapitol. V úvodní teoretické části (kapitola 2) je představen referenční síťový model ISO/OSI, na kterém je objasněn způsob identifikace a klasifikace síťových toků. Kapitola 3 pojednává o klasifikaci síťových toků a popisuje vybrané klasifikační algoritmy. Návrh systému stavového zpracování toků je předmětem kapitoly 4. Návrh a implementace simulačního modelu jsou popsány v kapitole 5. Závěr diskutuje získané poznatky a výsledky práce.

# Kapitola 2

# Úvod do počítačových sítí

## 2.1 Vrstvový model ISO/OSI

Referenční model OSI (Open System Interconnection Reference model) je abstraktní vrstvový model popisující vzájemné propojování různých systémů. Tento standard byl přijat mezinárodní organizací ISO (International Organization for Standardization) v roce 1984 jako norma ISO 7498 [2]. Úkolem tohoto modelu je usnadnit návrh architektury propojení systémů poskytnutím základny v podobě sedmi hierarchických vrstev, které jsou znázorněny na obrázku 2.1.




Obrázek 2.1: Referenční model ISO/OSI a model TCP/IP

Norma si neklade za cíl specifikovat implementaci komunikující systémů, ale uvádí všeobecné principy sedmivrstvé architektury – definuje rozhraní jednotlivých vrstev, popisuje jejich role a nabízené služby. Vzhledem ke své komplikaci se tento model využívá spíše jako referenční. Pro popis protokolů používaných v počítačových sítí se používá zjednodušeného modelu TCP/IP 2.2.

### Fyzická vrstva

Je nejnižší vrstvou modelu, která specifikuje komunikační médium a neuvažuje význam přenášených dat. Vrstva definuje všechny elektrické a fyzikální vlastnosti zařízení (napěťové úrovně reprezentující logické nuly a jedničky, tvary konektorů včetně rozmístění pinů, vlastnosti kabelů aj.). Hlavní rolí fyzické vrstvy je navazování a ukončování spojení s komunikačním médiem a převod digitální reprezentace dat na analogovou a zpět.

### Linková vrstva

Vytváří datový komunikační kanál mezi bezprostředními sousedními systémy. Strukturuje data přicházející z fyzické vrstvy do tzv. *rámců*. Řídí nastavení přenosové linky pomocí fyzické vrstvy a také řídí přístup ke sdílenému komunikančnímu médiu. Umí detekovat chybné rámce přenesené fyzickou vrstvou a zajistit jejich znovuposlání.

### Síťová vrstva

Zajišťuje přenos dat mezi koncovými komunikačními uzly s využitím směrování, odesilatel a adresát nemusí být přímo propojeni. Hlavním úkolem této vrstvy je správně směrovat data na jejich cestě sítí. K tomu je nutná jednoznačná identifikace komunikujících stran v rámci celé sítě, kterou nazýváme *síťová adresa*.

Síťová vrstva nabízí služby *spojované*, které zajišťují spolehlivý přenos dat a *nespojované*, které jsou založeny na principu *best effort delivery*. Nespojované služby nezaručují spolehlivý přenos dat, pouze se snaží data doručit s nejvyšším úsilím. Datovou jednotkou této vrstvy jsou *pakety*. Paket obsahuje ve své hlavičce obsahuje mimo jiné síťové adresy odesílatele a adresáta.

### Transportní vrstva

Tato vrstva vytváří komunikační kanál mezi procesy na jednotlivých uzlech sítě. Data z vyšších vrstev jsou zapouzdřena do *segmentů*, kterým je přiřazena adresa idenitifikující proces. Tato adresa je unikátní v daném systému (například číslo portu).

### Relační vrstva

Poskytuje mechanismy pro otevírání, uzavírání a správu relací mezi komunikujícími procesy. Pomocí těchto relací mohou procesy koordinovat svoji činnost. Příkladem může být realizace stavového modelu komunikace nad protokolem UDP, který je bezestavový.

### Prezentační vrstva

Role prezentační vrstvy spočívá v abstrakci aplikačních dat od jejich systémové reprezentace. Především se jedná o převody mezi kódováními, komprimace a šifrování dat.

### Aplikační vrstva

Nejvyšší vrstva modelu definuje aplikačně orientovné rozhraní nejbližší koncovému uživateli. Často bývá její součástí přímo implementace takových služeb jako například zasílání elektronické pošty, vzdáleného terminálového přístupu nebo přenosu souborů.

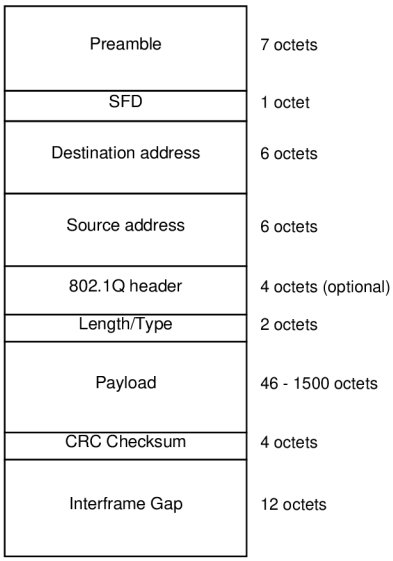
## 2.2 TCP/IP model

Tento model je odvozen od referenčního modelu ISO/OSI, ale nevyžaduje tak striktní dodržování vrstvové architektury. V RFC 1122 jsou definovány pouze čtyři vrstvy [13], které jsou znázorněny na obrázku 2.1.

### 2.2.1 Linková vrstva

Linková vrstva TCP/IP modelu zahrnuje v sobě vrstvy fyzickou a likovou referenčního modelu ISO/OSI. Nejpoužívanější protokoly linkové vrstvy pro drátové sítě jsou z rodiny Ethernet IEEE 802.3 [12]. Přístupu ke sdílenému médiu je v případě použití Ethernetových protokolů řešen metodou CSMA/CD (Carrier Sense Multiple Access with Collision Detection) [12].

Struktura Ethernetového rámce je znázorněna na obrázku 2.2.




Obrázek 2.2: Struktura Ethernetového rámce 802.3

**Preamble** – speciální značka skládající se vzoru (10101010), která zajišťuje synchronizaci přijímacího uzlu

**SDF (Start of Frame Delimiter)** – označuje začátek rámce. Preamble a SDF jsou využívany hardwarem pro rozpoznání nově příchozícho rámce a z příchozích dat jsou oříznuty již v rámci síťové karty (nejsou propagovány do vyšších vrstev).

**Destination address** – MAC adresa cílového uzlu

**Source address** – MAC adresa odesilatele

**802.1Q header** – volitelná značka 802.1Q standardu pro označení VLAN sítě [11]

**Length/Type** – význam této položky závísí na její hodnotě [16]. Hodnota menší nebo rovna 1536 (0x0600) znamená, že položka ubsahuje délku rámce (formát rámce IEEE

802.3), jehož datová část může dosahovat maximální délky 1500 oktetů. Hodnoty větší něž 0x0600 označují rámec formátu Ethernet II [12], kde tato položka identifikuje zapouzdřený paket vyšší vrstvy (0x0800 – IP paket, 0x0806 – ARP paket).

**Payload and padding** – zapouzdřená data protokolů vyšších vrstev. Položka *padding* (zarovnání, obsahuje libovolná data) je použita pro zarovnání v případě, že datová část rámce nedosahuje minimální délky 64 oktetů.

**CRC Checksum** – kontrolní součet datových položek rámce (nepočítá se pro samotný CRC Checksum)

**Interframe Gap** – mezipaketová mezera využitá zařízeními pro přípravu na přijetí dalšího rámce. Minimální délka mezery je doba přenosu 96 bitů ($9.6\mu s$ pro 10 Mbit/s Ethernet).

### 2.2.2 Internetová vrstva

Někdy též označovaná jako *síťová vrstva* [18], odpovídá síťové vrstvě referenčního modelu ISO/OSI. Jedná se o soubor protokolů které jsou využívány pro přenos paketů od zdrojového uzlu k přijímacímu uzlu (unicast) nebo skupině uzlů (multicast) přes komunikační síť. V této práci je nadále uvažován IP protokol, jehož definici lze nalézt v RFC 791 [21]. V současnosti je využívána verze protokolu IPv4, která je, kvůli téměř vyčerpanému adresovému prostoru, postupně nahrazována verzí IPv6 [14].

Hlavička IPv4 paketu je znázorněna na obrázku 2.3.

| bit/offset | 0 - 3 | 4 - 7 | 8 - 15 | 16 - 18 | 19 - 31 |
|---|---|---|---|---|---|
| 0 | Version | IHL | Type of Service | Total Length | |
| 32 | Identification | | | Flags | Fragment Offset |
| 64 | Time to Live | | Protocol | Header Checksum | |
| 96 | Source Address | | | | |
| 128 | Destination Address | | | | |
| 160 | Options | | | | |

Obrázek 2.3: Hlavička IPv4 paketu

**Version** – verze IP protokolu. Hodnota 4 značí IPv4 hlavičku.

**IHL (Internet Header Length)** – počet 32-bitových slov v IP hlavičce. Minimální hodnota je 5 (IP hlavička neobsahuje žádné volitelné položky *Options*).

**TOS (Type of Service)** – původně specifikace požadavků na kvalitu přenosu (QoS). Nově podle RFC 2474 [19] se položka nazývá *Differentiated Services*. Tento nový standard umožňuje lepší škálovatelnost, klasifikaci a správu služeb QoS.

**Total Length** – délka celého paketu v oktetech (včetně hlavičky). Minimální délka je 20 oktetů (pouze IP hlavička).

**Identification** – identifikační hodnota vložená odesílatelem, která je využita při rekonstrukci fragmentovaných paketů.

**Flags** – bitové pole popisující možnosti a stav fragmentace paketu. Bit *Don't Fragment* – zákaz fragmentace a *More Fragments* – další fragmenty.

**Fragment Offset** – hodnota určuje pozici fragmentu v ráci celého paketu (udáváno v osminásobcích oktetů)

**Time to Live** – zbraňuje nekonečnému přeposílání paketu mezi síťovými uzly (např. při zacyklení). Položka omezuje životnost datagramu v počtu přeposlání aktivním prvkem sítě jako je přepínač (switch) nebo směrovač (router).

**Protocol** – definuje vyšší protokol, který je zapouzdřen v datové části paketu. Seznam protokolů a jejich označení spravuje organizace IANA [7].

**Header Checksum** – kontrolní součet IPv4 hlavičky

**Source Address** – IPv4 adresa zdrojového uzlu

**Destination Address** – IPv4 adresa cílového uzlu (unicast), případně skupiny uzlů (multicast)

**Options** – volitelné položky IPv4 hlavičky. V případě, že délka této položky není dělitelná 32 bity je provedeno zarovnání přidáním libovolných dat označovaných jako *padding*.

### 2.2.3 Transportní vrstva

Transportní vrstva TCP/IP modelu odpovídá transportní vrstvě referenčního modelu ISO/OSI. Od referenčního modelu se liší například v možnosti použití nespojované služby (protokol UDP), přičemž ISO/OSI model definuje přenos dat na úrovni transportní vrstvy jako spolehlivý.

#### Transmission Control Protocol

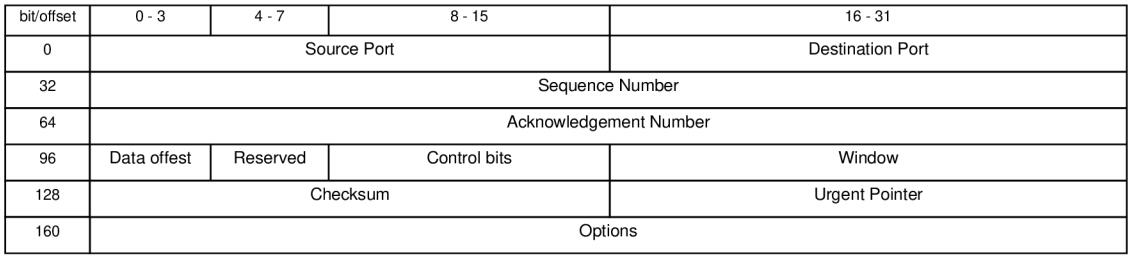
TCP protokol [22] využívá nespojovaného přenosového kanálu síťové vrstvy a proto musí zajistit spolehlivost přenosu dat ve vlastní režii. Tomu odpovídá celková robustnost a složitost tohoto protokolu (viz obrázek 2.4). Datové rámce TCP protokolu nazýváme *segmenty*. Mimo zabezpečení doručení všech segmentů ve správném pořadí protokol také poskytuje metody pro řízení toku (*flow control*).

**Source Port** (16 bitů) – identifikace komunikujícího procesu zdrojového uzlu pomocí čísla portu

**Destination Port** (16 bitů) – adresování procesu příjemce cílového uzlu číslem portu

**Sequence Number** (32 bitů) – pořadové číslo prvního datového oktetu segmentu. Pomocí tohoto identifikátoru je zajištěno případné přeskládání segmentů doručených v nesprávném pořadí.

**Acknowledgement Number** (32 bitů) – je-li nastaven příznak *ACK*, pak hodnota této položky je následující pořadové číslo segmentu, který příjemce očekává.




Obrázek 2.4: Hlavička TCP segmentu

**Data offset** (4 bity) – udává velikost TCP hlavičky v počtu 32-bitových slov

**Reserved** (6 bitů) – rezervováno pro budoucí využití. Tato hodnota musí být nulová.

**Control bits** (6 bitů) – bitové pole příznaků, které blíže specifikují typ segmentu. Mezi základní příznaky patří: SYN – ustavení TCP spojení, FIN – ukončení TCP spojení, RST – resetování spojení.

**Window** (16 bitů) – použito pro řízení toku

**Checksum** (16 bitů) – kontrolní součet počítaný z hlavičky a datové části TCP segmentu

**Urgent Pointer** (16 bitů) – v případě nastaveného příznaku *URG*, ukazuje tato položka na poslední oktet naléhavých dat

**Options** – obdobně jako u IPv4 hlavičky i hlavička TCP segmentu může obsahovat volitelné položky. RFC 793 [22] definuje dva formáty těchto položek. Jednooktetový typ položky a víceoktetový typ položky, který sestává z typu položky, délky položky a její hodnoty.

### User Datagram Protocol

Jednoduchý protokol, poskytující nespolehlivý přenos dat [20]. Protokol žádným způsobem nezabezpečuje doručení dat ani pořadí doručených datagramů – protokol je nespojovaný a bezestavový. Jeho výhodou je nízká režie a jednoduchost. Využití je například u bezestavových aplikačních protokolů jako DNS, u služeb vyžadujících rychlý přenos dat s tolerancí částečné ztráty přenášených datagramů jako jsou proudové přenosy hlasu (VoIP) a videa (IPTV). Struktura UDP datagramu je znázorněna obrázkem 2.5, kde je na první pohled patrné značné zjednodušení oproti hlavičce TCP segmentu (viz obrázek 2.4).

**Source Port Number** – identifikace komunikujícího procesu zdrojového uzlu pomocí čísla portu

**Destination Port Number** – adresování procesu příjemce cílového uzlu číslem portu

**Length** – délka celého UDP datagramu v oktetech. Minimální délka datagramu je 8 oktetů, což odpovídá délce UDP hlavičky.

| bit/offset | 0 - 15 | 16 - 31 |
|---|---|---|
| 0 | Source Port | Destination Port |
| 32 | Length | Checksum |
| 64 | Data | |

Obrázek 2.5: Struktura UDP datagramu

**Checksum** – kontrolní součet pro detekci chyb vzniklých během přenosu. Je počítán nad hlavičkou i datovou částí. V případě použití protokolu IPv4 na síťové vrstvě může být kontrolní součet ignorován a hodnota nastavena na 0. Při použití protokolu IPv6 je tato položka povinná.

# Kapitola 3

# Identifikace a klasifikace síťových toků

## 3.1 Identifikace síťových toků

Základním problémem systému stavového zpracování toků je tyto toky v síťovém provozu identifikovat. Síťový *tok* (dále jen tok) je podmnožina síťového provozu, jejíž pakety mají stejné vlastnosti, které označujeme jako *klíč* toku (např. zdrojová a cílová IP adresa, použitý transportní protokol, číslo zdrojového a cílového portu). Tok mohou představovat pakety zasílané mezi dvěma koncovými uzly sítě, pakety zasílané z jedné podsítě do jiné, pakety zasílané specifickým transportním protokolem a podobně.

Vhodným zvolením klíče dosáhneme rozdělení síťového provozu na jednotlivé toky, které budeme dále analyzovat. Z hlediska získávnání základních statistických informací je možné zvolit obecnější klíč a počet toků tak bude menší (větší míra abstrakce). Například poskytovatel připojení k internetu účtuje služby svým zákazníkům podle množství přenesených dat. Naproti tomu IDS systém monitorující páteřní linku potřebuje informace o probíhající komunikaci co nejpřesnější, aby mohl identifikovat zdroj potenciální hrozby. V tomto případě zvolíme konrétnější klíč, který obsahuje zdrojovou a cílovou IP adresu, číslo zdrojového a cílového portu a protokol transportní vrstvy, čímž definujeme tok jako komunikaci dvou procesů mezi dvěma koncovými uzly sítě.

Různé aplikace využívající stavového zpracování toků požadují rozdílné stanovení klíče pro identifikaci toků. V této práci bude nadále jako klíč toku uvažována pětice zdrojové a cílové IP adresy, čísla zdrojového a cílového portu a použitý transportní protokol (TCP, UDP).

## 3.2 Klasifikace síťových toků

Rozdělení síťového provozu na jednotlivé toky je základním způsobem kvantitativní klasifikace. Pouhé rozdělení na toky nám neumožňuje hodnotit síťový provoz z hlediska kvalitativního, kdy například porovnáváme jednotlivé toky mezi sebou a vybíráme z nich podmnožinu na základě určité vlastnosti (toky tvořené 100 a více pakety, toky trvající v čase déle než 1 minutu ap.).

Klasifikace síťových toků je nezbytným prostředkem využívaným systémy stavového zpracování toků, který umožňuje separaci a selekci požadovaných údajů z velké množiny dat, kterou představuje provoz páteřních síťových linek. Právě velké množství dat, které je

potřeba zpracovat ve velmi krátkém čase, představuje hlavní problém systémů stavového zpracování toků [15].

Například pro plně zatíženou 10 Gbps linku je průměrný počet přenesených paketů za 1 sekundu 1.644 mil., kde tyto pakety představují 94720 toků [23], přičemž hodinový provoz je tvořen $6 \cdot 10^9$ pakety a $340 \cdot 10^6$ toky.

Měřící zařízení mohou ukládat velké množství informací s využitím pamětí DRAM, které disponují dostatečnou kapacitou a jejich cena je nízká. Nevýhodou těchto pamětí je ovšem nízká rychlost čtení a zápisu, která se ročně průměrně zvýší o 7–9%, naproti tomu rychlost síťových linek roste průměrně o 100% každý rok [15]. Použití těchto pamětí v systémech stavového zpracování toků znemožňuje provádění analýzy bez ztráty paketů, což vnáší chyby do výsledků měření a zvyšuje riziko prolomení bezpečnostních systémů.

Řešení se nabízí v použití statických pamětí SRAM, které disponují menší kapacitou, ale poskytují větší datovou propustnost. Nižší kapacita je důsledkem vysokých výrobních nákladů – SRAM o stejné kapacitě jako DRAM je mnohonásobně dražší. Menší kapacita SRAM neumožňuje ukládat informace o veškerém síťovém provozu, ale pokud je vhodně zvolena pouze jeho podmnožina jsme schopni ji zpracovat beze ztráty paketů a to i na vysokých přenosových rychlostech (10 Gbps).

Otázkou je, jak vhodně zvolit sledovanou podmnožinu síťového provozu. Na obrázku 3.1 je znázorněn výsledek analýzy vzorku dvouhodinového síťového provozu T1 linky. Vzorek provozu byl získán z webových stránek projektu MAWI [6] a analýza byla provedena s použitím knihovny OTA[1]. Horní graf zobrazuje jednotlivé toky sestupně seřazené podle počtu paketů. Spodní graf vyjadřuje procentuální podíl toků na celkovém počtu paketů. Toto měření poskytuje zajímavý pohled na rozdělení síťového provozu na toky, kde prvních přibližně 20000 toků z celkových 102311 představuje 80% z celkového počtu paketů. Tyto toky můžeme označit jako tzv. *silné toky*. Velké množství toků (více jak 80%) je tvořeno maximálně 10 pakety. Na základě tohoto poznatku můžeme prohlásit, že zpracováním 20% nejsilnějších toků zpracujeme 80% síťového provozu.

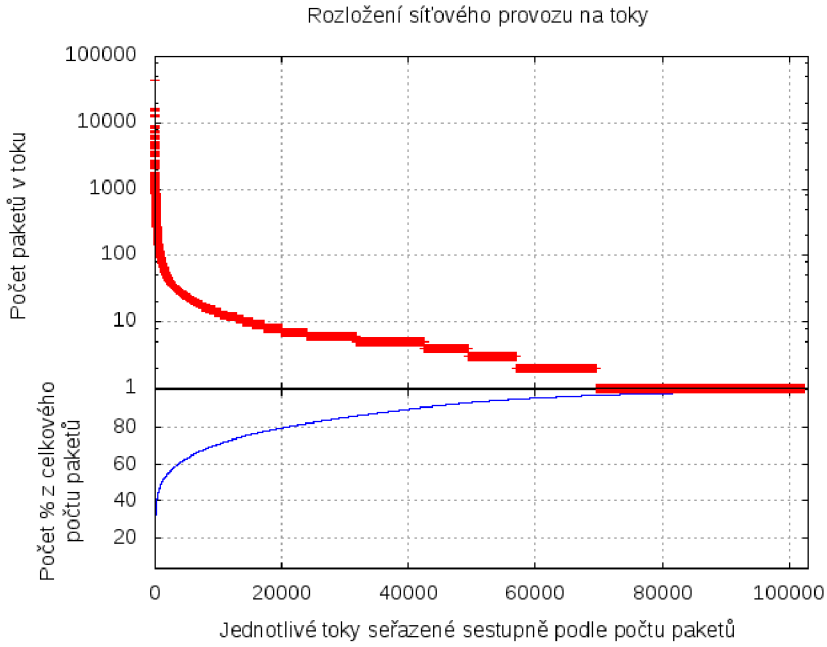
Z předchozích odstavců je patrné, že systémy stavového zpracování toků operující na páteřních síťových linkách nemohou být řešeny pouze softwarově na běžně dostupných platformách PC. Výkonnost těchto architektur je limitována propustností systémové sběrnice, rychlostí operační paměti RAM a výkonností univerzálního procesoru. Při použití vysoce optimalizovaného software lze dosáhnout zpracování až 3.3 mil. paketů za sekundu [23], což ovšem nepostačuje pro zpracování teoretického maxima 10 Gbps linky, které je přibližně 15 mil. paketů za sekundu pro nejkratší pakety[2].

Řešení tohoto problému se nabízí v použití specializovaného hardware, který bude akcelerovat jednoduché a časově kritické výpočty a případně provede agregaci vstupních dat. Tím se redukuje objem přenášených dat přes systémovou sběrnici a sníží se zátěž na PC, jehož SW vybavení bude možné použít pro složité výpočty, jejichž provední není časově kritické. Jako vhodný prostředek se nabízí využití programovatelných hradlových polí FPGA. Tyto čipy poskytují dostatečný výpočetní výkon a navíc umožňují rekonfiguraci podle konkrétních požadavků na zpracování síťového provozu.

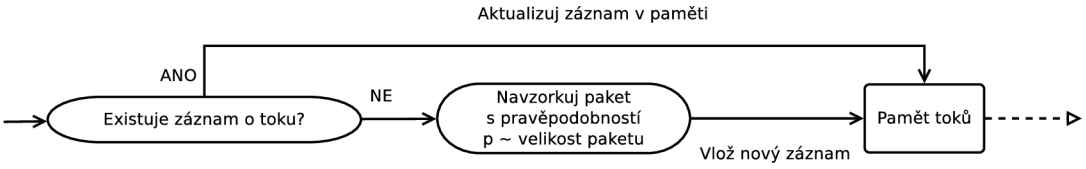
Tato práce se zabývá distribucí paměti toků (dále *flow cache*) mezi software a hardware tak, aby byl zpracován síťový provoz na vysokých přenosových rychlostech (10 Gbps a více) beze ztráty paketu. Cílová architektura specializovaného hardware jsou karty rodiny COMBOv2 [4].

[1] Offline Traffic Analysis – knihovna napsaná v jazyce Python a poskytující nástroje pro analýzu souborů zachytaného síťového provozu

[2] Nejkratší pakety o délce 84 B (včetně mezipaketové mezery)




Obrázek 3.1: Analýza síťového provozu T1 linky




Obrázek 3.2: Znázornění detekčního algoritmu Sample and Hold

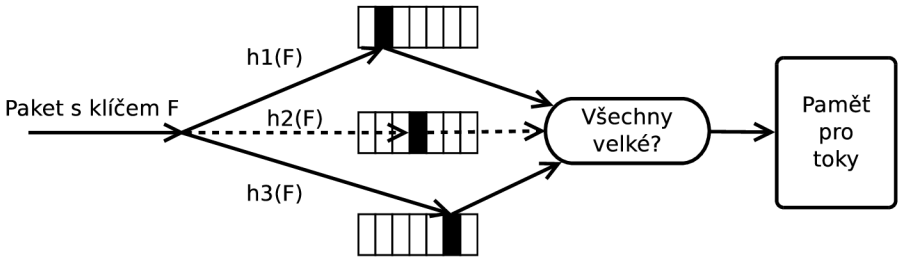
## 3.3 Algoritmy pro detekci intenzivních toků

### 3.3.1 Sample and Hold

První metoda popsaná v [15] určuje silné toky na základě vzorkování paketů. Vzorkování paketů je nejjednodušší způsob detekce silných toků, který ovšem vnáší velkou relativní chybu do výsledků měření. Tuto chybu lze vyjádřit vztahem $e = \frac{1}{\sqrt{M}}$, kde $M$ je množství paměti využívané měřící sondou pro uložení toků. Pro snížení relativní chyby je potřeba zvýšit kapacitu paměti, což z hlediska dané hardwarové platformy a jejích omezených zdrojů není možné.

Metoda Sample and Hold se snaží řešit problém nepřesnosti vylepšením klasického vzorkování paketů. Příchozí pakety jsou vzorkovány podle pravděpodobnosti závislé na délce paketu a pokud je paket vybrán, je vložen záznam o jemu příslušejícím toku do paměti. Jakmile je záznam pro daný tok v paměti pak všechny příchozí pakety patřící do tohoto toku jsou použity pro aktualizaci záznamu. Pro jednou navzorkovaný tok jsou už zpracovány všechny příchozí pakety. Algoritmus Sample and Hold je znázorněn nákresem 3.2.

Klasifikační metoda Sample and Hold není absolutně přesná v detekci silných toků a při jejím použití mohou nastat dva chybové stavy. První chybou je označení slabého toku




Obrázek 3.3: Znázornění detekční metody Multistage Filters

jako silný tok, kdy se navzorkuje paket nepatřícího do silného toku. Záznam o slabém toku pak zabírá místo v paměti určené pro silné toky. Druhou a závažnější chybou je nedetekování silného toku, kdy se nenavzorkuje žádný paket silného toku a tento silný tok je ignorován. Pravděpodobnost výskytu těchto chyb je velmi závislá na vhodném zvolení vzorkovací pravděpodobnosti, což lze například provést algoritmem popsaným v [15].

Metoda Sample and Hold mnohonásobně snižuje chybu měření oproti klasickému vzorkování paketů, ale její nevýhodou je přístup do paměti pro každý příchozí paket – výkonnost systému založeného na této metodě je limitována datovou propustností použité paměti.

Přesná specifikace této metody a její matematický popis není předmětem této práce a lze je nalézt v [15].

### 3.3.2 Multistage Filters

Druhá metoda popsaná v [15] nabízí přesnější detekci silných toků než Sample and Hold a navíc zamezuje chybě nedetekování silného toku. Metoda Multistage Filters je založena na tabulce čítačů, které nesou informaci o počtu přenesených bajtů pro daný tok. Silné toky jsou pak vybrány na základě počtu bajtů získaného z tabulky. Pro adresaci čítače se používá hash klíče toku. Problém namapování více toků na stejný čítač je řešen použitím několika tabulek s čítači, kdy pro adresaci v každé tabulce je použita jiná hashovací funkce. Tok je prohlášen za silný teprve v případě, kdy jemu odpovídající čítače ve všech tabulkách překročí požadovanou mezní hodnotu.

Přesnost metody detekce záleží na vhodném zvolení počtu tabulek, počtu čítačů v každé z tabulek a také na určení prahové hodnoty počtu bajtů.

Tato metoda umožňuje vznik chyby, kdy je slabý tok označen jako silný. To může nastat v případě kdy se skupina slabých toků namapuje na stejné čítače ve všech tabulkách a hodnoty čítačů pak překročí nastavenou mez. Obdobný problém nastává když se slabý tok namapuje na čítače jincých silných toků a je pak neprávem označen také jako silný.

Výhodou této metody oproti Sample and Hold je větší přesnost detekce a eliminace problému nedetekování silného toku. Nevýhodou je větší paměťová náročnost jak kapacitní (uložení tabulek s čítači), tak v počtu přístupů do paměti (inkrementace a porovnání čítačů).

Přesnou specifikaci metody lze opět nalézt v [15].

Metody Sample and Hold a Multistage Filters pracují periodicky v časových intervalech, kdy po skončení intervalu dojde k exportování všech záznamů z paměti toků a v případě Multistage Filters také k vynulování všech čítačů. V příští periodě tak detekce vychází opět z výchozího stavu. Metody totiž nenabízejí jinou možnost odstraňovaní zastaralých a slabých toků, které jsou uloženy v paměti a zabírají tak místo nově vzniklým silným tokům.

Tento způsob „regenerace“ paměti toků je nenáročný z hlediska implementace, ale přináší řadu komplikací, které mají nepříznivý dopad na celkovou výkonnost a přesnost detekčního systému.

Jedním z problémů je ztráta počátečních paketů silných toků na začátku intervalu. Tyto pakety jsou použity pouze pro detekci, ale nejsou dále zpracovány, což způsobuje nepřesnost výsledků měření. Tuto zápornou vlastnost se snaží řešit některé optimalizace popsané v [15] (např. *Preserving Entries* – ponechání záznamů o silných tocích v paměti i po skončení měřícího intervalu).

Dalším problémem je export stávajících záznamů z paměti toků. To je potřeba řešit tak, aby nedošlo k ovlivnění výkonnosti systému, snížení datové propustnosti paměti pro nově příchozí pakety, způsobené vyčítáním stávajících záznamů.

Výhodami těchto metod jsou jednoduchost implementace, a tedy snadná realizace v hardware, a vysoká přesnost detekce silných toků. Jejich hlavní nevýhodou je neschopnost zpracovat počáteční pakety silných toků, které jsou použity pouze pro detekci. Přitom počáteční pakety nesou informace o ustavení spojení a jsou tak velmi důležité z hlediska monitorování síťového provozu.

### 3.3.3 Detekce silných toků s využitím flow cache

Třetí způsob detekce silných toků není přímo klasifikační metoda jako již zmíněné Sample and Hold a Multistage Filters. Jedná se o princip správy paměti toků, jehož přirozenou vlastností je upřednostnění silných toků před slabými. Tato metoda s ohledem na implementaci v FPGA byla představena na 19. mezinárodní konferenci FPL (Field Programmable Logic and Applications) v Praze roku 2009 [24].

Paměť flow cache není chápána jako lineární struktura, ale má podobu tabulky s pevně danými rozměry (n-cestná cache). Řádek představuje sekvenci záznamů o tocích. Adresa řádku je hash klíče toku. Pro přístup ke konkrétnímu toku je potřeba adresovat řádek a ten pak sekvenčně projít a vybrat požadovaný záznam.

Hlavním úkolem správy flow cache je udržování záznamů v rámci řádku. Metoda je založena na vhodném určení pozice pro vložení nového záznamu, na výběru stávajícího záznamu pro odstranění a na přesunu záznamu při jeho aktualizaci. Při pokusu vložení nového záznamu do již plného řádku je potřeba některý ze stávajících toků odstranit. Při aktualizaci záznamu dojde k jeho přesunutí o určitý počet pozic směrem k začátku řádku. Cílem je přesunout silné toky na začátek řádku, aby byly při sekvenčním procházení nalezeny jako první a slabé toky odsunout na konec řádku odkud jsou odstraňovány.

Na rozdíl od předchozích detekčních metod nedochází k zahazování počátečních paketů toků. Navíc tato metoda nepracuje v časových intervalech, takže odpadají problémy spojené s hromadným exportem záznamů na konci intervalu. Nově se ale objevuje problém stárnutí toků v paměti, který je potřeba řešit sledováním neaktivního timeoutu. Implementace je složitější a paměťová náročnost mnohem vyšší než u předchozích metod.

# Kapitola 4

# Návrh systému stavového zpracování toků

Tato kapitola se zabývá návrhem systému stavového zpracování toků, který využívá distribuci flow cache mezi hardware a software. V úvodní části bude představena cílová platforma, pro kterou je systém navrhován. V dalších částech bude prezentována myšlenka rozdělení flow cache mezi HW a SW včetně různých možností správy. Závěr kapitoly je věnován návrhu architektury.

## 4.1 Cílová platforma

Cílovou platformou jsou hardwarové karty rodiny COMBO (aktuálně COMBOv2) [4] vyvýjené v rámci projektu Liberouter [5], který je výzkumným záměrem společnosti CESNET [9]. Tyto karty jsou osazeny výkonnými FPGA čipy Virtex 5 od firmy Xilinx [10]. Dále je na nich k dispozici dvojice statických pamětí QDR II RAM do výrobce Cypress [3], každá o kapacitě 8 MB. Paměti QDR jsou využity jako rychlé vyrovnávací paměti s minimální latencí, jejich teoretická datová propustnost je až 18 Gbps. Komunikace se softwarem probíhá přes PCI Express sběrnici, jejíž teoretická datová propustnost je až 16 Gbps.

## 4.2 Rozdělení flow cache mezi HW s SW

Jak již bylo uvedeno v předchozí kapitole, tato práce se zabývá návrhem systému stavového zpracování toků pro nasazení na páteřních síťových linkách s využitím specializovaného hardware pro akceleraci výpočtů. Na navrhovaný systém jsou kladeny vysoké nároky ohledně rychlosti zpracování dat, kdy při monitorování plně zatížené linky nesmí dojít k zahazování paketů, aby byly výsledky měření maximálně přesné.

Cílová architektura karet COMBO by měla být schopná zpracovat síťový provoz až na rychlostech 42 Gbps[1] pro nejkratší pakty [23]. Toto omezení je dáno propustností použité statické paměti QDR II. Maximální propustnost do softwaru byla aproximována na 7.68 Gbps [23]. Z výše uvedených poznatků vyplývá, že je potřeba dosáhnout minimální agregace $\frac{40}{7.68} = 5.21$, aby byly zpracovány nejkratší pakety na plně vytížené 40 Gbps lince.

Navrhovaný systém provádí základní agregaci příchozích paketů jejich rozdělením do jednotlivých toků, jejichž záznamy ukládá do rychlé vyrovnávací paměti QDR. S příchozím

[1] Pro délku paketu 84 bajtů a délku záznamu o toku 32 bajtů.

paketem je aktualizován záznam přečtený z QDR a zpracování tak probíhá pouze v rámci karty a tím je snížena zátěž na softwarovou část. Problémem je malá kapacita statických pamětí, které nemohou uchovávat záznamy o všech tocích. Toto je řešeno použitím klasifikační metody, která vybere ze síťového provozu pouze nejsilnější toky, které budou zpracovány na kartě. Zbylé slabé toky jsou přeposílany ke zpracovaní do softwaru. Protože silné toky tvoří většinu síťového provozu, nebude objem dat přenášených po systémové sběrnici tak velký a nedojde k zahlcení hostitelského počítače.

Z předcházejících odstavců je patrné, že dojde k rozdělení flow cache mezi hardware s software. Použití distribuované flow cache přináší problémy jako je správa oddělených pamětí včetně zajištění konzistence uložených záznamů a umístění klasifikace do HW nebo do SW části systému.

### 4.2.1 Správa distribuované flow cache

Hlavním úkolem správy distribuované flow cache je zajištění konzistence dat tak, aby uložené záznamy o tocích vždy obsahovaly platné údaje. K porušení konzistence může dojít například při chybně provedném exportu záznamů z hardwarové cache do softwaru, kdy se tyto záznamy částečně překrývají a jejich spojením dojde k poškození záznamu. K zneplatnění dat může také dojít vlivem zpoždění propagace záznamů, kdy během přenosu aktuálního záznamu po systémové sběrnici dojde k aktualizaci zastaralého záznamu v paměti nově příchozím paketem.

## 4.3 Umístění klasifikační jednotky

Klasifikační jednotka může být umístěna jak v hardwaru, tak v softwaru. Obě možnosti mají své výhody a nevýhody. Při umístění v hardwaru je výsledek klasifikace okamžitě dostupný pro správce hardwarové flow cache a reakce na nově detekovaný silný tok je velmi rychlá. Nevýhodou jsou omezené zdroje čipu, což může snížit přesnost klasifikační metody.

Implementace v softwaru poskytne velké množství paměti dostupné pro klasifikační metodu a výhody univerzálního procesoru pro složitější výpočty a tím je dosaženo větší přesnosti klasifikační metody. Nevýhodou softwarového řešení je zpoždění propagace aktualizací o silných tocích do hardwaru, kde probíhá jejich agregace, což může způsobit snížení efektivnosti celého systému.

Odpověď na tuto otázku by měl přinést simulační model, kde budou testovány obě varianty umístění.

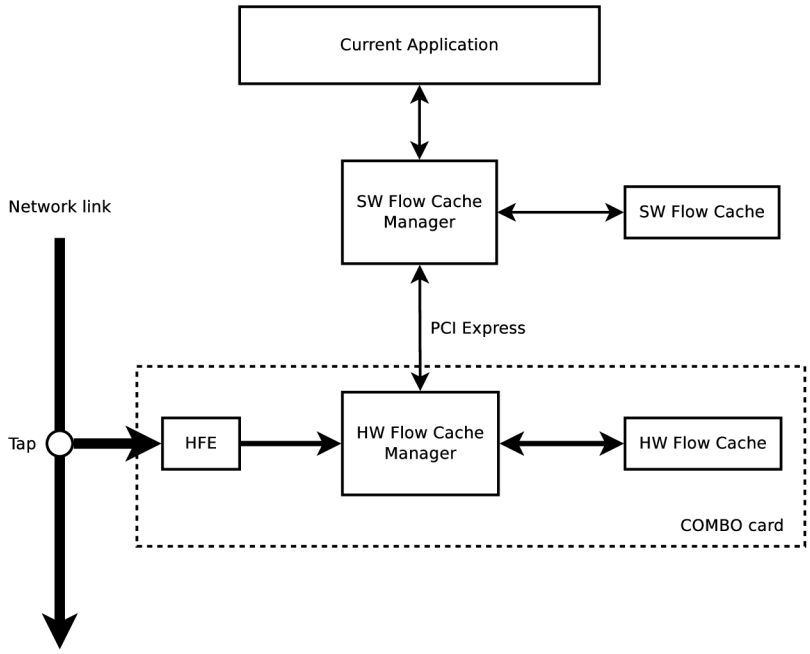
## 4.4 Návrh architektury

Architektura navrhovaného systému je znázorněna na obrázku 4.1. Jednotlivé komponenty jsou popsány dále v dokumentu.

### 4.4.1 Header Field Extractor

Provádí extrakci položek z hlaviček paketů. Získaná data jsou použita pro vytvoření UH hlavičky[2], která ukládá podmnožinu atributů paketu nezbytných pro stavové zpracování toku (např. zdrojová a cílová adresa, číslo zdrojového a cílového portu, použitý transportní protokol a časová značka).

[2] *Unified Header* – definice a označení bylo převzato z projektu Liberouter [5].




Obrázek 4.1: Pohled na celkovou architekturu monitorovací sondy

Tato komponenta je vyvýjena v rámci projektu Liberouter [5] a je dostupná její hotová implementace.

### 4.4.2 Flow cache

Pamět uchovávající záznamy o síťových tocích. V rámci systému na ni nejsou kladeny žádné speciální požadavky. Jak již bylo uvedeno výše, hlavními rozdíly mezi hardwarovou a softwarovou flow cache pamětí je jejich kapacita a latence. Statická paměť QDR, která tvoří hardwarovou flow cache, je použita jako rychlá vyrovnávací paměť pro časově kritické aktualizace záznamů silných toků. Naproti tomu softwarová cache umístěná v operační paměti hostitelského počítače má delší dobu přístupu, ale mnohosábně vyšší kapacitu a je vhodná pro uložení většího počtu záznamů, jejichž aktualizace nejsou tak časté.

### 4.4.3 Flow Cache Manager

Hlavní komponenta systému, provádí klasifikaci síťového provozu, řeší stavové zpracovaní toků a spravuje záznamy v paměti flow cache. Vnitřní implementace této komponenty závisí na použité klasifikační metodě, na umístění komponenty (HW nebo SW) a také na umístění klasifikační metody.

Obecně Flow Cache Manager přijímá UH hlavičky a aktualizuje záznamy odpovídajících toků v paměti flow cache. Pokud navíc implementuje klasifikační metodu, tak vkládá nové záznamy o tocích do paměti flow cache v případě, že byl detekován nový silný tok. V případě, že se namapují různé toky na stejnou adresu paměti flow cache, je proveden export stávajícího záznamu, aby bylo uvolněno místo pro nový silný tok.

Softwarový Flow Cache Manager musí být navíc schopen zpracovávat záznamy o tocích exportovaných z hardwarové paměti flow cache. Pokud je klasifikační metoda implemento-

vána v SW Flow Cache Manageru, pak tento musí spravovat hardwarovou paměť flow cache zasíláním aktualizací informujících o silných tocích. V tomto případě musí HW Flow Cache Manager vkládat nové silné toky do hardwatové paměti flow cache na základě aktualizací obdržených ze SW.

# Kapitola 5

# Návrh a implementace simulačního modelu

Tato kapitola popisuje návrh a implementaci simulačního modelu systému představeného v kapitole 4. Impulsem k vytvoření simulačního modelu je možnost jeho využití k ověření správnosti návrhu modelovaného systému před jeho samotnou implementací. Simulační model by měl také poskytnout detailní pohled na chování celého systému, což umožní odladění a optimalizaci již na úrovni návrhu a fáze implementace tak bude jednodušší.

Pro vývoj bylo zvoleno implementační prostředí skriptovacího jazyka Python, který se díky své široké nabídce vyšších abstraktních datových struktur a možnosti použití objektového paradigma jeví jako ideální pro tvorbu simulačního modelu.

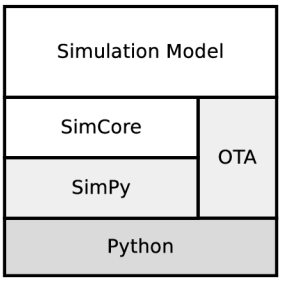
Na obrázku 5.1 je zachycena architektura modelu z pohledu použitých knihoven a vztahů mezi nimi. Následuje popis jednotlivých vrstev.

**Python** – použitý programovací jazyk

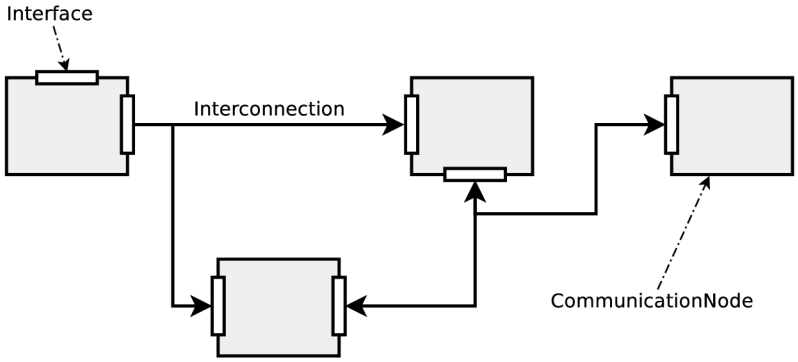
**OTA** – Offline Traffic Analysis, knihovna napsaná v jazyce Python, která usnadňuje čtení dat ze souborů zachyceného síťového provozu a nabízí základní prostředky k jejich analýze.

**SimPy** – Simulation in Python, objektově orientované prostředí pro popis diskrétních simulačních modelů založených na procesech [8].

**SimCore** – Knihovna pro podporu návrhu hardwarových komponent a jejich propojení, implementována v rámci této práce.




Obrázek 5.1: Vrstvová architektra simulačního modelu.




Obrázek 5.2: Systém komunikačních uzlů a propojů.

**Simulation Model** – vlastní simulační model postavený na knihovně SimCore a využívající knihovny OTA.

Následující část je zaměřena na popis knihovny SimCore. V druhé části kapitoly je pak představen simulační model systému stavového zpracování toků.

## 5.1 Knihovna SimCore

Hlavním cílem knihovny SimCore je usnadnění modelování hardwarových komponent a jejich vzájemné komunikace. Ačkoliv byla knihovna implementována pro konkrétní použití v rámci této práce, její využití může být obecné.

Základní filosofií je pohled na modelovaný systém jako na množinu entit, které spolu komunikují zasíláním zpráv. Každá entita je popsána svými rozhraními a svým chováním. Entity spolu komunikují prostřednictvím svých rozhraní, jež musí být navzájem propojena – obdobně jako hardwarové komponenty propojené pomocí vodičů. Ukázka systému propojení entit je na obrázku 5.2. Toto pojetí může být chápáno jako graf, kde množina uzlů je množinou komunikačních uzlů a množina hran je množina propojů.

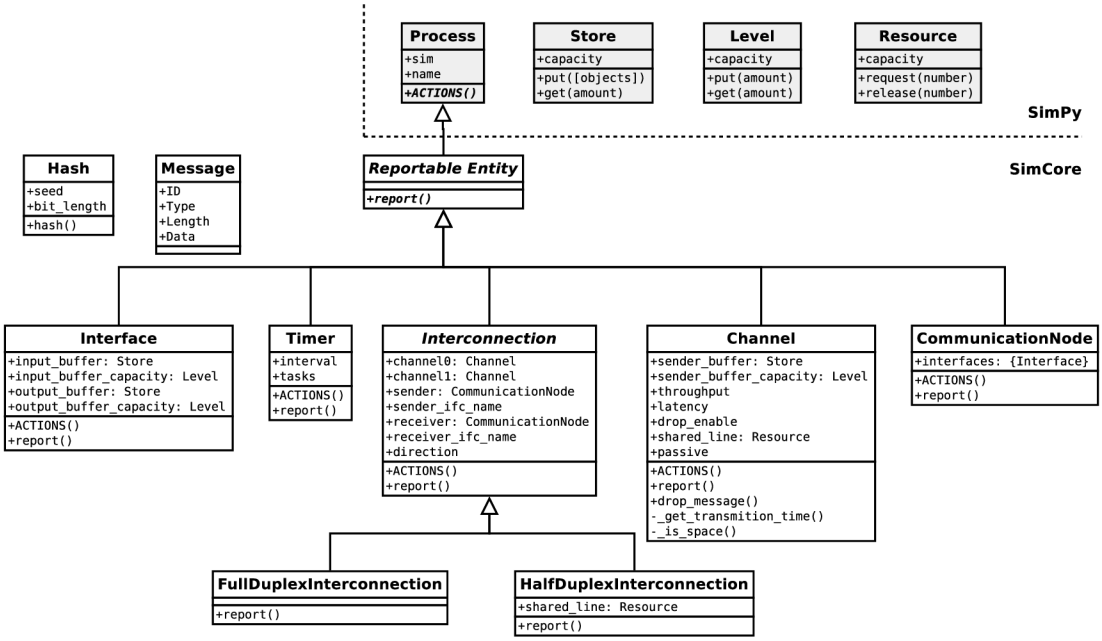
Tato knihovna je implementována v prostředí SimPy, které nabízí objektově orientovaný přístup k diskrétní simulaci založené na procesech. Nejdůležitější třídy knihovny SimPy použité pro implementaci SimCore jsou ve zkratce popsány následujícím výčtem, kde je také uvedeno jejich použití v SimCore. Kompletní dokumentace prostředí SimPy včetně dokreslujících příkladů lze nalézt na webových stránkách projektu SimPy [8] v sekci *Documentation*.

**Simulation** – třída zabalující simulační experiment, spuštění simulace, nastavení doby běhu simulace. Tato třída řídí průběh simulace posouváním modelového času a správou kalendáře událostí.

**Process** – aktivní objekt simulace. Třídy odvozené musí definovat PEM metodu (Process Execution Method), která popisuje chování objektu v rámci běhu simulace.

**Resource** – modelování zařízení se sdílenou frontou (např. obslužná linka).

**Level** – obdobně jako Resource. Zařízení modelující produkci a spotřebu homogeního materiálu (např. palivová nádrž nebo kapacita datového bufferu).




Obrázek 5.3: Zjednodušený diagram hierarchie tříd knihovny SimCore.

**Store** – další zástupce zařízení. Modeluje sklad objektů, které jsou obecně různého typu (např. datový buffer ukládající různé typy zpráv).

Obrázek 5.3 reprezentuje zjednodušený diagram hierarchie tříd knihovny SimCore s návazností na prostředí SimPy. Význam zobrazených tříd je dokumentován níže.

Důležitým modulem knihovny SimCore je konfigurační modul knihovny implementovaný v souboru `Configuration`. Nejdůležitějším nastavením je pak konstanta `TIME_UNIT`, která definuje základní jednotku modelového času. Konstanta udává počet časových jednotek tvořících 1 sekundu. Implicitní hodnota je $1 \cdot 10^9$, což představuje 1 ns.

### Reportable Entity

Abstraktní třída odvozená z třídy `Process` prostředí SimPy, která deklaruje metodu `report()`, jež je použita pro získání informací o aktuálním stavu objektu. Tato třída byla zavedena především z důvodu testování knihovny během vývoje, ale je dobře využitelná i během ladění simulačního modelu vyvýjeného nad knihovnou SimPy.

Důležitou součástí modulu `ReportableEntity` je seznam všech instancí této třídy (`SimulationEntities`). Tento seznam je využit pro hromadnou aktivaci všech simulačních komponent před spuštěním simulace. Simulačních komponent může být obecně velké množství a jejich ruční aktivace by byla pracná zvýšilo by se tak riziko opomenutí aktivace některé z komponent, což by vedlo k nedefinovánemu chování simulačního modelu, které by se špatně odhalovalo.

### Hash

Tato třída zabaluje výpočet hashovací funkce. Metoda `hash(message)` vrací hash spočítanou pro zprávu `message`. Třída byla zavedena z důvodu požadavku na výpočet hashí různé

bitové délky, kterou lze jednoduše měnit nastavením atributu `bit_size`.

### Message

Veškeré zprávy zasílané mezi rozhraními prostřednictvím propojů musí být objekty třídy `Message`. Tento pevně stanovený formát umožňuje jednotnou implementaci všech propojů. Systém zpráv navíc umožňuje snadné protokolování komunikace a řeší problém adresace komponent (v případě nepřímé komunikace). Následuje popis jednotlivých atributů.

**ID** – identifikátor zprávy

**type** – typ zprávy, určuje použitý protokol

**data** – přenášená data

**length** – délka zprávy v bitech. Určuje potřebnou kapacitu pro uložení v bufferu a také je použita pro výpočet doby přenosu po propoji.

V rámci simulačního modelu je ustavena množina protokolů, které přesně definují obsah jednotlivých položek zprávy. Tím je zajištěno, že naprosto rozdílné objekty jsou schopné vzájemné komunikace na základě dodržení daného protokolu.

Pokud probíhá komunikace uzlů nepřímo, například prostřednictvím sdílené sběrnice, poskytují zprávy jednoduchou možnost adresace uzlů s využitím kombinací atributů `ID` a `type`.

### Communication Node

Komunikační uzly tvoří obálku pro hardwarové komponenty. Tyto objekty jsou rozhodující z pohledu simulačního modelu, protože implementují funkčnost celého modelu. Z pohledu knihovny jsou to pouze objekty, které jsou popsány svými rozhraními a implementace chování uzlu je ponechána na simulačním modelu postaveném nad knihovnou SimCore.

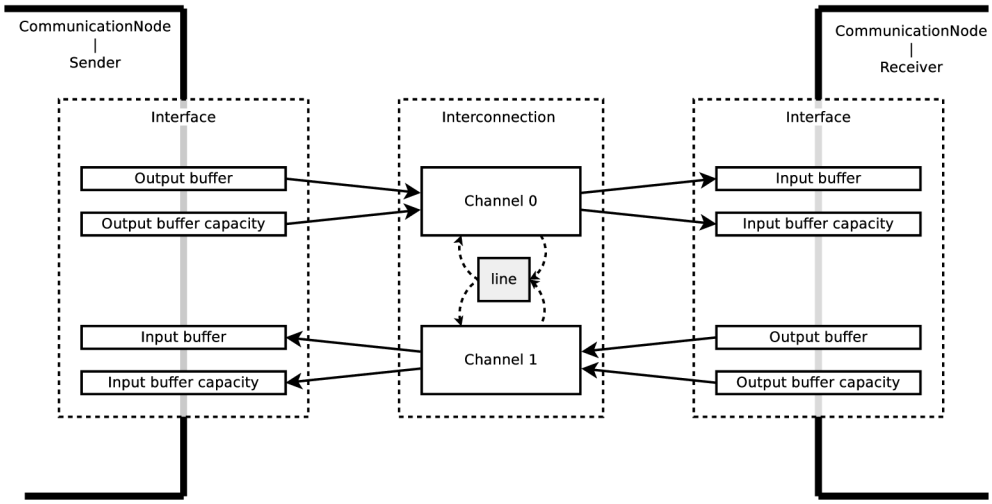
Chování uzlu je popsánou metodou `ACTIONS()`[1]. Při vytváření instance třídy je nutné specifikovat rozhraní objektu. To je v konstruktoru řešeno předáním parametru asociativního pole, jehož klíči jsou názvy rozhraní. Hodnoty pole tvoří další asociativní pole jehož klíči jsou názvy parametrů rozhraní a jejichž hodnoty jsou hodnotami paramterů. V rámci simulačního modelu jsou rozhraní adresována vlastnícím objektem a jménem rozhraní.

Princip činnosti komunikačního uzlu spočívá v příjmu zpráv v daném protokolu, jejich zpracování a případné reakce zasláním zpráv ostaním uzlům. Příjem a odesílání zpráv se děje prostřednictvím vyzvedávání a vkládání zpráv z datových bufferů rozhraní, což je implementováno v prostředí SimPy. Samotný přenos je komunikačnímu uzlu skryt a je tak pro něj transparentní.

### Interface

Rozhraní tvoří přípojný bod komunikačního uzlu a umožňuje tak propojení s ostaními uzly. Každé rozhraní je tvořeno čtyřmi vyrovnávacími buffery, dva jsou použity pro výstup a dva pro vstup. U těchto bufferů lze parametrizovat kapacitu počtem bitů. Objasnění použití čtveřice bufferů je uvedeno u popisu třídy `Channel`.

[1] Název metody definován knihovnou SimPy. Tato metoda je implictně volána při aktivaci procesu před spuštěním simulace.




Obrázek 5.4: Detailní pohled na propojení rozhraní komunikačních uzlů.

### Interconnection

Stěžejní částí knihovny je systém propojů umožňující komunikaci jednotlivých uzlů mezi sebou. Propoj sestává z dvojice komunikačních kanálů, které realizují přenos zpráv mezi propojenými rozhraními (viz třída `Channel`). Nastavení propustnosti propoje (atribut `throughput`) a povolení zahazování zpráv (atribut `drop_enable`) jsou propagovány do komunikačních kanálů.

Propoj standardně umožňuje obousměrnou komunikaci odesilatele a příjemce, což lze ovšem změnit parametrem `direction`, kde se nastaví přenos pouze od odesilatele k příjemci, přenos pouze od příjemce k odesilateli případně přenos oběma směry. Pojmenování účastníků je v tomto případě zavádějící. Vše se snaží vysvětlit obrázek 5.4, který znázorňuje detail propojení dvou rozhraní pomocí propoje. Omezení směru komunikace je řešeno zablokováním jednotlivých komunikačních kanálů.

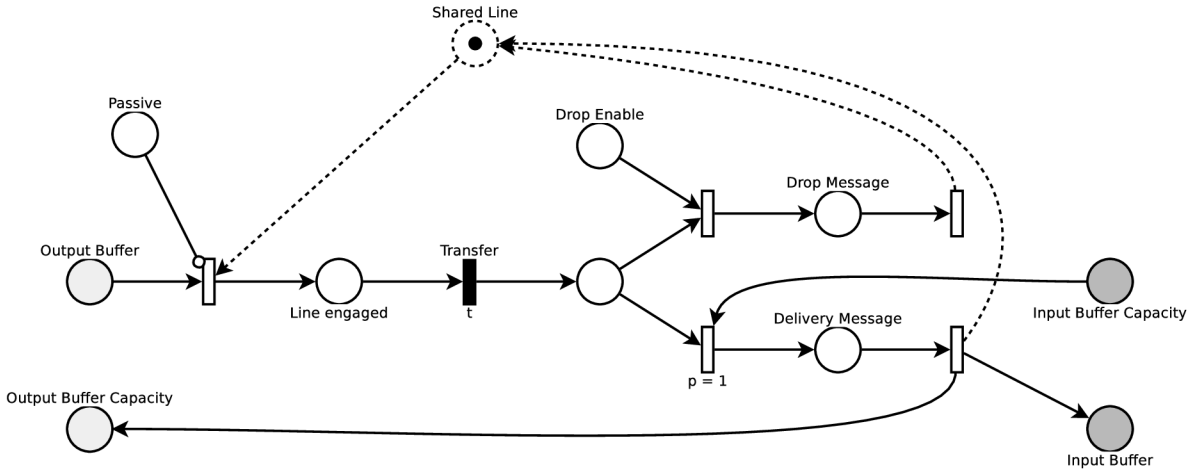
Abstraktní třída `Interconnection` poskytuje jednotné rozhraní sdílené odvozenými třídami `FullDuplexInterconnection` a `HalfDuplexInterconnection`, které, jak je patrné z jejich názvů, implementují plně duplexní a poloviční duplexní spojení. V případě poloviční duplexu přistupují oba komunikační kanály ke sdílené lince (viz obrázek 5.4).

### Channel

Komunikační kanály realizují přenos zpráv mezi propojenými rozhraními. Petriho síť na obrázku 5.5 znázorňuje průběh přenosu zprávy po komunikačním kanále. Simulační proces komunikačního kanálu vyzvedne zprávu z výstupní fronty odesilatele (`output_buffer`). Pokud se jedná o poloviční duplexní propoj, pokusí se zabrat sdílenou linku.

Na základě délky zprávy a propustnosti propoje spočítá délku přenosu a po tuto dobu počká (přechod *Transfer*). Poté se pokusí zabrat kapacitu ze vstupní fronty příjemce (`input_buffer_capacity`). Není-li dostatek dostatek kapacity pro vložení přenášené zprávy a je povoleno zahazování zpráv, pak zprávu zahodí, jinak čeká dokud se neuvolní místo pro přijeté zprávy.

Po zabrání kapacity vloží zprávu do vstupní datové fronty příjemce (`input_buffer`), uvolní kapacitu výstupní fronty odesilatele (`output_buffer_capacity`). Nakonec uvolní




Obrázek 5.5: Petriho síť popisující chování komunikačního kanálu.

sdílenou linku.

Nastavením atributu `passive` lze daný komunikační kanál pasivovat a tím dojde k zablokování přenosu zpráv. Tímto lze dosáhnout povolení určitého směru komunikace v rámci propoje.

Další důležitou vlastností komunikačních kanálů je jejich schopnost zahazovat zprávy, pokud nejsou doručitelné. Toto chování lze nastavit prostřednictvím atributu `drop_enable`. Zpráva je nedoručitelná tehdy, není-li dostatek místa pro její uložení do vstupní fronty na straně příjemce. Simulačnímu modelu postavenému nad knihovnou SimCore je jednodušeno umožněno zahazované zprávy zpracovat ve vlastní režii a to reimplementací metody `drop_message`, které je v parametru předána zahazovaná zpráva.

### Timer

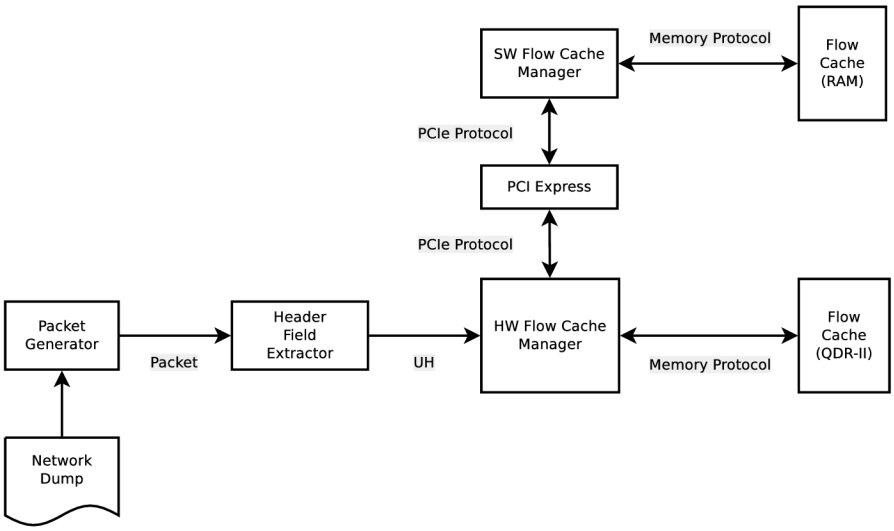
Pomocná třída představující periodický časovač, který po uplynutí stanovené periody (atribut `interval`) provede seznam požadovaných akcí (atribut `tasks`). Časovač byl implementován proto, aby umožnil pravidelné vyčítání stavu komponent simulačního modelu v průběhu simulace. Takto získaný průběh simulace lze poté zobrazit například pomocí grafu nebo tabulky.

Získání údajů se děje mimo režii simulačních komponent, které pouze musí definovat příslušné metody `report()`.

## 5.2 Simulační model

Tato část kapitoly se zabývá návrhem modelu systému stavového zpracování toků. Nejprve je představen koncept univerzálních komponent, které jsou základními stavebními bloky simulačního modelu a jejich využití je obecné. V druhé části jsou pak popsány komponenty specifické pro systém stavového zpracování toků, jímž se tato práce zabývá.

Snahou je vytvořit sadu komponent modelujících základní prvky reálného systému, které půjde mezi sebou snadno propojovat a vytvářet tak komplexní modely. Výsledný simulační model by měl být také snadno konfigurovatelný, aby bylo možno měnit nastavení




Obrázek 5.6: Architektura simulačního modelu včetně komunikačních protokolů.

experimentů a to, pokud možno, bez zásahu do zdrojových kódů. Řešení snadné konfigurovatelnosti je v implementaci modulu `Config`, který obsahuje všechna důležitá nastavení simulačního modelu a jehož editací se dosáhne požadovaných vlastností simulačního experimentu.

Rozdělení modelu na jednotlivé komponenty včetně jejich propojení je znázorněno na obrázku 5.6. Názvy zvýrazněné šedým podbarvením představují použité komunikační protokoly.

### 5.2.1 Komunikační protokoly

Komunikační protokoly použité v simulačním modelu implementovaném v rámci této práce jsou znázorněny v tabulce 5.1. Protokoly využívají obálky na zprávy z knihovny SimCore (viz třída `Message`).

| **Položka zprávy** | **Protokol** | |
|---|---|---|
| | *Packet* | *UH* |
| ID | pořadové číslo paketu | ID toku |
| Type | `packet` | `uh` |
| Data | paket | unifikovaná hlavička |
| Length | délka paketu v bitech | délka unifikované hlavičky |
| **Položka zprávy** | **Protokol** | |
| | *Memory Protocol* | *PCIe Protocol* |
| ID | adresa | závislé na neseném protokolu |
| Type | `mem_write`, `mem_read` | závislé na neseném protokolu |
| Data | flow záznam, prázdné | závislé na neseném protokolu |
| Length | délka flow záznamu v bitech | závislé na neseném protokolu |

Tabulka 5.1: Přehled komunikačních protokolů.

Protokol *PCIe Protocol* je množinou protokolů, které jsou přenášeny přes systémovou sběrnici PCI Express. Hodnoty položek zprávy jsou proto závislé na neseném protokolu.

### 5.2.2 Základní komponenty simulačního modelu

Následuje popis základních komponent simulačního modelu znázorněných na obrázku 5.6. Komponenty *HW Flow Cache Manager* a *SW Flow Cache Manager* jsou specifické pro konkrétní klasifikační metody.

#### Packet Generator

Tato komponenta čte pomocí knihovny OTA soubor obsahující zachytaný síťový provoz (například pomocí programu `tcpdump`). Z přečtených paketů je vybrán pouze IPv4 provoz s TCP nebo UDP protokolem transportní vrstvy, což představuje provoz zpracovávaný systémem stavového zpracování toků. Pakety jsou zabaleny do zprávy protokolu *Packet*, kde je délka zprávy je nastavena na délku paketu v bitech a jako identifikátor zprávy je použito pořadové číslo paketu. Takto vytvořené zprávy jsou předány na výstupní rozhraní generátoru.

Komponenta ukládá základní statistické údaje o počtu přečtených paketů, počtu odfiltrovaných paketů a počtu odeslaných zpráv.

#### Header Field Extractor

Provádí extrakci položek z hlaviček paketů. Z příchozích paketů vyčte údaje potřebné pro sestavení klíče toku, což jsou zdrojová a cílová IP adresa, číslo zdrojového a cílového portu a typ transportního protokolu, a některé další údaje nezbytné pro vytvoření záznamu o toku (např. časová značka a délka paketu v bajtech). Výsledek je zabalen do zprávy odpovídající protokolu *UH* a vložen na výstupní rozhraní.

Jako statistické údaje uchovává tato komponenta počet zpracovaných paketů.

#### Flow Cache

Modelování paměti ukládající záznamy o tocích. Komunikace s pamětí probíhá na základě protokolu *Memory Protocol* popsaného výše. Pamět je chápána jako lineární homogení datová struktura, pro kterou lze nastavit její kapacitu v bitech, velikost položky, šířku datové sběrnice, periodu řídicího hodinového signálu a počet taktů nutný ke zpracování požadavku. Poslední tři parametry jsou použity pro výpočet latence zpracování požadavku.

Takto definovaná komponenta se snaží co nejvíce přiblížit chování reálné fyzické paměti a tím je dosaženo její univerzálnosti. V rámci simulačního modelu implementovaného v této práci je použití paměti velmi zjednodušeno, přičemž není uvažována latence paměti. Ignorování latence paměti je umožněno díky komplexnímu systému rychlých vyrovnávacích pamětí a ukládání seznamu rozpracovaných položek, který překlenuje dobu přístupu do paměti. Operace čtení a zápisu se pak pro proces přistupující do paměti jeví jako okamžité.

Komponenta ukládá počty operací čtení a zápisu. Dalším statistickým údajem je počet platných položek uložených v paměti.

#### PCI Express

Komponenta modeluje systémovou sběrnici PCI Express, která je charakterisktická svojí datovou propustností a latencí. Latence sběrnice je velmi závislá na použitém hardware a na

mnoha dalších okolnostech jako je například aktuální vytížení procesoru. Přesné stanovení délky latence tedy není možné a proto byl v rámci této práce proveden odhad latence na $4\,\mu$s.

Komunikace po sběrnici je řízena protokolem *PCIe Protocol*, který je vlastně množinou protokolů specifických pro konkrétní simulační model. Upřesnění této množiny je uvedeno níže u popisu modelů jednotlivých klasifikačních metod.

Dostupné stavové informace o komponentě jsou počty zpráv a počty bitů přenesených každým směrem.

# Kapitola 6

# Závěr

Cílem této práce byl návrh systému stavového zpracování toků využívajícho rozdělení paměti flow cache na hardwarovou a softwarovou část. Hlavní důraz byl kladen na analýzu systému s pomocí simulačního modelu, který byl v základní podobě implementován. Mezi hlavní přednosti navrženého simulačního modelu patří jeho rozšiřtelnost, jednoduché nastavení simulačního prostředí pomocí konfiguračního modulu a možnost použití základních komponent jako stavebních bloků pro sestavení komplexnějších modelů.

Pro podporu modelování hardwarových komponent byla navržena a implementována univerzální simulační knihovna SimCore s využitím objektově orientovaného prostředí pro diskrétní simulaci SimPy [8] v jazyce Python. Tato knihovna umožňuje popsat modelovaný systém množinou vzájemně propojených entit, které spolu komunikují zasíláním zpráv. Tento princip vychází z konceptu hardwarových komponent propojených vodiči.

V rámci vypracování práce byly nastudovány odborné články zaměřené na klasifikaci síťových toků [15] a identifikaci silných toků pomocí paměti cache implementované technologií FPGA [24]. Pro lepší pochopení systémů stavového zpracování toků byla prostudována bakalářská práce řešící implementaci tohoto systému na platformě karet COMBO [17]. Podrobnější informace o počítačové komunikaci a síťových protokolech byly získány prostudováním [2], [13] a [21]. Pro seznámí s vysokorychlostními páteřními linkami bylo čerpáno z odborných článků organizace CAIDA [1]. Teoretické znalosti byly využity jak při psaní úvodní části práce, tak při návrhu architektury simulačního modelu.

Tato práce skýtá možnosti rozšíření z hlediska implementace navrženého systému stavového zpracování toků. Dalším možným pokračováním je vylepšení knihovny SimCore z hlediska jejího univerzálního použití. Jinou možností je rozšíření základní množiny komponent simulačního modelu a tím poskytnutí nástroje pro podporu modelování systémů vyvýjených na platformě COMBO karet.

# Literatura


[1] CAIDA. The Cooperative Association for Internet Data Analysis. http://www.caida.org.

[2] Cisco: Internetworking Basics. http://www.cisco.com/en/US/docs/internetworking/technology/handbook/Intro-to-Internet.html, [Online; cit. 26.4.2010].

[3] Cypress. Webové stránky výrobce QDR–II pamětí. http://www.cypress.com.

[4] Liberouter: Stránka věnovaná rodině karet COMBO. http://www.liberouter.org/hardware.php?flag=1, [Online; cit. 25.4.2010].

[5] Stránky projektu Liberouter. http://www.liberouter.org, [Online; cit. 25.4.2010].

[6] Stránky projektu MAWI (Measurement and Analysis on the Wide Internet). http://www.wide.ad.jp/project/wg/mawi.html, [Online; cit. 17.5.2010].

[7] Webové stránky organizace IANA, Internet Assigned Number Authority. http://www.iana.org.

[8] Webové stránky projektu SimPy, Simulation in Python. http://simpy.sourceforge.net.

[9] Webové stránky sdružení CESNET. http://www.cesnet.cz.

[10] Xilinx. Webové stránky výrobce FPGA čipů. http://www.xilinx.com.

[11] IEEE Standard 802.1q, Virtual bridged Local Area Networks. http://standards.ieee.org/getieee802/download/802.1Q-2005.pdf, 2006, [Online; cit. 26.4.2010].

[12] IEEE Standard 802.3, Carrier sense multiple access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications. http://standards.ieee.org/getieee802/download/802.3-2008_section1.pdf, 2008, [Online; cit. 26.4.2010].

[13] BRANDEN, R.: RFC1122 – Requirements for Internet Hosts – Communication Layers. http://www.ietf.org/rfc/rfc1122.txt, 1989, [Online; cit. 26.4.2010].

[14] DEERING, S.; HINDEN, R.: RFC 2460 – Internet Protocol, Version 6 (IPv6) Specification. http://www.ietf.org/rfc/rfc2460.txt, 1998, [Online; cit. 26.4.2010].

[15] ESTAN, C.; VARGHESE, G.: New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. *ACM Trans. Comput. Syst.*, ročník 21, č. 3, 2003: s. 270–313, ISSN 0734-2071, [Online; cit. 15.5.2010].

[16] KAPLAN, J.; SINGH, P.; O'DELL, M.; aj.: Extended Ethernet Frame Size Support. http://tools.ietf.org/html/draft-ietf-isis-ext-eth-01, 2001, [Online; cit. 26.4.2010].

[17] KOŠEK, M.: *Stavové zpracování TCP/IP toků*. Diplomová práce, Vysoké učení technické v Brně, Fakulta informačních technologií, 2007, vedoucí diplomové práce Ing. Jan Kořenek.

[18] KUROSE, J. F.; ROSS, K. W.: *Computer Networking: A Top-Down Approach Featuring the Internet*. Boston: Addison-Wesley, druhé vydání, 2003, ISBN 0-321-17644-8.

[19] NICHOLS, K.; BLAKE, S.; BAKER, F.; aj.: RFC 2474 – Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers. http://www.ietf.org/rfc/rfc2474.txt, 1998, [Online; cit. 26.4.2010].

[20] POSTEL, J.: RFC 768 – User Datagram Protocol. http://www.ietf.org/rfc/rfc768.txt, 1980, [Online; cit. 26.4.2010].

[21] POSTEL, J.: RFC 791 – Internet Protocol. http://www.ietf.org/rfc/rfc0791.txt, 1981, [Online; cit. 26.4.2010].

[22] POSTEL, J.: RFC 793 – Transmission Control Protocol. http://www.ietf.org/rfc/rfc793.txt, 1981, [Online; cit. 26.4.2010].

[23] SOĽANKA, L.: *Design of probe for flow based monitoring*. Diplomová práce, Vysoké učení technické v Brně, Fakulta informačních technologií, 2009, vedoucí diplomové práce Ing. Jan Kořenek.

[24] ŽÁDNÍK, M.; CANINI, M.; Moore, A. W.; aj.: Tracking Elephant Flows in Internet Backbone Traffic with an FPGA-based Cache. 2009.

# Dodatek A

# Obsah CD

K bakalářské práci je přiloženo CD obsahující elektronickou verzi textu technické zprávy a také zdrojové soubory simulačního modelu a simulační knihovny implementovaných v rámci této práce.